On Unsupervised Learning of Probabilistic Graphical Models

Jens Dalgaard Nielsen
Department of Computer Science
Aalborg University
Denmark
dalgaard@cs.aau.dk

July 2, 2007

Preface

This dissertation is the result of my Ph.D. study at the Department of Computer Science at Aalborg University Denmark, from August 2003 to November 2006. The study has been focused on the development of algorithms for automatic learning of probabilistic graphical models from data. Specifically, I have focused on algorithms for learning Bayesian Network models, Probabilistic Decision Graph models and Naïve Bayes models. I report on experiments of learning models both from real and synthetic data. I perform a comparative analysis of the three languages and their performance w.r.t. computational efficiency and accuracy of the approximations offered.

Acknowledgements

During my study I have received the necessary financial funding from Ph.D. stipend No. 562/06–16–23603 granted to me by the Faculty of Technology and Engineering Sciences at Aalborg University.

This dissertation is the result of many different collaborations, and I could never have achieved the results reported here without the support and help that I have been very fortunate to receive.

First and foremost, I wish to thank my adviser Manfred Jaeger who always gave me very encouraging and extremely competent advise. I have benefited greatly from his numerous constructive and always very honest comments on my work. He made it possible for me to get my work published, and he taught me how to present complex material in the condensed form of a research paper. For his thorough reviews of preliminary versions of this dissertation — which he did in his spare time — I am especially thankful.

I wish to thank Jose M. Peña and Tomáš Kočka for the fruitful collaboration we had at the Decision Support Systems group at Aalborg University. This collaboration resulted in my first publication, a publication that would never have been possible without the insight and ideas Jose and Tomáš shared with me. They both were always willing to help me understand many difficult issues concerning learning of Bayesian Network models, which was a new topic for me.

I would like to thank Tomi Silander for the collaborations we have had. Even though we only communicated through email, the collaboration resulted in two publications. I still hope to see you face to face one day.

Finn Verner Jensen was supportive and encouraging during my study. I especially thank him for hiring me for the position of research assistant and, thereby, providing me with a perfect opportunity to get a taste of research before deciding on whether to apply for a PhD. stipend.

From March to August 2005, I spend 5 month as a visiting scholar at the AutonLab, Carnegie Mellon University in Pittsburgh (PA) USA. I wish to thank Andrew W. Moore for our many inspiring discussions and for his very enthusiastic and encouraging attitude towards my work and my ideas. I wish to thank all the brilliant members of the AutonLab for making

me feel as a member of your little family right away. I wish to thank all of my Pittsburgh friends who made my stay extremely enjoyable — I never felt alone in the Steel City.

I wish to thank the very competent group of secretaries at the Department of Computer Science at Aalborg University. They made my daily encounters with the university bureaucracy much less painful.

Also, I wish to thank all the people in the Machine Intelligence group at the Department of Computer Science, Aalborg University for making my time there so enjoyable, in particular my office-mate Søren Holbech Nielsen with whom I had numerous weird discussions on random current events. I especially liked our white-board-pillory, where we would hold any person (famous or not) up to ridicule if he/she so deserved.

I thank the members of my committee Marek Druzdzel, Antonio Salmerón and Thomas D. Nielsen for valuable review comments on the first version of this dissertation. The comments were very useful in my preparation of this final version.

Finally, I wish to thank my friends and family for showing me sincere and loving support at all times — and especially when I needed it most in the last stages of the writing process.

Jens Dalgaard Nielsen
Almería, July 2, 2007

Contents

Chapter 1

Introduction

Probabilistic graphical models (PGMs) is a mathematical framework for representing joint probability distributions over sets of random variables (Cowell et al., 1999; Jensen, 2001; Lauritzen, 1996; Pearl, 1988). PGMs have become a standard approach for representation and handling of uncertainty in the field of Artificial Intelligence. Also in the related fields of Pattern Recognition and Machine Learning, PGMs have received a lot of attention and have been applied with success in numerous domains (Bishop, 2006; Mitchell, 1997; Duda et al., 2001).

When PGMs are learnt from data (as opposed to being manually constructed), some score function is used to assess the quality of models and, thereby, discriminate between alternatives. The learning procedure then selects from amongst alternative models the one that is optimal w.r.t. the score function. A typical score-function combine in a weighted sum a reward for accuracy (computed w.r.t. a database) and a penalty for complexity. In general, we call such score-functions for penalised likelihood scores, and they take the following simple form:

$$S(M, \mathcal{D}) = \lambda \cdot L(\mathcal{D}|M) - (1 - \lambda) \cdot size(M), \tag{1.1}$$

for PGM M, data $\mathcal{D}$, likelihood L, and some trade-off coefficient $0 < \lambda < 1$. Typically, the number of different alternative models is much too big to allow exhaustive search, and studies have shown that many instances of learning tasks for PGMs are NP-hard (Chickering et al., 2004; Chickering, 1996). Consequently, heuristic procedures are appropriate and often necessary in practise.

The study reported in this dissertation has focused on aspects of learning PGMs from data. In the following, we will briefly discuss the problems addressed and the solutions proposed.

One of the most popular types of PGMs is the Bayesian Network (BN) (Pearl, 1988; Jensen, 2001). The learning of BN models has received much attention and both discouraging and encouraging results have been found. While it has been proved that the problem of learning BN models that optimise (1.1) is NP-hard (Chickering et al., 2004), learning procedures that recover the optimal BN models have been shown to be tractable for many relevant domains (the SGS algorithm (Spirtes et al., 2000) and the GES algorithm (Chickering and Meek, 2002;

Meek, 1997)). These learning procedures, however, rely on the strong assumption that the data generating process that exhibits independence relations between the observed variables that can be encoded in the directed acyclic graph (DAG) structure of the BN model, that is, the process exhibits DAG faithfulness. This assumption is often unrealistic in real world applications, and the quality of the models that are learnt may be very dependent on this assumption being satisfied. Therefore, the practical applicability of such learning procedures may be limited.

In this dissertation we propose a simple generalisation of a greedy search procedure. The generalisation introduces a parameter for trading off greediness for randomness in the decision-rule guiding the search. By employing multiple restarts in connection with stochastic decision rule, the algorithm maintains the theoretical optimality of greedy search, and, in addition, it allows a broader exploration of the search space. This is important when the strong assumption of a DAG faithful generative distribution is violated. In this case, the deterministic search implemented by a greedy decision rule may lead to a suboptimal model while a multiple restart stochastic search will identify multiple local optimal models.

In most application areas, one of the main tasks for PGMs is to provide a representation that allows for efficient belief updating. By belief updating we understand the process of computing all posterior marginal probability distributions for all variables in the domain given observations of a subset of variables. For BN models this task is NP-hard (Cooper, 1987). Often, however, it is possible to obtain a computationally tractable BN model that still offers a sufficiently accurate approximation. On the other hand, example distributions can be constructed where any model less complex than the maximally complex model will be unable to approximate the distribution accurately (Jaeger, 2004; Beygelzimer and Rish, 2003). Such challenging examples are constructed by defining distributions that contains *context-specific* (in)dependence (CSI) relations, also sometimes called *asymmetric* (in)dependencies. The existence of CSI relations not representable by the BN model has motivated the development of extensions to the BN model that are able to efficiently represent such distributions. Examples include the Bayesian Multinets (BM) by Geiger and Heckerman (1996), Mixtures of Bayesian Networks (MBN) by Thiesson et al. (1997) and Recursive Bayesian Multinets (RBM) by Peña et al. (2002). These are all variations of the following common architecture: a context is defined by a (set of) distinguished variable(s), and conditioned on the context, a BN representation over the remaining variables is selected. For MBNs the context is defined by a non-observed latent variable, and for RBMs the context is defined by a set of observed variables. Inference algorithms in these models can benefit from the CSI relations encoded by the model, but ultimately the inference complexity of BN models persists.

In this dissertation we propose a procedure for learning of Probabilistic Decision Graph (PDG) models. The PDG language is a recent addition to the growing set of PGM representation language for discrete joint probability distributions (Jaeger, 2004). PDGs offer both a natural encoding of a certain class of CSI relations between the observable variables and also offers efficient belief updating in the presence of evidence. One particularly welcoming property of the PDG language is that the representation structure is itself a primary structure

for efficient computations of general belief updating. This is important for learning procedures when the learnt models are expected to offer efficient belief updating. In this scenario, we can then readily discriminate between models w.r.t. computational complexity of belief updating from the given representation. Retrieving a meaningful measure of computational complexity is troublesome for many other relevant PGM languages — in particular for BN models, where determining the computational complexity of a model involves an NP-complete optimisation problem (Arnborg et al., 1987).

It is often necessary to assume data to be complete in the sense that no latent (non-observed) variables influences the observed variables through non-trivial interactions. However, this is often a very strong assumption and may not be consistent with the understanding provided by domain experts. The existence of such latent variables may yield a data generating process that exhibits a set of independence relations that is not representable by the DAG structure of BN models. Recovering the existence of such latent variables explicitly is an ambitious task. Nevertheless, many recent studies have pursued a solution to the problem of learning latent variables both in a general DAG structured BN model (Elidan, 2004) and when focusing on hierarchical (tree) structures (Karčiauskas, 2005). A tree-structured BN model that models all observed variables conditionally independent given the state of a single latent variable (usually denoted a Naïve Bayes (NB) model), is well studied for probabilistic *soft* clustering of data instances (Duda et al., 2001). However, such models can also just as easy and naturally be used for general computation of probabilistic inference tasks. Recent studies have shown encouraging results favouring the NB model when comparing NB to BN models w.r.t. computational complexity and accuracy of the approximation offered (Lowd and Domingos, 2005).

In this dissertation we perform a comparative analyses of different PGM languages, their ability to efficiently and accurately approximate distributions and our ability to learn such approximations from a finite data sample. Such analyses are not new, and we therefore augment the analyses performed in previous studies such as the comparative analyses of empirical measurements of efficiency and accuracy of BN and NB models by Lowd and Domingos (2005) and the more theoretical study of the range of different approximations offered by BN models by Beygelzimer and Rish (2003). First, in our analysis we employ the analytical tool of SL-curves. SL-curves show language characteristics by plotting efficiency and accuracy of models from the language. For efficiency we use a measure of computational complexity which is, therefore, a theoretical quantity, while for accuracy we use the likelihood of the data given the model. Second, we perform an empirical analysis of computational efficiency using implementations of state-of-the-art algorithms for probabilistic inference. We also include a comparison of accuracy measured empirically by averaging over randomly generated queries. Third, we include the novel PGM language of PDGs in the comparative analyses.

Finally, as a somewhat separate issue, we propose an algorithm that constructs a PDG model from a Clique Tree (CT) representation of a distribution. We combine BN learning and PDG learning by constructing a CT representation of the distribution represented by the BN

model, then translating this CT into an equivalent PDG model that is then exposed to optimisation operations that may yield a representation that is competitive with the original BN model. We denote this approach "hybrid learning" of PDG models as it combines a learnt BN model and its CT representation with learning a refined and optimised PDG representation.

1.1 Outline of The Dissertation

In Chapter 2 we give an introduction to relevant background concepts and basic notational conventions used in the remainder of this dissertation. Chapter 3 introduces formally the PGM representation languages that we investigate in the later analysis. We include discussions on computational complexity of general probabilistic inference by presenting for each language procedures for performing exact belief updating. In Chapter 4, we propose procedures for learning models from data for each of the PGM languages presented earlier. Chapter 5 contains a description of experiments on learning PGMs from data, and we perform both theoretical and empirical comparative analyses of the PGM languages using the proposed learning procedures. In addition, Chapter 5 contains an analysis of hybrid learning of PDG models.Finally, in Chapter 6 we summarise important observations made from the comparative analyses and discuss the conclusions that can be drawn from the study reported in this dissertation.

Chapter 2

Preliminaries and Notation

2.1 Probability theory

In this section we introduce probability theory as a framework for handling uncertainty. We will limit this introduction to concepts that are of particular relevance to the study reported in this thesis. For a complete formal introduction to the field of probability theory, the reader may consult the books of DeGroot (1986) and Billingsley (1986). Also Hájek (2003) and the references found there should be mentioned as an excellent review of different interpretations of probability theory.

Let $\mathbf{\Omega}$ be an arbitrary set where each element $\omega \in \mathbf{\Omega}$ represents a possible state of nature. An event is a subset of $\mathbf{\Omega}$, and an event space $\mathcal{R}$ w.r.t. $\mathbf{\Omega}$ is a non-empty set of events including $\mathbf{\Omega}$ that is closed under the operations of complement and finite union, and therefore also closed under finite intersection as $A \cap B = \overline{\overline{A} \cup \overline{B}}$. In measure theory, the pair $\langle \mathbf{\Omega}, \mathcal{R} \rangle$ is called a *measurable* space. A real-valued function P on $\mathcal{R}$ is a probability measure on $\mathbf{\Omega}$ when P satisfies the basic axioms of probability (Kolmogorov, 1950):

Axiom 2.1 (Non-negativity)
$P(E) \geq 0$, *for all* $E \in \mathcal{R}$.

Axiom 2.2 (Normalisation)
$P(\mathbf{\Omega}) = 1$.

Axiom 2.3 (Finite additivity)
For any sequence of disjoints $E_1, E_2, \ldots, E_n \in \mathcal{R}$

$$P(\cup_{i=1}^{n} E_i) = \sum_{i=1}^{n} P(E_i). \tag{2.1}$$

From these axioms it follows that $P(E) = 1 - P(\mathbf{\Omega} \setminus E)$, $P(\emptyset) = 0$ and $P(E) \leq 1$ for all $E \in \mathcal{R}$.

The triple $\langle \mathbf{\Omega}, \mathcal{R}, P \rangle$ is called a probability space. We usually think of the probability of event E (denoted $P(E)$) as the likelihood that E will occur, where E occurs if the current state of nature $\omega \in \mathbf{\Omega}$ is included in E.

$\mathbf{\Omega}$ is sometimes viewed as the set of all possible outcomes of some experiment. Some schools of probability theory (e.g., frequentists) requires experiments to be (in principle) repeatable in order to assign a probability measure to an event space. The probability of an event E is then defined as the limiting relative frequency with which E occurs:

$$P(E) := \lim_{n \to \infty} \frac{N_E}{N}, \tag{2.2}$$

where N is the number of times the experiment has been performed and N_E is the number of times E has occurred. From this definition, a probability $P(E)$ is an objective measure.

Other schools of probability theory (e.g., Bayesians) do not require experiments to be repeatable in order to talk about probabilities of events. For instance, when we talk about the probability of our local soccer club winning the national league this year, we are not able to establish this number by repeated experiments. Instead, we have to come up with some number that sounds "right" to us, so this will be a subjective measure. Proponents of subjective probabilities usually term a persons subjective probability as this persons belief. Your belief in some event E can be determined by having you set a price of a bet of 1 € on whether E occurs or not. You must set the price x while not knowing whether you will have to sell or buy the bet. That is, if I decide to buy the bet from you for the price of x €, you will have to pay me 1 € in the case that E occurs, and otherwise pay me nothing (and, thereby, earn the x €). The value of x for which you are indifferent of whether to buy or sell the bet is your belief in E. When x is selected such that one is not expose to certain loss against a prudent opponent with the same prior knowledge, beliefs will satisfy Axiom s2.1-2.3, see (Bernardo and Smith, 1994; Skyrms, 1984).

2.1.1 Random Variables

Given a probability space $\langle \mathbf{\Omega}, \mathcal{R}, P \rangle$, a *discrete random variable* X is a mapping:

$$X : \mathbf{\Omega} \to R(X), \tag{2.3}$$

where $R(X)$ is a finite set of states. When X is defined w.r.t. probability space $\langle \mathbf{\Omega}, \mathcal{R}, P \rangle$, we require for each $x \in R(X)$, $\{\omega \in \mathbf{\Omega} : X(\omega) = x\} \in \mathcal{R}$, and define the *probability* of discrete random variable X being in state x as:

$$P(X = x) := P(\{\omega \in \mathbf{\Omega} : X(\omega) = x\}). \tag{2.4}$$

We denote by $P(X)$ the *probability distribution* or *probability mass function* of variable X, which is then a function on $R(X)$.

From the basic axioms of probability, it follows that $P(X)$ satisfies:

1. $0 \leq P(X = x) \leq 1$ for all $x \in R(X)$, and
2. $\sum_{x \in R(X)} P(X = x) = 1$.

Let $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ be a set of discrete random variables w.r.t. probability space $\langle \mathbf{\Omega}, \mathcal{R}, P \rangle$. $\mathbf{X}$ then defines a mapping from $\mathbf{\Omega}$ to $R(\mathbf{X})$, where $R(\mathbf{X}) = \times_{X_i \in \mathbf{X}} R(X_i)$:

$$\mathbf{X} : \mathbf{\Omega} \rightarrow R(\mathbf{X}). \tag{2.5}$$

As $\mathcal{R}$ is closed under finite intersection, it follows that $\{\omega \in \mathbf{\Omega} : \mathbf{X}(\omega) = \mathbf{x}\} \in \mathcal{R}$ for any $\mathbf{x} \in R(\mathbf{X})$. We can define the *joint probability* of $\mathbf{X}$ being in joint state $\mathbf{x}$ as:

$$P(\mathbf{X} = \mathbf{x}) := P(\{\omega : \omega \in \mathbf{\Omega}, \mathbf{X}(\omega) = \mathbf{x}\}). \tag{2.6}$$

We will use the notation $P(\mathbf{X})$ to refer to the *joint probability distribution* of random variables $\mathbf{X}$ which is then a function on $R(\mathbf{X})$.

From the basic axioms of probability, it follows that $P(\mathbf{X})$ satisfies:

1. $0 \leq P(\mathbf{X} = \mathbf{x}) \leq 1$ for all $\mathbf{x} \in R(\mathbf{X})$, and
2. $\sum_{\mathbf{x} \in R(\mathbf{X})} P(\mathbf{X} = \mathbf{x}) = 1$.

A set of random variables $\mathbf{X}$ is therefore equivalent to a single random variable with state space $R(\mathbf{X})$.

Let $\mathbf{X}$ be a set of random variables, $\mathbf{Y} \subseteq \mathbf{X}$, and $\mathbf{x} \in R(\mathbf{X})$. Then we denote by $\mathbf{x}[\mathbf{Y}]$ the projection of $\mathbf{x}$ onto variables $\mathbf{Y}$. Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ be a set of discrete random variables, and let $P(\mathbf{X})$ be a distribution for $\mathbf{X}$. We can derive the *marginal* distribution for a subset $\mathbf{Y} \subset \mathbf{X}$ by marginalisation, which amounts to summing over all joint states of $R(\mathbf{X} \setminus \mathbf{Y})$:

$$P(\mathbf{Y} = \mathbf{y}) = \sum_{\substack{\mathbf{x}:\mathbf{x} \in R(\mathbf{X}) \\ \text{and } \mathbf{x}[\mathbf{Y}] = \mathbf{y}}} P(\mathbf{X} = \mathbf{x}). \tag{2.7}$$

2.1.2 Conditional Distributions

Let $\mathbf{X}$ be a set of random variables w.r.t. probability space $\langle \mathbf{\Omega}, \mathcal{R}, P \rangle$, and let $\mathscr{A}$ be a partition of $\mathbf{\Omega}$ into the k collectively exhaustive and mutually exclusive sets $\{A_1, A_2, \ldots, A_k\}$ where $A_l \in \mathcal{R}$ for $1 \leq l \leq k$. We can construct the *joint conditional probability* or *joint posterior probability* of $\mathbf{X}$ being in joint state $\mathbf{x}$ given some A_l $(1 \leq l \leq k)$, as:

$$P(\mathbf{X} = \mathbf{x} | A_l) = \frac{P(A_l \cap \{\omega \in \mathbf{\Omega} : \mathbf{X}(\omega) = \mathbf{x}\})}{P(A_l)}. \tag{2.8}$$

As $\mathcal{R}$ is closed under intersection it is clear that $A_l \cap \{\omega \in \mathbf{\Omega} : \mathbf{X}(\omega) = \mathbf{x}\} \in \mathcal{R}$, and therefore P is defined on the intersection. However, equation (2.8) requires the denominator $P(A_l)$ to be non-zero for $P(\mathbf{X} = \mathbf{x}|A_l)$ to be defined, and we will leave the conditional probability undefined when the condition A_l has zero probability[1].

We denote by $P(\mathbf{X}|\mathscr{A})$ the *joint conditional distribution* of $\mathbf{X}$ given partition $\mathscr{A}$ which is a function on $R(\mathbf{X}) \times \mathscr{A}$.

[1] While intuitively it may not make much sense to allow conditioning on the impossible event (that is, a partitions of zero probability), it is allowed within certain formalisations of probability theory such as that of De Finetti.

There are several ways that one can define a partitioning of $\mathbf{\Omega}$. Given set a of variables $\mathbf{X}$, we can define a partition $\mathscr{A}$ in terms of $R(\mathbf{X})$. That is, a partitioning $\mathscr{R} = \{R_0, \ldots, R_k\}$ of $R(\mathbf{X})$ immediately defines the partitioning $\mathscr{A} = \{A_0, \ldots, A_k\}$ of $\mathbf{\Omega}$: $A_i = \{\omega \in \mathbf{\Omega} : \mathbf{X}(\omega) = \mathbf{x} \text{ and } \mathbf{x} \in R_i\}$. Then we get a special case of eq. (2.8):

$$P(\mathbf{X} = \mathbf{x}|A_l) = \begin{cases} \frac{P(\mathbf{X}=\mathbf{x})}{\sum_{\mathbf{x}' \in R_l} P(\mathbf{X}=\mathbf{x}')} & \text{if } \mathbf{x} \in R_l, \\ 0 & \text{otherwise.} \end{cases} \tag{2.9}$$

A partitioning that is often used is the one induced by a subset of variables $\mathbf{Z} \subseteq \mathbf{X}$, such that every joint state $\mathbf{z}_l \in R(\mathbf{Z})$ induces $A_l = \{\omega \in \mathbf{\Omega} : \mathbf{Z}(\omega) = \mathbf{z}_l\}$. Then we get yet another special case of eq. (2.8):

$$P(\mathbf{X} = \mathbf{x}|A_l) = \begin{cases} \frac{P(\mathbf{X}=\mathbf{x})}{P(\mathbf{Z}=\mathbf{z}_l)} & \text{if } \mathbf{x}[\mathbf{Z}] = \mathbf{z}_l, \\ 0 & \text{otherwise.} \end{cases} \tag{2.10}$$

We will denote by $P(\mathbf{X}|\mathbf{Z})$ the *joint conditional distribution* of $\mathbf{X}$ given the partition of $\mathbf{\Omega}$ induced by $\mathbf{Z}$, which is then a function on $R(\mathbf{X}) \times R(\mathbf{Z})$.

By suitable marginalisation and recursive application of (2.10) one can construct the following factorisation of a joint distribution $P(\mathbf{X})$ over a set of variables $\mathbf{X}$:

$$P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|X_{i+1}, \ldots, X_n). \tag{2.11}$$

2.1.3 Independence

Let $\mathbf{X}$ be a set of random variables defined on probability space $\langle \mathbf{\Omega}, \mathcal{R}, P \rangle$. We say that disjoint subset of random variables $\mathbf{W}, \mathbf{Y} \subset \mathbf{X}$ are marginally independent under P iff:

$$\forall \mathbf{w} \in R(\mathbf{W}), \forall \mathbf{y} \in R(\mathbf{Y}) : P(\mathbf{W} = \mathbf{w}, \mathbf{Y} = \mathbf{y}) = P(\mathbf{W} = \mathbf{w})P(\mathbf{Y} = \mathbf{y}), \tag{2.12}$$

and we will use notation $\mathbf{W} \perp\!\!\!\perp \mathbf{Y}[P]$ to denote this relation.

Let $\mathscr{A} = \{A_1, \ldots, A_k\}$ be a partition of $\mathbf{\Omega}$. We then say that $\mathbf{W}$ and $\mathbf{Y}$ are conditionally independent given $\mathscr{A}$ under P iff:

$$\forall \mathbf{w} \in R(\mathbf{W}), \forall \mathbf{y} \in R(\mathbf{Y}), \forall A_l \in \mathscr{A} : \\ P(\mathbf{W} = \mathbf{w}, \mathbf{Y} = \mathbf{y}|A_l) = P(\mathbf{W} = \mathbf{w}|A_l)P(\mathbf{Y} = \mathbf{y}|A_l). \tag{2.13}$$

We will use $\mathbf{Y} \perp\!\!\!\perp \mathbf{W}|\mathscr{A}[P]$ to denote this relation. Marginal independence is just a special case of conditional independence where the conditioning partition is the trivial partitioning $\mathscr{A} = \{\mathbf{\Omega}\}$.

When the partitioning of $\mathbf{\Omega}$ is induced by a subset of variables $\mathbf{Z}$ disjoint from $\mathbf{Y}$ and $\mathbf{W}$, we will write $\mathbf{Y} \perp\!\!\!\perp \mathbf{W}|\mathbf{Z}[P]$ to denote that $\mathbf{Y}$ and $\mathbf{W}$ are conditionally independent given the state of $\mathbf{Z}$ under joint distribution P. Equation (2.13) can then be rewritten as:

$$\forall \mathbf{w} \in R(\mathbf{W}), \forall \mathbf{y} \in R(\mathbf{Y}), \forall \mathbf{z} \in R(\mathbf{Z}) :$$
$$P(\mathbf{W} = \mathbf{w}, \mathbf{Y} = \mathbf{y} | \mathbf{Z} = \mathbf{z}) = P(\mathbf{W} = \mathbf{w} | \mathbf{Z} = \mathbf{z}) P(\mathbf{Y} = \mathbf{y} | \mathbf{Z} = \mathbf{z}). \quad (2.14)$$

We allow the conditioning set $\mathbf{Z}$ of variables to be empty, but using the notation $\mathbf{Y} \perp\!\!\!\perp \mathbf{W} | \emptyset [P]$ is confusing as $P(\emptyset) = 0$, and (2.14) would not be defined. However, $\mathbf{Z} = \emptyset$ generates the trivial partitioning $\mathscr{A} = \{\mathbf{\Omega}\}$, and instead of $\mathbf{Y} \perp\!\!\!\perp \mathbf{W} | \emptyset [P]$ we understand $\mathbf{Y} \perp\!\!\!\perp \mathbf{W} | \mathbf{Z} [P]$ as $\mathbf{Y} \perp\!\!\!\perp \mathbf{W} [P]$ when $\mathbf{Z} = \emptyset$.

If $\mathbf{Y} \perp\!\!\!\perp \mathbf{W} [P]$ (respectably $\mathbf{Y} \perp\!\!\!\perp \mathbf{W} | \mathbf{Z} [P]$) is not true, we write $\mathbf{Y} \not\perp\!\!\!\perp \mathbf{W} | P$ (respectively $\mathbf{Y} \not\perp\!\!\!\perp \mathbf{W} | \mathbf{Z} [P]$).

Definition 2.1 (Dependency Model)
A statement of conditional independence is an expression of the form $\mathbf{Y} \perp\!\!\!\perp \mathbf{W} | \mathscr{A}$. Let $\langle \mathbf{\Omega}, \mathcal{R} \rangle$ be measurable space, and let $\mathbf{X}$ be a set of random variables defined on $\mathbf{\Omega}$. A dependency model over $\mathbf{X}$ is a rule that assigns a truth value to all statements of conditional independence of the form:

$$\mathbf{Y} \perp\!\!\!\perp \mathbf{W} | \mathscr{A},$$

where $\mathbf{Y}$ and $\mathbf{W}$ are disjoint non-empty subsets of $\mathbf{X}$ and $\mathscr{A}$ is any partitioning of $\mathbf{\Omega}$ from a certain class $\mathcal{A}$ of partitionings.

Example 2.1
Consider a probability space $\langle \mathbf{\Omega}, \mathcal{R}, P \rangle$. Probability measure P encodes a dependency model over any set of variables $\mathbf{X}$ defined on $\mathbf{\Omega}$ as any statement $\mathbf{Y} \perp\!\!\!\perp \mathbf{W} | \mathscr{A} [P]$ can be verified by inspecting relation (2.13) under P.

Example 2.2
Consider a measurable space $\langle \mathbf{\Omega}, \mathcal{R} \rangle$. One class $\mathcal{A}$ of partitionings all those partitionings that partition $\mathbf{\Omega}$ into measurable partitions $\mathscr{A} = \{A_1, \ldots, A_l\}$, that is $A_l \in \mathcal{R}$ for any $1 \leq l \leq l$. This is the least restrictive class of partitionings. Another class of partitionings arises from a set variables $\mathbf{X}$ defined on $\mathbf{\Omega}$. A class $\mathcal{A}_{\mathbf{X}}$ of partitionings of $\mathbf{\Omega}$ is generated from all possible partitionings $\mathscr{R} = \{R_0, \ldots, R_k\}$ of $R(\mathbf{X})$. Here, partitioning $\mathscr{R}$ immediately defines the partitioning $\mathscr{A} = \{A_0, \ldots, A_k\}$ of $\mathbf{\Omega}$: $A_i = \{\omega \in \mathbf{\Omega} : \mathbf{X}(\omega) = \mathbf{x} \text{ and } \mathbf{x} \in R_i\}$. A very common class of partitionings is the subclass of $\mathcal{A}_{\mathbf{X}}$ that is generated by any proper subset $\mathbf{Z} \subset \mathbf{X}$.

Given two partitionings $\mathscr{B}$ and $\mathscr{C}$, we will define the partition $\mathscr{I}(\mathscr{B}, \mathscr{C})$ as the partition consisting of the elements $\{B \cap C : B \in \mathscr{B}, C \in \mathscr{C}\}$.

Let $\mathbf{Y}$ be a set of discrete random variables w.r.t. probability space $\langle \mathbf{\Omega}, \mathcal{R}, P \rangle$. The partitioning $\mathscr{A}(\mathbf{Y})$ is then defined as:

$$\mathscr{A}(\mathbf{Y}) = \begin{cases} \{\{\omega \in \mathbf{\Omega} : \mathbf{Y}(\omega) = \mathbf{y}\} (\mathbf{y} \in R(\mathbf{Y}))\} & \text{if } \mathbf{Y} \neq \emptyset, \\ \{\mathbf{\Omega}\} & \text{otherwise.} \end{cases} \quad (2.15)$$

Axiomatic Characterisations of Conditional Independence

Extensive work has been done to characterise dependency models of a joint probability distribution. In the following we will review a set of axioms provided by Pearl (1988).[2] Let $\mathbf{X}$ be a set of random variables w.r.t. probability space $\langle \mathbf{\Omega}, \mathcal{R}, P \rangle$, and let $\mathbf{U}$, $\mathbf{Y}$ and $\mathbf{W}$ be arbitrary disjoint subsets of variables $\mathbf{X}$. Also, let $\mathscr{S}$ be some partition of the sample space $\mathbf{\Omega}$.[3] The following axioms 2.4, 2.7, then provide a sound characterisation of the dependency model encoded by P.[4]

Axiom 2.4 (Symmetry)

$$\mathbf{W} \perp\!\!\!\perp \mathbf{Y} | \mathscr{S} \Leftrightarrow \mathbf{Y} \perp\!\!\!\perp \mathbf{W} | \mathscr{S}. \tag{2.16}$$

Axiom 2.5 (Decomposition)

$$\mathbf{W} \perp\!\!\!\perp \{\mathbf{Y} \cup \mathbf{U}\} | \mathscr{S} \Rightarrow \mathbf{W} \perp\!\!\!\perp \mathbf{Y} | \mathscr{S} \wedge \mathbf{W} \perp\!\!\!\perp \mathbf{U} | \mathscr{S}. \tag{2.17}$$

Axiom 2.6 (Weak Union)

$$\mathbf{W} \perp\!\!\!\perp \{\mathbf{Y} \cup \mathbf{U}\} | \mathscr{S} \Rightarrow \mathbf{W} \perp\!\!\!\perp \mathbf{Y} | \mathscr{I}(\mathscr{S}, \mathscr{A}(\mathbf{U})). \tag{2.18}$$

Axiom 2.7 (Contraction)

$$\mathbf{W} \perp\!\!\!\perp \mathbf{Y} | \mathscr{I}(\mathscr{S}, \mathscr{A}(\mathbf{U})) \wedge \mathbf{W} \perp\!\!\!\perp \mathbf{U} | \mathscr{S} \Rightarrow \mathbf{W} \perp\!\!\!\perp \{\mathbf{Y} \cup \mathbf{U}\} | \mathscr{S}. \tag{2.19}$$

From contraction, weak union and decomposition follows the so called *block independence lemma*:

$$\mathbf{W} \perp\!\!\!\perp \mathbf{Y} | \mathscr{I}(\mathscr{S}, \mathscr{A}(\mathbf{U})) \wedge \mathbf{W} \perp\!\!\!\perp \mathbf{U} | \mathscr{S} \Leftrightarrow \mathbf{W} \perp\!\!\!\perp \{\mathbf{Y} \cup \mathbf{U}\} | \mathscr{S}. \tag{2.20}$$

Furthermore, if P is positive, then we also have the *Intersection* axiom:

Axiom 2.8 (Intersection)

$$\mathbf{W} \perp\!\!\!\perp \mathbf{Y} | \mathscr{I}(\mathscr{S}, \mathscr{A}(\mathbf{U})) \wedge \mathbf{W} \perp\!\!\!\perp \mathbf{U} | \mathscr{I}(\mathscr{S}, \mathscr{A}(\mathbf{Y})) \Rightarrow \mathbf{W} \perp\!\!\!\perp \{\mathbf{U} \cup \mathbf{Y}\} | \mathscr{S}. \tag{2.21}$$

A three-way relation that satisfies Axioms 2.4 to 2.7 are called the *semi-graphoid*, and if axiom 2.8 is also satisfied, the relation is called *graphoid*. Conditional independence is a semi-graphoid relation. A set of inference rules is complete iff all true statements can be inferred using the set of inference rules. It was shown by Studený (1989) that the *semi-graphoids* does not provide a complete characterisation of conditional independence. Still, the set of axioms

[2] It should be mentioned that the axiomatisation of Pearl (1988) was preceded by an alternative but equivalent axiomatisation of conditional independence proposed by Dawid (1979).

[3] Originally, the axioms proposed by Pearl (1988) only concerned conditional independence relations, where the conditioning partition was generated by a subset of variables. The axioms, however, are still true when the conditioning partition is allowed to be any general partition of $\mathbf{\Omega}$.

[4] A set of inference rules forms a sound characterisation if no sequence of applications of the rules can infer a false statement from a set of true statements, but rather only true statements can be inferred from true statements.

provides a sound characterisation, and can therefore still be used to infer more conditional independence relations from a set of true relations.

2.1.4 Sampled Data and likelihood

Given a joint distribution $P(\mathbf{X})$ over random variable $\mathbf{X}$, an *independent* and *identically distributed* (iid) sample of $\mathbf{X}$ of length l is a set of l random variables $\mathbf{X}_1, \ldots, \mathbf{X}_l$, each with state-space $R(\mathbf{X}_i) = R(\mathbf{X})$ and distribution $P(\mathbf{X}_i) = P(\mathbf{X})$. A database of *cases* or *instances* of $\mathbf{X}$ is a set $\mathcal{D} = \{d_1, \ldots, d_n\}$ where each element d_i is a realisation of variable $\mathbf{X}_i$ in an iid sample of $\mathbf{X}$ of length n. We will not emphasise the distinction between an iid sample and a database of realisation of an iid sample, and will for simplicity say that $\mathcal{D}$ is an iid sample of $\mathbf{X}$ of length n when in fact $\mathcal{D}$ is a realisation of an iid sample of $\mathbf{X}$ of length n. When $\mathcal{D} = \{d_1, \ldots, d_n\}$ is an iid sample of $\mathbf{X}$, $\mathbf{Y} \subset \mathbf{X}$, and $d_i \in \mathcal{D}$, we will denote by $d_i[\mathbf{Y}]$ the projection of realisation d_i onto variables $\mathbf{Y}$.

Let $\mathcal{D}$ be an iid sample of random variables $\mathbf{X}$ and let $P(\mathbf{X})$ be an arbitrary distribution over variables $\mathbf{X}$. The likelihood of data $\mathcal{D}$ under P is then defined as:

$$l(\mathcal{D}|P) = \prod_{d \in \mathcal{D}} P(\mathbf{X} = d[\mathbf{X}]). \tag{2.22}$$

Taking the log of (2.22) yields the log-likelihood ($L(\mathcal{D}|P)$) that decomposes into a sum of logs of probabilities:

$$L(\mathcal{D}|P) = \sum_{d \in \mathcal{D}} \log P(\mathbf{X} = d[\mathbf{X}]). \tag{2.23}$$

2.2 Graphical Concepts

An *undirected graph* (UDG) is a pair $G = \langle \mathbf{V}, \mathbf{E} \rangle$, where $\mathbf{V}$ is a finite set of distinct nodes and $\mathbf{E}$ is a finite set of edges, defined as unordered pairs of distinct nodes, $\mathbf{E} \subseteq \{\{X, Y\} : X \in \mathbf{V}, Y \in \mathbf{V} \text{ and } X \neq Y\}$. In a graph $G = \langle \mathbf{V}, \mathbf{E} \rangle$, iff $\{X, Y\} \in \mathbf{E}$ we say that X and Y are *adjacent* in G. We denote the set of all adjacent nodes of node X in graph G by $adj_G(X) = \{Y \in \mathbf{V} : \{Y, X\} \in \mathbf{E}\}$.

For a graph $G = \langle \mathbf{V}, \mathbf{E} \rangle$, a subset $A \subseteq \mathbf{V}$ induces the *subgraph* $G_A = \langle A, \mathbf{E}_A \rangle$, where $\mathbf{E}_A = \{\{X, Y\} : \{X, Y\} \in \mathbf{E}, X \in A \text{ and } Y \in A\}$. A *path* π from node A to node B in a UDG $G = \langle \mathbf{V}, \mathbf{E} \rangle$ is a sequence of n nodes $X_1, X_2, \ldots, X_n$ where $n \geq 2$ and $\{X_i, X_{i+1}\} \in \mathbf{E}$ for all $1 \leq i \leq n$, and $X_1 = A$ and $X_n = B$.

Let $G = \langle \mathbf{V}, \mathbf{E} \rangle$ be an UDG and $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{S}$ be disjoint subsets of $\mathbf{V}$. Then $\mathbf{A}$ and $\mathbf{B}$ are separated by $\mathbf{S}$ iff all paths between nodes A and B, where $A \in \mathbf{A}$ and $B \in \mathbf{B}$, contains at least one node $S \in \mathbf{S}$.

A path π from A to B in graph $G = \langle \mathbf{V}, \mathbf{E} \rangle$, is a *cycle* iff $A = B$.

A graph is *connected* iff there exists a path between any two distinct nodes. Otherwise it is *disconnected*. A *connected component* in a graph $G = \langle \mathbf{V}, \mathbf{E} \rangle$ is a subgraph of G induced by a maximal subset $\mathbf{A} \subseteq \mathbf{V}$ where $G_{\mathbf{A}}$ is connected.

A *directed graph* or *digraph* is a pair $G = \langle \mathbf{V}, \mathbf{E} \rangle$ of nodes $\mathbf{V}$ and directed edges E, defined as ordered pairs of distinct nodes. We will denote a pair of nodes as being ordered by enclosing the pair in parenthesis (X_i, X_j), which represents an edge with orientation $X_i \rightarrow X_j$.

The *skeleton* of a graph $G = \langle \mathbf{V}, \mathbf{E} \rangle$ is the undirected graph obtained from G by dropping the orientation of all edges. We denote the skeleton of G by G^u. The skeleton of UDG G is just G itself.

Extending paths and cycles to digraphs gives rise to both *undirected* and *directed* versions. Let $G = \langle \mathbf{V}, \mathbf{E} \rangle$ be a digraph. A sequence of nodes $X_1, \ldots X_n$ in digraph G forms an *undirected path* iff it is a path in G^u, and it forms an *undirected cycle* iff it is a cycle in G^u. Furthermore, the sequence of nodes forms a *directed path* iff $X_i \rightarrow X_{i+1} \in \mathbf{E}$ for all $i \in [1..n]$, and forms a *directed cycle* iff it forms a directed path and $X_1 = X_n$.

A *directed acyclic graph* (DAG) is a digraph $G = \langle \mathbf{V}, \mathbf{E} \rangle$ that does not contain any directed cycles.

If $G = \langle \mathbf{V}, \mathbf{E} \rangle$ is a DAG and $(X, Y) \in \mathbf{E}$, we say (as for undirected graphs) that X and Y are adjacent, and in addition we say that Y is a child of X and X is a parent of Y in G. The set of all parents and children of node X in digraph G will be denoted $pa_G(X)$ respectively $ch_G(X)$. If there exists a directed path from node X to node Y in DAG G, we say that Y is a descendant of X in G, and we denote the set of all descendants of node X in graph G by $de_G(X)$. By $de^*_G(X)$ we denote $de_G(X) \cup X$. By $pa^*_G(X)$ we denote the set $\{Y \in \mathbf{V} : X \in de_G(Y)\}$. A set of nodes $\mathbf{A}$ is *ancestral* iff for any node $X \in \mathbf{A}$ the parents of X are also included in $\mathbf{A}$. By $pa^*_G(\mathbf{A})$ we denote the smallest ancestral set in G including $\mathbf{A}$, that is $pa^*_G(\mathbf{A}) = \mathbf{A} \cup \{\cup_{X \in \mathbf{A}} pa^*_G(X)\}$.

A *rooted* DAG is a DAG where a single unique node (the *root* node) has no parents.

A *tree* is a rooted DAG with no cycles, which also implies that any node X only has at-most one parent. A *forest* is a set of trees.

A *poly-tree* is a UDG that does not contain any cycles.

In directed graph $G = \langle \mathbf{E}, \mathbf{V} \rangle$ the set of non-descendants of node X is denote by $nd_G(X) = \mathbf{V} \setminus de^*_G(X)$.

A *chain graph* is a pair $G = \langle \mathbf{V}, \mathbf{E} \rangle$ of nodes $\mathbf{V}$ and edges $\mathbf{E}$, where an edge can either be directed or undirected. The graph obtained by removing all undirected edges from G must be a DAG (connected or disconnected). Both DAGs and UDGs are chain graphs.

The *moral graph* of a DAG $G = \langle \mathbf{V}, \mathbf{E} \rangle$ is constructed by connecting all non-adjacent pairs nodes A and B where A and B have a common child ($\{ch_G(A) \cap ch_G(B)\} \neq \emptyset$) and dropping all directions of edges in G. We denote the moral graph of G by G^m.

A graph $G = \langle \mathbf{V}, \mathbf{E} \rangle$ is said to be *complete* if all nodes in $\mathbf{V}$ are pairwise connected by edges in $\mathbf{E}$. A *clique* of graph $G = \langle \mathbf{V}, \mathbf{E} \rangle$, is a maximal subset of nodes $C \subseteq \mathbf{V}$, where G_C is complete. By $Cliques(G)$, we denote the set of all cliques in graph G.

Chapter 3

Probabilistic Graphical Models

In this chapter we introduce three different types of probabilistic graphical models. A probabilistic graphical model is a compact representation of a joint probability distribution over a finite domain of random variables, and it is composed of two parts:

1. a dependency model, and
2. a set of parameters.

The success of graphical models in a practical application often relies on the existence of efficient algorithms for solving different kinds of inference tasks. Together with the general syntax and semantics of three different probabilistic graphical model languages, we will also introduce algorithms for efficient and exact computation of inference.

We will introduce the Bayesian Network (BN) model in Section 3.2, the Naïve Bayes (NB) model in Section 3.4, and the Probabilistic Decision Graph (PDG) model in Section 3.3.

The BN model is probably one of the most popular graphical models, and it has become a standard method for handling uncertainty in many fields of research, especially in the field of artificial intelligence (Jensen, 2001; Castillo et al., 1997; Pearl, 1988). The BN model represents a distribution over a set of variables through a factorisation of local conditional distributions. The dependency model encoded by the BN model is defined by a DAG structure and using certain separation criteria, the dependency model can easily be enumerated from that DAG.

The NB model represents a distribution over a set of variables $\mathbf{X}$ by introducing a special unobserved or latent variable C. The dependency model encoded by the NB model renders all pairs of disjoint subsets of $\mathbf{X}$ conditionally independent given C.

The PDG model is still a fairly new language for probabilistic graphical modelling, and was first introduced by Jaeger (2004). Like the BN model, the PDG model also represents a distribution over a set of variables $\mathbf{X}$ through a factorisation of local conditional distributions for each variable. The dependency model encoded by the PDG model is different from the BN dependency model, as it dictates variables as independent given certain partitions of $R(\mathbf{X})$.

3.1 Inference Tasks

There are many different kinds of relevant probabilistic queries that we might want to infer answers for using PGMs. For a set of random variables $\mathbf{X}$ and a joint probability distribution $P(\mathbf{X})$ over $\mathbf{X}$, the most common queries include:

Belief Updating: This is the task of updating probabilities in the presence of evidence, that is observations of a subset of variables $\mathbf{E} \subset \mathbf{X}$. Given that variables $\mathbf{E} \subset \mathbf{X}$ have been observed in joint state $\mathbf{e} \in R(\mathbf{E})$, compute the posterior marginal $P(X_i|\mathbf{E} = \mathbf{e})$ for all $X_i \in \{\mathbf{X} \setminus \mathbf{E}\}$.

Most Probable Explanation (MPE): The task of finding the joint configuration of unobserved variables with maximal joint posterior probability given some evidence. That is, given $\mathbf{E} \subset \mathbf{X}$ have been observed in joint state $\mathbf{e} \in R(\mathbf{E})$, then the solution to MPE is:

$$\mathbf{y} = \underset{\mathbf{y}' \in R(\mathbf{Y})}{argmax}\, P(\mathbf{Y} = \mathbf{y}'|\mathbf{E} = \mathbf{e}), \tag{3.1}$$

where $\mathbf{Y} = \{\mathbf{X} \setminus \mathbf{E}\}$.

Maximum a Posteriori Hypothesis (MAP): This is a generalisation of the MPE (3.1), where $\mathbf{Y}$ is not necessarily all remaining variables but may be a proper subset $\mathbf{Y} \subseteq \{\mathbf{X} \setminus \mathbf{E}\}$.

We regard belief updating as the primary task for any general purpose language for probabilistic graphical modelling. We will, therefore, identify for each language the complexity associated with solving this problem in general. In particular, for a model M from language $\mathcal{L}$, we will identify the effective size of model M, denoted $size_{\mathit{eff}}(M)$. The effective size is a model specific parameter such that in M general belief updating is computable in linear time in $size_{\mathit{eff}}(M)$. This will enable easy comparison of the (theoretical) efficiency of models from different languages.

3.2 Bayesian Network Models

A BN $B = \langle G, \theta \rangle$ is a pair consisting of a DAG $G = \langle \mathbf{V}, \mathbf{E} \rangle$ and parameters θ. Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ be a set of n discrete random variables. A DAG over $\mathbf{X}$ is a DAG $G = \langle \mathbf{V}, \mathbf{E} \rangle$, where nodes are defined in a 1-to-1 correspondence with variables in $\mathbf{X}$. We will not distinguish between nodes of a DAG and associated random variables, when the meaning is clear from context. Thus, for random variable X associated with node V, we will use the notation $pa_G(X)$ to mean both the parents of V in G, and the set of random variables associated with parents of V in G.

A BN $B = \langle G, \theta \rangle$ over $\mathbf{X}$ represents $P(\mathbf{X})$ by the *directed* factorisation defined by (3.2), where θ defines local distributions for each variable X_i conditional on its parents in G, $P(X_i|pa_G(X_i))$.

Definition 3.1 (Directed Factorisation(DF))
A joint probability distribution P over variables $\mathbf{X}$ is said to factorise w.r.t. DAG G over $\mathbf{X}$ iff:

$$P(\mathbf{X}) = \prod_{X_i \in \mathbf{X}} P(X_i | pa_G(X_i)). \tag{3.2}$$

3.2.1 The Bayesian Network Dependency Model

The dependency model encoded by the BN has received enormous attention (Lauritzen et al., 1990; Castelo, 2002; Pearl, 1988; Kočka, 2001). It is usually termed the DAG Markov model, and we will review the so-called Markov properties that follows from Definition 3.1. The dependency model is important for our learning procedure for BN models and for efficient inference in a BN model. Some of the most popular algorithms for exact inference in BN models does not work on the DAG structure, but instead compiles the DAG into an equivalent undirected (UDG) model on which computations are then performed. Such algorithms are typically referred to as clique tree algorithms, junction tree algorithms, or variable clustering algorithms. We will review the basic architecture of such algorithms in Section 3.2.3. For our learning algorithms, it is important to establish an efficient characterisation of equivalence classes of BN models. The study of such characterisations builds on results of UDG models. Therefore we will briefly review important results concerning the UDG model.

Factorisation w.r.t. an undirected graph over random variables $\mathbf{X}$ is defined as a factorisation over *clique* potentials of the graph in Definition 3.2.

Definition 3.2 (Undirected Factorisation (UF))
A joint probability distribution P over variables $\mathbf{X}$ is said to satisfy undirected factorisation (UF) w.r.t. UDG $G = \langle \mathbf{V}, \mathbf{E} \rangle$, iff there exists non-negative mutually independent clique-potential functions $\psi_{\mathbf{A}}$ for which:

$$P(\mathbf{X}) = \prod_{\mathbf{A} \in Cliques(G)} \psi_{\mathbf{A}}, \tag{3.3}$$

where $\psi_{\mathbf{A}}$ is a function or potential over clique $\mathbf{A}$.

Definition 3.3 (Undirected Global Markov Property (UG))
A joint probability distribution P over random variables $\mathbf{X}$ satisfies the Undirected Global Markov Property (UG) w.r.t. UDG G iff for any triple of disjoint subsets $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{S}$ of $\mathbf{X}$, where $\mathbf{S}$ separates $\mathbf{A}$ from $\mathbf{B}$ in G, the following holds:

$$\mathbf{A} \perp\!\!\!\perp \mathbf{B} | \mathbf{S} [P]. \tag{3.4}$$

UF and UG are connected by Proposition 3.1. It was first stated and proved by Lauritzen et al. (1990):

Proposition 3.1
(Lauritzen et al., 1990, Proposition 1) If joint distribution P over random variables $\mathbf{X}$ satisfy UF w.r.t. UDG G, then P satisfies UG w.r.t. G.

Lauritzen et al. (1990) connects undirected and directed factorisations (Definitions 3.1 and 3.2) in Lemmas 3.1 and 3.2:

Lemma 3.1
(Lauritzen et al., 1990, Lemma 1) If joint probability distribution P satisfies DF w.r.t. DAG G, then P satisfies UF w.r.t. G^m (and therefore UG w.r.t. G^m).

Lemma 3.2
(Lauritzen et al., 1990, Lemma 2) If joint probability distribution P over random variables $\mathbf{X}$ satisfies DF w.r.t. DAG G, and $\mathbf{A}$ is an ancestral set in G, then the marginal distribution $P(\mathbf{A})$ satisfies DF w.r.t. $G_{\mathbf{A}}$

Definition 3.4 (Directed Global Markov Property (DG))
A discrete joint probability distribution P over random variables $\mathbf{X}$ is said to satisfy the directed global Markov property (DG) w.r.t. DAG G over $\mathbf{X}$ iff for any triple of disjoint subsets $\mathbf{A} \subseteq \mathbf{X}$, $\mathbf{B} \subseteq \mathbf{X}$ and $\mathbf{S} \subseteq \mathbf{X}$, where $\mathbf{S}$ separates $\mathbf{A}$ from $\mathbf{B}$ in $(G_{pa^*_G(\mathbf{A}\cup\mathbf{B}\cup\mathbf{S})})^m$:

$$\mathbf{A} \perp\!\!\!\perp \mathbf{B} | \mathbf{S} [P]. \tag{3.5}$$

From Lemmas 3.1 and 3.2, it follows that if P satisfies DF w.r.t. DAG G, then P satisfies the DG w.r.t. G (Lauritzen et al., 1990, Corollary 1).

Definition 3.5 (Directed Local Markov Property (DL))
A discrete joint probability distribution P over variables $\mathbf{X}$ satisfies the directed local Markov property w.r.t. DAG G iff for any variable $X \in \mathbf{X}$:

$$X \perp\!\!\!\perp nd_G(X) \backslash pa_G(X) | pa_G(X) [P]. \tag{3.6}$$

Lauritzen et al. (1990) state and prove equivalence of directed factorisation, directed global and directed local Markov properties (Definitions 3.1, 3.4 and 3.5):

Theorem 3.1
(Lauritzen et al., 1990, Theorem 1) For a discrete probability distribution P over random variables $\mathbf{X}$ and DAG G over $\mathbf{X}$, the following statements are equivalent:

1. *P satisfies DF w.r.t. G,*
2. *P satisfies DG w.r.t. G,*
3. *P satisfies DL w.r.t. G.*

When distribution P factorise w.r.t. DAG G, G is called an *I-map* of P. Let I be a statement of conditional independence, we then say that DAG G *entails* I iff I is true for all distributions P for which G is an I-map (denoted $G \models_P I$).

A popular graphical criterion for reading independence relations entailed by a DAG is the *d-separation* criterion (Pearl and Verma, 1987), defined as:

Definition 3.6 (d-separation)
Let $G = \langle \mathbf{V}, \mathbf{E} \rangle$ be a DAG with nodes $\mathbf{V}$ and directed edges $\mathbf{E}$. Two distinct nodes $X, Y \in \mathbf{V}$ are said to be d-separated in DAG G by $\mathbf{Z} \subset \mathbf{V}$ iff for every path π (undirected or directed) between X and Y there exists a node W such that either:

- *$W \in \mathbf{Z}$ and there is no head-to-head connection at W w.r.t. path π, or*
- *$W \notin \mathbf{Z}$, non of $de_G(W)$ are included in $\mathbf{Z}$ and there is a head-to-head connection at W w.r.t. path π.*

The definition extends to sets of variables by denoting $\mathbf{U} \subset \mathbf{X}$ being d-separated from subset $\mathbf{W} \subset \mathbf{X}$ by $\mathbf{Z} \subset \mathbf{X}$ in G iff any two nodes $U \in \mathbf{U}$ and $W \in \mathbf{W}$ are d-separated by $\mathbf{Z}$ in G.

We denote by $G \models_{d\text{-}sep} X \perp\!\!\!\perp Y | \mathbf{Z}$ the statement that in DAG G, X and Y are d-separated by $\mathbf{Z}$. As a rule for inferring conditional independencies entailed by DAG G, d-separation is both sound ($[G \models_{d\text{-}sep} I] \Rightarrow [G \models_P I]$) and complete ($[G \models_P I] \Rightarrow [G \models_{d\text{-}sep} I]$), first proved by Geiger and Pearl (1988).

If G is an I-map of P, and P does not contain any more independencies than those entailed by G, then G is a *perfect map* of P. If some DAG G is a perfect map of distribution P, then P is called DAG-faithful.

Lauritzen et al. (1990) prove that d-separation is equivalent to the directed global Markov property as a separation criterion.

Definition 3.7 (Bayesian Network Dependency Model)
The BN B with DAG structure G over variables $\mathbf{X}$ defines a dependency model in which the true independencies are:

$$M(G) = \{\mathbf{A} \perp\!\!\!\perp \mathbf{B} | \mathbf{S} : G \models_P \mathbf{A} \perp\!\!\!\perp \mathbf{B} | \mathbf{S}\}. \tag{3.7}$$

So any distribution P that factorise w.r.t. DAG G will contain (at least) all the independencies $M(G)$. Using the terminology of Definition 2.1, we say that the class of partitionings used in BN dependency models, is the class of all partitionings that can be induced by some subset of variables $\mathbf{S} \subset \mathbf{X}$.

Given a DAG $G = \langle \mathbf{V}, \mathbf{E} \rangle$, and disjoint subsets $\mathbf{A}, \mathbf{S} \subseteq \mathbf{V}$, Geiger et al. (1990) present an algorithm for computing the set $\mathbf{B}$ of all nodes that are d-separated from $\mathbf{A}$ given $\mathbf{S}$. We present it here as function `getDSeparated` (Algorithm 3.2), which uses the subroutine `getReachable` to determine a set of nodes that are reachable by a legal path (see Algorithm 3.1). `getReachable` has complexity $O(|\mathbf{E}| \cdot |\mathbf{V}|)$ in general, however Geiger et al. (1990) show that when the set of illegal pairs of edges is constructed as in line 4 of `getDSeparated`, `getReachable` will run in time linear in $|\mathbf{E}|$. As no operation in `getDSeparated` has worse complexity than $O(|\mathbf{E}|)$, the overall complexity of `getDSeparated` is therefore $O(|\mathbf{E}|)$.

Algorithm 3.1 This algorithm is needed by algorithm 3.2.

Input: G : DAG over $\mathbf{X}$; $\mathbf{F}$: a set of illegal pairs of edges; $\mathbf{A}$: a set of nodes $\mathbf{A} \subset \mathbf{X}$.

Output: A set of nodes $\mathbf{R} \subset \mathbf{X}$ reachable from $\mathbf{A}$ via a legal path.

1: **function** `getReachable`(G, $\mathbf{F}$, $\mathbf{A}$)
2: $\mathbf{X} := \mathbf{X} \cup X_s$
3: $\mathbf{R} := \{X_s\} \cup \mathbf{A}$
4: **for all** $X \in \mathbf{A}$ **do**
5: $\mathbf{E} := \mathbf{E} \cup X_s \to X$
6: label $X_s \to X$ with 1
7: $i := 1$
8: **repeat**
9: Let $\mathbf{U}$ be the set of unlabelled edges $X_k \to X_l$ from $\mathbf{E}$ s.t. there exists $X_j \to X_k$ labelled i and $(X_j \to X_k, X_k \to X_i) \notin \mathbf{F}$.
10: **for all** $X_k \to X_l \in \mathbf{U}$ **do**
11: $\mathbf{R} := \mathbf{R} \cup \{X_l\}$
12: label $X_k \to X_l$ with $i + 1$.
13: $i := i + 1$
14: **until** $\mathbf{U} = \emptyset$
15: **return** $\mathbf{R}$

Algorithm 3.2 This function computes and returns the set of variables $\mathbf{B}$ d-separated from a target set $\mathbf{A}$ given a separating set $\mathbf{S}$ in a DAG G.

Input: G : DAG structure over variables $\mathbf{X}$; disjoint subsets $\mathbf{A}, \mathbf{S} \subset \mathbf{X}$.

Output: The set of variables $\mathbf{B}$ *d-separated* from $\mathbf{A}$ by $\mathbf{S}$.

1: **function** `getDSeparated`(B, $\mathbf{A}$, $\mathbf{S}$)
2: Construct the graph $G' = \langle \mathbf{V}, \mathbf{E}' \rangle$ where $\mathbf{E}' := \mathbf{E} \cup \{X_i \to X_j : X_j \to X_i \in \mathbf{E}\}$.
3: Construct the table $\mathit{descendant}(X_i) := \begin{cases} \texttt{true} & \text{if } \{\{X_i\} \cup de_G(X_i)\} \cap \mathbf{S} \neq \emptyset \\ \texttt{false} & \text{otherwise} \end{cases}$
4: Construct the set $\mathbf{F}^C$ of pairs of edges $(X_j \to X_k, X_k \to X_l)$ where $X_j \neq X_l$ and either
- $X_j \to X_k, X_k \leftarrow X_l \in \mathbf{E}$ and $\mathit{descendant}(X_k) = \texttt{true}$, or
- $X_j \to X_k, X_k \leftarrow X_l \notin \mathbf{E}$ and $X_k \notin \mathbf{S}$.

5: $\mathbf{B}' := \texttt{getReachable}(G', \mathbf{E}' \setminus \mathbf{F}^C, \mathbf{A})$
6: **return** $\mathbf{V} \setminus \{\mathbf{B}' \cup \mathbf{A} \cup \mathbf{S}\}$

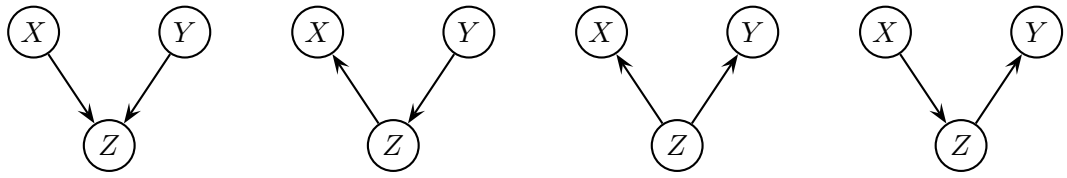


Figure 3.1. 4 different DAG structures over $\mathbf{X} = \{X, Y, Z\}$. (a) is not equivalent with any of the other, and (b), (c) and (d) are all equivalent.

3.2.2 BN Model Equivalence and Inclusion

In this section, we define a partial ordering of BN dependency models. By Definition 3.7, the BN dependency model is the set of statements of independence that are entailed by the DAG structure of the BN model. Inclusion of one dependency model in another is now defined w.r.t. the set of distributions that can be represented by the models:

Definition 3.8
Let $G_1 = \langle \mathbf{X}, \mathbf{E}_1 \rangle$ and $G_2 = \langle \mathbf{X}, \mathbf{E}_2 \rangle$ be DAGs. We say that model $M(G_2)$ distributionally includes $M(G_1)$ iff $M(G_2) \subseteq M(G_1)$. We will denote this by $M(G_1) \subseteq_D M(G_2)$.

If $M(G_1) \subseteq_D M(G_2)$ then for any parametrisation θ of BN $B_1 = \langle G_1, \theta \rangle$ there exists a parametrisation θ' of BN $B_2 = \langle G_2, \theta' \rangle$ such that $P^{B_1}(\mathbf{X}) = P^{B_2}(\mathbf{X})$.

Definition 3.9
Let G and H be DAGs over the same set of variables $\mathbf{X}$. G and H are distributionally equivalent iff $M(G) = M(H)$. We will denote distributional equivalence by $G \approx H$.

In the reminder of this thesis, we will refer to *distributional inclusion* and *distributional equivalence* by simply *inclusion* and *equivalence* unless otherwise stated.

Example 3.1
The empty DAG $G^{\emptyset}$ with no edges defines dependency model $M(G^{\emptyset}) = \{A \perp\!\!\!\perp B | \mathbf{S} : A, B \in \mathbf{X}, \mathbf{S} \subseteq \mathbf{X} \setminus \{A, B\}\}$, i.e., all pairs of disjoint sets of variables are marginally and conditionally independent. The dependency model $M(G^{\emptyset})$ is included in all other BN dependency models over $\mathbf{X}$. The complete DAG G^* where all pairs of nodes are connected by an edge, defines the dependency model $M(G^*) = \emptyset$, i.e., G^* entails no independencies. $M(G^*)$ obviously includes all other BN dependency models over $\mathbf{X}$.

Definition 3.10
For a DAG G we define the equivalence class $\mathcal{E}(G)$ as:

$$\mathcal{E}(G) = \{H : H \approx G\}.$$

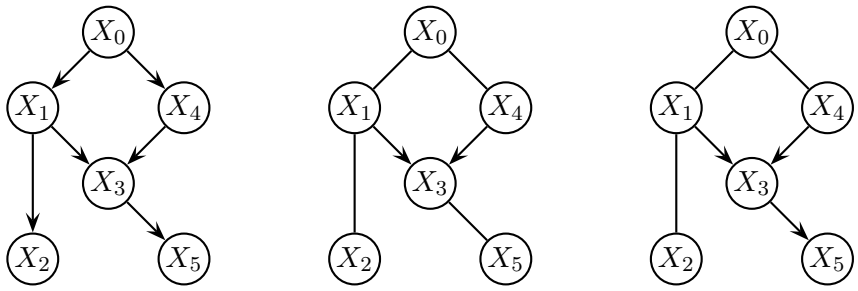


Figure 3.2. (a) shows DAG G, (b) shows the pattern common to all members of $\mathcal{E}(G)$ and (c) shows the completed pattern that identifies all compelled and reversible edges.

Example 3.2
The 4 different DAG structures G_a, G_b, G_c and G_d over $\mathbf{X} = \{X, Y, Z\}$ in Figure 3.1(a)-(d) respectively, are related in terms of equivalence as: $G_a \not\approx G_b \approx G_c \approx G_d$. G_b, G_c and G_d all entail the single statement $X \perp\!\!\!\perp Y|Z$, while G_a entails the statement $X \not\perp\!\!\!\perp Y$.

A *v-structure* in a DAG $G = \langle \mathbf{X}, \mathbf{E} \rangle$ is a triple of nodes $(X, Z, Y) \in \mathbf{X}$ where $X \to Z \in \mathbf{E}$ and $Y \to Z \in \mathbf{E}$ and $X \notin adj_G(Y)$. A classic characterisation of DAG equivalence was given by Verma and Pearl (1991):

Theorem 3.2
(Verma and Pearl, 1991, Theorem 1) Let G and H be DAGs over the same set of nodes $\mathbf{X}$. Then $G \approx H$ iff G and H have the same skeleton ($G^u = H^u$) and contains the same set of v-structures.

Theorem 3.2 says that not only is the skeleton invariant for equivalent DAGs, but also the orientation of some edges, in particular those participating in *v-structures*. An edge in DAG G that has the same orientation in all DAGs $G' \in \mathcal{E}(G)$ is said to be *compelled*. An edge that is not compelled is *reversible*.

Verma and Pearl (1991) defines the *pattern* of a DAG as the *partially directed acyclic graph (PDAG)* constructed by dropping the orientation of any edge *not* participating in a v-structure. By theorem 3.2, the pattern of a DAG G provides canonical representation of $\mathcal{E}(G)$.

Given a DAG G, we will denote the PDAG that contains directed edges for all compelled edges and undirected edges for all reversible edges in G, as the *completed PDGA (CPDAG)* for G.

Example 3.3
For a DAG G, there may be more edges than the ones participating in a v-structure that are compelled and, hence, the pattern and the completed PDAG does not always coincide. Consider for example the DAG G in Fig. 3.2(a) for which 3 edges are compelled ($X_1 \to X_3$,

Algorithm 3.3 Convert a DAG structure to its pattern.

Input: DAG $G = \langle \mathbf{V}, \mathbf{E} \rangle$
Output: Pattern of DAG G

```
1: function DAGToPattern(G)
2:     G' := copy(G^u)
3:     L := ∅
4:     for all X_i → X_k ∈ E do
5:         if X_i → X_k ∉ L then
6:             if pa_G(X_k) \ {adj_G(X_i) ∪ X_i} ≠ ∅ then
7:                 for all X_j ∈ {pa_G(X_k) \ {adj_G(X_i)} do
8:                     direct X_j → X_k in G'
9:                     L := L ∪ (X_j → X_k)
10:    return G'
```

$X_4 \rightarrow X_3$ and $X_3 \rightarrow X_5$). The pattern of G is shown in Fig. 3.2(b) and the CPDAG of G in Fig. 3.2(c).

A simple algorithm for constructing the pattern from a DAG is given in Algorithm 3.3. It visits every edge only once, and for each edge a set subtraction is performed, which can be done in linear time in the size of the largest set. The size of the largest set is bounded by $|\mathbf{E}|$, and the complexity of the algorithm will then be bounded by $O(k \cdot |\mathbf{E}|^2)$. When DAGs are sparsely connected (as is typically the case for BN models) the sets $pa_G(X_i)$ and $adj_G(X_i)$ are small compared to $\mathbf{E}$, yielding in practise sub-polynomial complexity.

A characterisation of equivalent DAGs based on a local transformation was developed by Chickering (1995) using the concept of *covered* edges in DAGs. An edge $X_i \rightarrow X_j$ in DAG G is covered iff $pa_G(X_i) = pa_G(X_j) \setminus X_i$.

Lemma 3.3
(Chickering, 1995, Lemma 1) Let G be a DAG over variables $\mathbf{X}$ containing the edge $X_i \rightarrow X_j$. Let H be a DAG identical to G with the single exception that H contains $X_i \leftarrow X_j$ instead of $X_i \rightarrow X_j$. Then $G \approx H$ iff $X_i \rightarrow X_j$ is covered in G.

Chickering (1995) uses Lemma 3.3 to develop the following characterisation of $\mathcal{E}(G)$:

Theorem 3.3
(Chickering, 1995, Theorem 2) Let G and H be DAGs over the same set of variables $\mathbf{X}$, let $G \approx H$ and let n be the number of edges that do not have the same orientation in H and G. Then there exists a sequence of n distinct edge reversals in G where:

1. *each edge when reversed is covered,*
2. *after each reversal G is a DAG and $G \approx H$, and*
3. *after all reversals $G = H$, that is G and H are identical.*

In the ordering of models defined by the inclusion relation, we can define the boundary of a model, the *inclusion boundary* (Kočka, 2001; Kočka et al., 2001) of a BN dependency model $M(G)$:

Definition 3.11 (Inclusion Boundary)
Let $B = \langle G, \theta \rangle$ be a BN model. The Inclusion Boundary of BN dependency model $M(G)$, denoted $IB(M(G))$ is defined as:

$$IB(M(G)) \quad = \quad UIB(M(G)) \cup LIB(M(G)), \tag{3.8}$$

where:

$$UIB(M(G)) = \{M(U) : M(G) \subset_D M(U),\ \nexists U'[M(G) \subset_D M(U') \subset_D M(U)]\}, \tag{3.9}$$
$$LIB(M(G)) = \{M(L) : M(L) \subset_D M(G),\ \nexists L'[M(L) \subset_D M(L') \subset_D M(G)]\}. \tag{3.10}$$

$LIB(M(G))$ consists of BN dependency models that contains more statements of conditional independence than $M(G)$, and $UIB(M(G))$ consists of BN dependency models that contains less statements of conditional independence than $M(G)$. Both boundaries consists of the set of BN models "closest" to $M(G)$. A transformational characterisation of the inclusion boundary was provided by Castelo and Kočka (2003):

Theorem 3.4
(Castelo and Kočka, 2003, Theorem 3.2) Let G be a DAG, and let G^{+e} and G^{-e} be the set of DAGs that can be constructed from G by a single edge addition or removal, respectively. The inclusion boundary of the BN dependency model defined by DAG structure G is:

$$IB(M(G)) = \{M(Q') : Q' \in \{Q^{-e} \cup Q^{+e}\} \text{ and } Q \approx G\}. \tag{3.11}$$

It is certainly the case that $IB(M(G)) \supseteq \{M(G') : G' \in \{G^{-e} \cup G^{+e}\}\}$. However, not all models in $IB(M(G))$ can be generated by adding or removing an edge from DAG G, as the following example (Example 3.4) shows.

Example 3.4
Consider a domain $\mathbf{X} = \{X, Y, Z\}$. Let G be the DAG shown in Figure 3.3(a). The inclusion boundary $IB(M(G))$ is defined by the DAGs with patterns shown in Figure 3.3(b)-(f). Notice that from DAG G we can not construct a DAG with the pattern shown in Figure 3.3(e) by edge addition or removal. However, by reversing the covered edge $X \rightarrow Y$, creating DAG $Q \approx G$ and adding $Z \rightarrow Y$ to Q we get the single DAG of Figure 3.3(e).

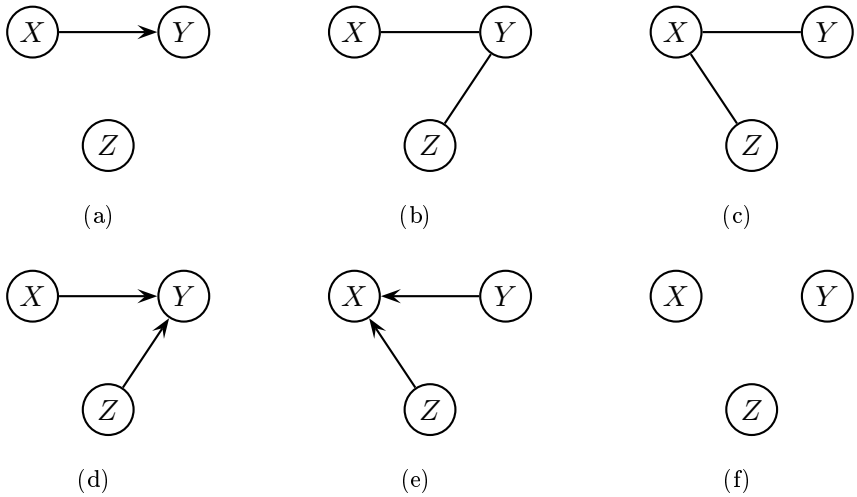


Figure 3.3. A DAG and its inclusion boundary. Figure (a) shows DAG G over $\mathbf{X} = \{X, Y, Z\}$, Figure (b), (c), (d) and (e) shows the patterns representing the 4 equivalence classes in $UIB(M(G))$. Figure (f) shows the single model in $LIB(M(G))$, the empty DAG.

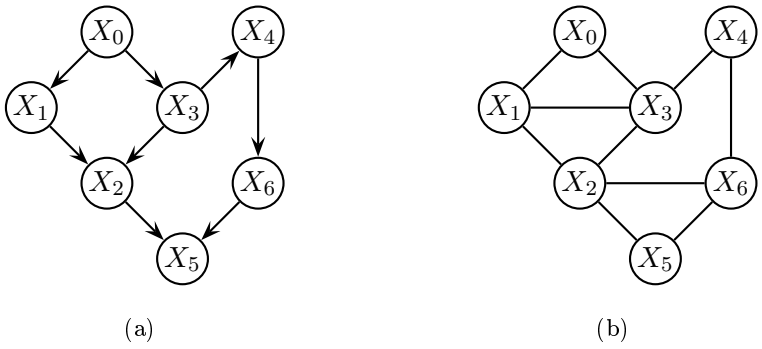


Figure 3.4. A DAG G (a) and its underlying moral graph G^m (b).

3.2.3 Inference

The general problem of belief updating in BNs is NP-hard (Cooper, 1987) and this is true even for algorithms that only compute approximate solutions (Dagum and Luby, 1993). In this section, we will give an overview of the nature of popular approaches to the problem of exact belief updating and general inference in BNs.

Consider the BN $B = \langle G, V \rangle$ over variables $\mathbf{X} = \{X_0, X_1, \ldots, X_6\}$ with the structure G shown in Fig. 3.4(a). We have the following factorisation:

$$\begin{aligned} P(\mathbf{X}) &= P(X_0)P(X_1|X_0)P(X_2|X_1, X_3)P(X_3|X_0) \\ &\quad P(X_4|X_3)P(X_5|X_2, X_6)P(X_6|X_4). \end{aligned} \tag{3.12}$$

We will first focus on calculating $P_B(\mathbf{Y} = \mathbf{y})$ for some $\mathbf{Y} \subset \mathbf{X}$ and $\mathbf{y} \in R(\mathbf{Y})$. Let $\mathbf{Z} = \{\mathbf{x} : \mathbf{x} \in R(\mathbf{X}) \text{ and } \mathbf{x}[\mathbf{Y}] = \mathbf{y}\}$. Then :

$$P^B(\mathbf{Y} = \mathbf{y}) = \sum_{\mathbf{x} \in \mathbf{Z}} P^B(\mathbf{X} = \mathbf{x}) \tag{3.13}$$

However, it is not tractable to compute the full joint distribution $P^B(\mathbf{X})$ as that would require storrage-space of exponential size in the number of variables. By systematic query specific manipulations of (3.12) we can often reduce the complexity of (3.13).

Query specific simplification

This approach aims at simplifying the factorisation (3.12), before an answer to a given query is computed through repeated multiplications and summations. The simplifications are captured graphically by the removal of variables that are irrelevant w.r.t. the specific query. Shachter (1988) introduces the concept of *barren* variables:

Definition 3.12
Let G be a DAG over variables $\mathbf{X}$, $\mathbf{Y} \subseteq \mathbf{X}$ and $\mathbf{y} \in R(\mathbf{Y})$. A variable $X \in \mathbf{X}$ in a BN $B = \langle G, \theta \rangle$ is barren w.r.t. a query $P(\mathbf{Y} = \mathbf{y})$ if X is a leaf and $X \notin \mathbf{Y}$.

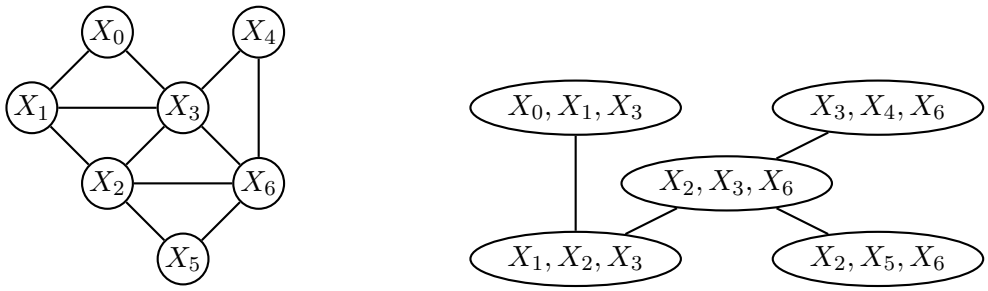
Let B be a BN model over random variables $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{y}$ be like in Definition 3.12, and let B' be the BN obtained from B by removing all barren variables X and the associated potentials $P^B(X|pa_G(X))$. Shachter (1988) then shows that:

$$P^B(\mathbf{Y} = \mathbf{y}) = P^{B'}(\mathbf{Y} = \mathbf{y}).$$

Removal of barren variables is equivalent to removing potentials in the factorisation that will sum to 1. When removing a barren variable, more variables may become barren. In fact, by repeatedly removing barren variables, we end up with a BN over $\mathbf{X}' = \{X : X \in pa^*_G(\mathbf{Y})\}$ with structure $G_{pa^*_G(\mathbf{Y})}$. After removing from B all variables $X \notin pa^*_G(\mathbf{Y})$, we can further remove variables that are d-separated from $\mathbf{Y}$. These variables can be identified using Algorithm 3.2 in linear time in the number of edges in the structure. By removing all variables that are irrelevant w.r.t. our query in BN B we get a reduced BN B', and we can continue calculating $P(\mathbf{Y} = \mathbf{y})$ using the simpler structure of B' instead of the original structure B.

The *variable elimination* algorithm by Zhang and Poole (1994) starts by pruning variables that are irrelevant to the specific query. After variable pruning, the remaining variables that are *not* irrelevant but not included in the final result (i.e., not in $\mathbf{Y}$), are *eliminated* through summation as in eq. (3.13). This summation may be done in more stages, in each stage only performing the required multiplications. Assume that we wish to compute $P(X_5 = x_{5,h})$ in the model with structure G shown in Figure 3.4(a). We could for instance partition the sum in (3.13) into two sums, one over joint configurations of the variables $\mathbf{X} \setminus X_1$ and one over all $x_{1,h} \in R(X_1)$, and get the equivalent sum:

$$P^B(X_5 = x_{5,h}) \sum_{\substack{\mathbf{x}' \in \\ R(\mathbf{X} \setminus X_1)}} \sum_{\substack{x_{1,h} \in \\ R(X_1)}} P(\mathbf{X} = (\mathbf{x}', x_{1,h})), \tag{3.14}$$



(a) (b)

Figure 3.5. A triangulated version of the DAG shown in Fig. 3.4(a), and a join tree over the cliques constructed from this triangulation (b).

For this operation we need to create the potential $\phi = P(X_1|X_0)P(X_2|X_1, X_3)$, then sum over values $R(X_1)$ of entries in ϕ creating the new potential ϕ' over X_0, X_2 and X_3 which we then work with from here on. Different sequences of such summations lead to different sized potentials that we need to handle in the computation. Some sequences might lead to intractably large potentials, and a good *elimination sequence* has to be established. An optimal elimination sequence results in working only with potentials of minimal size.

The moral graph of the DAG structure reveals the cost of an elimination sequence in terms of the size of the potentials one will need to perform operations on. In the moral graph, any two variables that are contained in the same factor are adjacent. Fig. 3.4(b) shows the moral graph of the DAG in Fig. 3.4(a). When eliminating a variable X, one creates a potential over all neighbours of X in the moral graph. If the moral graph is triangulated,[1] it is possible to find an elimination sequence that does not introduce potentials larger than the original factors. Such a sequence can be constructed by repeatedly removing variables from the graph, always choosing as the next variable to be removed, a variable that is only a member of one single clique. The moral graph in Fig. 3.4(b) is not triangulated, but we can triangulate it by adding an extra *fill-in* edge, either $X_2 - X_4$ or $X_3 - X_6$. In Fig. 3.5(a) the fill-in $X_3 - X_6$ has been added to triangulate the moral graph in Fig. 3.4(b). The cliques of the triangulated moral graph determines the size of the potentials that we need to work with in a summation. In our example we see that cliques over at most 3 variables are necessary. Depending on the range of the variables in the domain, the size of the potentials over the cliques can be different for different triangulations. Finding a minimum triangulation is NP-complete (Arnborg et al., 1987), but efficient heuristics are known, see (Kjærulff, 1990) for an empirical comparison of some common heuristic approaches to minimal triangulation.

[1] A graph is triangulated iff there are no cordless cycles. A cordless cycle is a cycle π of length 4 or more where no proper subset of nodes from π forms a cycle.

Clique Tree Propagation

A somewhat different approach is taken in *clique tree* based algorithms. These algorithms work on a secondary *clique tree* structure build from the triangulated moral graph. A *clique tree* for a graph is any tree structure over the cliques satisfying the running intersection property. The running intersection property is satisfied if and only if for any two cliques C_i and C_j in the clique tree, all cliques on the path between C_i and C_j contain the variables $C_i \cap C_j$.

Figure 3.5(b) shows a clique tree over the cliques in the triangulated moral graph of Figure 3.5(a). By attaching each potential $P(X_i|pa_G(X_i))$ from the original BN model to a single clique C_j containing $X_i \cup pa_G(X_i)$, we construct *clique*-potentials:

$$\Phi_j = \begin{cases} \prod_{\mathbf{A}_j} P(X_i|pa_G(X_i)) & \mathbf{A}_j \neq \emptyset \\ 1 & otherwise \end{cases},$$

where $\mathbf{A}_j$ is the set of potentials attached to clique C_j. The undirected factorisation of the potentials w.r.t. the clique tree over cliques $\mathbf{C}$ is then:

$$P(\mathbf{X}) = \prod_{C_j \in \mathbf{C}} \Phi_j. \tag{3.15}$$

Algorithms for inference in a clique tree structure have been studied extensively (Jensen et al., 1990a,b; Lauritzen and Spiegelhalter, 1988; Shafer and Shenoy, 1990), and they are all variations over the common idea of absorbing evidence and passing messages. For answering a query on the posterior distribution $P(X_i|\mathbf{E} = \mathbf{e})$, evidence $\mathbf{e}$ is absorbed as follows: for each variable $E \in \mathbf{E}$ find a clique C_i containing E and update the potential Φ_i as:

$$\Phi_i = \Phi_i \cdot \mathbf{1}_{\mathbf{e}[E]}(E), \tag{3.16}$$

where $\mathbf{1}_{\mathbf{e}[E]}(E)$ is the indicator function:

$$\mathbf{1}_{\mathbf{e}[E]}(E) = \begin{cases} 1 & \text{if } E = \mathbf{e}[E], \\ 0 & \text{otherwise.} \end{cases}$$

In the message passing phase, messages are send between adjacent cliques. The message $\phi_{i\to j}$ send from clique C_i to adjacent clique C_j is constructed as:

$$\phi_{i\to j} = \sum_{C_i \setminus C_j} \Phi_i. \tag{3.17}$$

A message can be sent from C_i to C_j when C_i has received a message from all other neighbours, which means that initially only leafs can send messages. When a message $\phi_{i\to j}$ is received in clique C_j, the potential Φ_j is updated as:

$$\Phi_j = \Phi_j \cdot \frac{\phi_{i\to j}}{\phi_{j\to i}}, \tag{3.18}$$

where $\phi_{j\to i} = 1$ if no message has yet been sent from clique C_j to clique C_i.

When one message has been sent in both directions along every link in the clique tree, the posterior $P(X_i, \mathbf{E} = \mathbf{e})$ can be constructed from any clique potential Φ_j containing X_i by:

$$P(X_i, \mathbf{E} = \mathbf{e}) = \sum_{X_j \in C_j \setminus \{X_i\}} \Phi_j. \tag{3.19}$$

From (3.19) the posterior $P(X_i|\mathbf{E} = \mathbf{e})$ can easily be constructed by multiplication with $P(\mathbf{E} = \mathbf{e})^{-1} = (\sum_{x_i \in R(X_i)} P(X_i = x_i, \mathbf{E} = \mathbf{e}))^{-1}$.

For the general query containing multiple query variables $\mathbf{Q}$, it is clear that $P(\mathbf{Q} = \mathbf{q}|\mathbf{E} = \mathbf{e})$ can be computed by first absorbing both $\mathbf{Q} = \mathbf{q}$ and $\mathbf{E} = \mathbf{e}$ as evidence to compute the joint probability $P(\mathbf{Q} = \mathbf{q}, \mathbf{E} = \mathbf{e})$ and thereafter computing $P(\mathbf{E} = \mathbf{e})$, and finally producing the posterior $P(\mathbf{Q} = \mathbf{q}|\mathbf{E} = \mathbf{e})$. If all of the variables $\mathbf{Q}$ are members of the same clique C', the computation can be done simply by absorbing $\mathbf{E} = \mathbf{e}$ and performing one full propagation. The *variable propagation* approach described in (Jensen, 2001, Section 6.2) is a general approach to constructing the posterior distributions $P(\mathbf{Q}|\mathbf{E} = \mathbf{e})$ of arbitrary sets $\mathbf{Q}$.

Complexity

Clique tree propagation approaches require absorption of evidence as defined in Eq. (3.16), computation of messages as defined in Eq. (3.17), propagation of messages and updating of potentials as defined in Eq. (3.18) and finally marginalisation as defined in Eq. (3.19). The time complexity of these computations is linear in the total number of parameters in the clique tree, that is, the number of entries in clique potentials. The number is bounded only by the size of the joint state-space of all variables $|R(\mathbf{X})|$ as we may (in the worst case scenario) have a single clique containing all variables, so the overall complexity ends up being exponential in the number of variables.

The query specific simplification of the factorisation employed in direct approaches like the variable elimination algorithm does not mitigate this problem, as we still need to construct a good elimination sequence, which is equivalent to finding a triangulation of the moralised graph yielding minimal cliques. Thus the complexity is the same as clique tree propagation.

Zhang (1998) compares clique tree propagation and variable elimination approach in terms of execution times. He finds that variable elimination is advantageous when the subset of the queried variables is relatively small. The difference in performance decreases as more variables are added to the query and, for larger queries, clique tree propagation is shown to outperform variable elimination.

Madsen and Jensen (1998) studies combinations of the two approaches, and propose a lazy evaluation scheme in the general clique tree architecture. In short, query specific pruning of barren variables and simplifications from *d-separation* can be employed to minimise the necessary computations of messages. See also (Madsen, 1999).

3.2.4 Representation and Effective Size

As previously stated, we regard the problem of belief propagation as the primary task for PGMs. Then, given a clique tree for the BN model, belief updating is solved by absorbing

evidence and performing one full propagation. The complexity of this operation is linear in the number of parameters in the clique tree. We define effective size of a BN model B (denoted $size_{\mathit{eff}}(B)$) as the size of the minimal clique tree constructed from B:

$$size_{\mathit{eff}}(M) = \sum_{C \in \mathbf{C}} |R(var(C))|, \tag{3.20}$$

where $\mathbf{C}$ is the set of cliques in the clique tree and $var(C)$ is the set of variables that are members in clique C. In general there will not be only a single unique clique tree for M, and, as mentioned above, constructing the minimal clique tree is an NP complete problem. In our experiments we will rely on clique trees constructed through heuristics. In particular, we use the default triangulation method implemented in the Hugin system (Jensen, 2006), which combines good (local) triangulations of prime components of the moral graph to get a good global triangulation. As we shall see later, the triangulation provided by the Hugin system usually is very satisfactory.

The representational size of a BN model M is the number of free parameters defined by the model, and is trivially computed from its DAG structure G over variables $\mathbf{X}$:

$$size_{rep}(M) = \sum_{X \in \mathbf{X}} (|R(X)| - 1) \cdot |R(pa_G(X))|. \tag{3.21}$$

3.3 Probabilistic Decision Graphs

The Probabilistic Decision Graph (PDG) model was first introduced by Bozga and Maler (1999), and was originally proposed as an efficient representation of probabilistic transition systems. In this study, we consider the more generalised version of PDGs introduced by Jaeger (2004).

A PDG structure is defined w.r.t. an underlying variable forest:

Definition 3.13 (Variable Forest)
Let F be a forest of rooted and directed trees $F = \{T_0, \ldots, T_k\}$ and let $\mathbf{X} = \{X_0, \ldots, X_n\}$ be a domain of n random variables. F is a variable forest over $\mathbf{X}$ when nodes from F and variables from $\mathbf{X}$ are associated in a one-to-one relation.

Definition 3.14 (PDG Structure)
Let F be a variable forest over domain $\mathbf{X}$. A PDG-structure *$G = \langle \mathbf{V}, \mathbf{E} \rangle$ for $\mathbf{X}$ w.r.t. F is a set of* rooted *DAGs (RDAGs), such that:*

1. *Each node $\nu \in \mathbf{V}$ is labelled with some $X_i \in \mathbf{X}$. By V_i, we will refer to the set of all nodes in a PDG-structure label-led with the same variable X_i.*

2. *For each node ν_i label-led with X_i, each possible state $x_{i,h}$ of X_i and each successor $X_j \in ch_F(X_i)$ there exists exactly* one *edge label-led with $x_{i,h}$ from ν_i to some node ν_j label-ed with random variable X_j. Let $X_j \in ch_F(X_i)$ and $\nu_i \in V_i$. By $succ(\nu_i, X_j, x_{i,h})$ we will then refer to the unique node $\nu_j \in V_j$ that is reached from ν_i by an edge label-led $x_{i,h}$.*

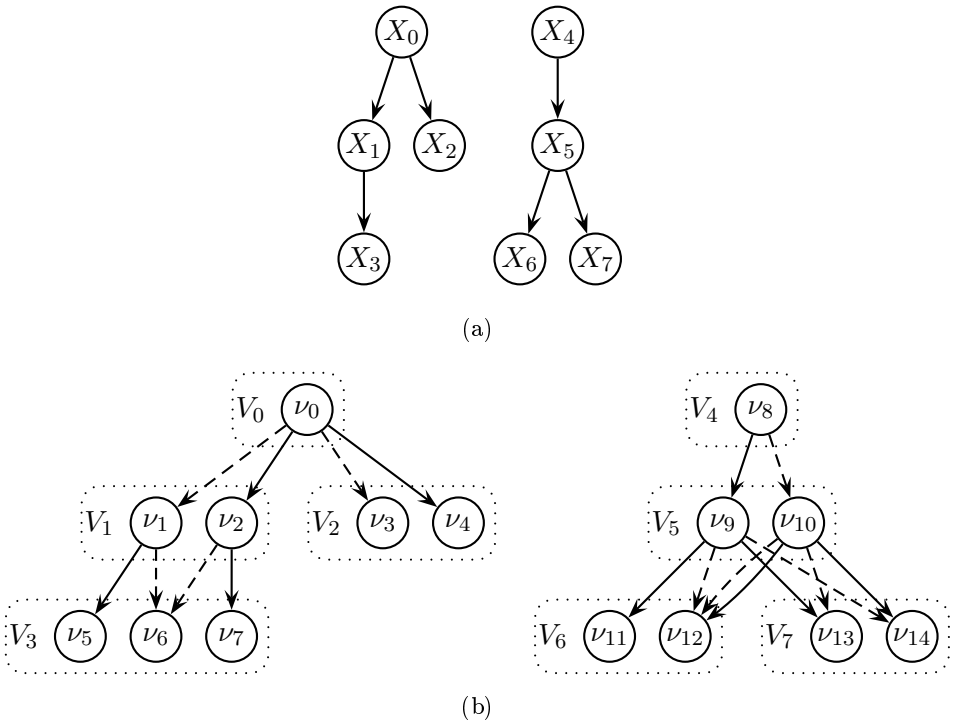


Figure 3.6. A variable forest F over binary variables $\mathbf{X} = \{X_0, \ldots, X_7\}$ is shown in (a), and a PDG-structure over $\mathbf{X}$ w.r.t. variable forest F is shown in (b).

Example 3.5
A variable forest F over binary variables $\mathbf{X} = \{X_0, \ldots, X_7\}$ can be seen in Figure 3.6(a), and a PDG structure over $\mathbf{X}$ w.r.t. F in Figure 3.6(b). The labelling of nodes ν in the PDG-structure is indicated by the dashed boxes, e.g., the nodes label-led with X_2 are visualised as the set $V_2 = \{\nu_3, \nu_4\}$. Dashed edges corresponds to edges labelled 0 and solid edges corresponds to edges labelled 1, for instance $succ(\nu_9, X_6, 0) = \nu_{12}$.

A PDG model is a special instance of a general Real Function Graph (RFG) model:

Definition 3.15 (Real Function Graph)
A Real Function Graph *(RFG) model $D = \langle G, \theta \rangle$ over discrete random variables $\mathbf{X}$ consists of a PDG-structure $G = \langle \mathbf{V}, \mathbf{E} \rangle$ w.r.t. variable forest F and independent parameters θ. θ defines for each node ν labelled with X_i a local real function over $R(X_i)$:*

$$\mathbf{p}^{\nu} : R(X_i) \rightarrow \mathbb{R}. \tag{3.22}$$

.

Definition 3.16 (Probabilistic Decision Graph)
Let $D = \langle G, \theta \rangle$ be an RFG model over $\mathbf{X}$. If for all $X_i \in \mathbf{X}$ and $\nu \in V_i$, $\mathbf{p}^{\nu}$ defines a probability distribution for random variable X_i we call D a Probabilistic Decision Graph *(PDG) model.*

For notational convenience, we will refer to the local distribution at node ν in a PDG/RFG in the form of a *parameter vector* $\mathbf{p}^\nu = (p_1^\nu, \dots p_{k_i}^\nu) \in \mathbb{R}^{k_i}$, where $k_i = |R(X_i)|$ is the number of distinct states of X_i. We will by $p_{x_{i,h}}^\nu$ refer to the h'th element of $\mathbf{p}^\nu$ under some ordering of $R(X_i)$.

The remainder of this section will be focused on reviewing important aspects of the semantics of the PDG model w.r.t. its dependency model and efficient methods for performing exact inference, previously developed by Jaeger (2004). To make the interpretation of the PDG model more smooth, we give the following Example 3.6. This is meant to help the reader build a more intuitive understanding of the PDG model.

Example 3.6
A patient arrives at the doctor with pain in the stomach. The doctor considers three possible causes of the pain: food poisoning (p), stomach flu (f) or an ulcer (u). Under the assumption that these three causes are mutually exclusive and collectively exhaustive, we can represent the unknown cause of the stomach pain by a random variable H with possible states $\{p, f, u\}$. To perform the diagnostics of the patient, the doctor is interested in the presence (p) or absence (a) of two symptoms: diarrhoea and fever. We can represent these two symptoms by two binary random variables D and F with possible states $\{p, a\}$. The doctors beliefs are the following:

- *If the patient is suffering from food poisoning, he/she is likely to experience diarrhoea but not necessarily fever which is only likely in severe cases where diarrhoea is certainly present. In terms of conditional (in)dependence, this is expressed as $D \not\perp\!\!\!\perp F | H = p$.*
- *If, however, the patient is suffering from stomach flu, the doctor expects the patient to have a fever but not necessarily any diarrhoea. Again, if the flu is unusually severe, diarrhoea may be present, and then certainly also the patient has a fever. In terms of conditional (in)dependence this is expressed as $D \not\perp\!\!\!\perp F | H = f$*
- *Lastly, if the patient suffers from an ulcer, the doctor does not imagine any connection between the presence/absence of diarrhoea and fever. This is captured in terms of conditional (in)dependence as $D \perp\!\!\!\perp F | H = c$.*

The scenario described above can be represented in the PDG model over variables H, D and F shown in Figure 3.7(a). Outgoing edges from ν_0 have been labelled according to the states of H, and edges outgoing from ν_1, ν_2 and ν_3 are solid corresponding to state p and dashed corresponding to state a of variable F.

The parameters of the PDG shown in Figure 3.7 have the probabilistic interpretation listed in Table 3.1.

Assume that the doctor has the same belief of the likelihood of observing diarrhoea given the two following unexpected states of nature:

1. *the patient suffers from food poisoning ($H = p$) and has fever, and*
2. *the patient suffers from stomach flu ($H = f$) and has diarrhoea.*

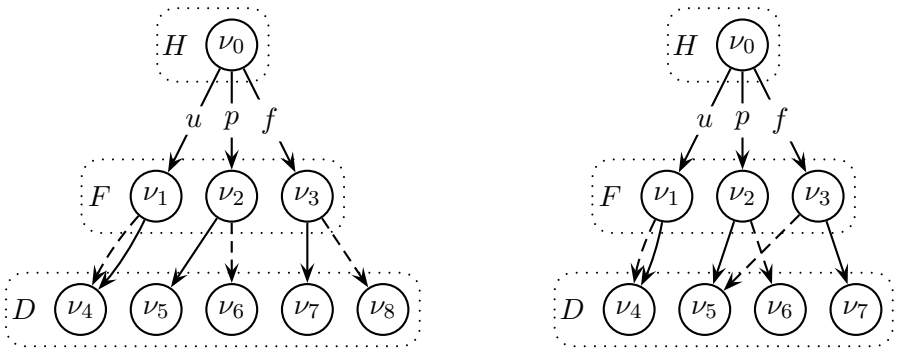


Figure 3.7. Sub-figure (a) shows the PDG structure capturing the belief of the doctor from Example 3.6, and (b) shows one example of refining the model by reusing parameters.

Parameter vector		Local distribution		Example instantiation
$\mathbf{p}^{\nu_0}$	=	$P(H)$	=	$\{.3, .3, .4\}$
$\mathbf{p}^{\nu_1}$	=	$P(F\|H = u)$	=	$\{.2, .8\}$
$\mathbf{p}^{\nu_2}$	=	$P(F\|H = p)$	=	$\{.7, .3\}$
$\mathbf{p}^{\nu_3}$	=	$P(F\|H = f)$	=	$\{.6, .4\}$
$\mathbf{p}^{\nu_4}$	=	$P(D\|H = u)$	=	$\{.7, .3\}$
$\mathbf{p}^{\nu_5}$	=	$P(D\|H = p, F = p)$	=	$\{.1, .9\}$
$\mathbf{p}^{\nu_6}$	=	$P(D\|H = p, F = a)$	=	$\{.5, .5\}$
$\mathbf{p}^{\nu_7}$	=	$P(D\|H = f, F = p)$	=	$\{.1, .9\}$
$\mathbf{p}^{\nu_8}$	=	$P(D\|H = f, F = a)$	=	$\{.3, .7\}$

Table 3.1. Probabilistic interpretation of the parameters defined by the PDG-structure in Figure 3.7(a).

This means that $\mathbf{p}^{\nu_5} = \mathbf{p}^{\nu_8}$ in Figure 3.7(a), and such reuse of parameters are easily captured in the graph structure by redirecting the edge $\nu_3 \xrightarrow{a} \nu_8$ to ν_5 and then removing ν_8, see Figure 3.7(b).

Example 3.7
A full parametrisation of the PDG structure in Fig. 3.6(b) consists of a binary probability distribution for each parameter-node ν_i, an example is shown in Table 3.2 including also the probabilistic interpretation of the parameters.

The following two definitions introduce the concepts of a node being *reached* by a joint state $\mathbf{x} \in R(\mathbf{X})$ (Definition 3.17) and the concept of a *path* (Definition 3.18).

Parameter vector		Local distribution		Example instantiation
$\mathbf{p}^{\nu_0}$	=	$P(X_0)$	=	$\{.9, .1\}$
$\mathbf{p}^{\nu_1}$	=	$P(X_1 \mid X_0 = 0)$	=	$\{.7, .3\}$
$\mathbf{p}^{\nu_2}$	=	$P(X_1 \mid X_0 = 1)$	=	$\{.1, .9\}$
$\mathbf{p}^{\nu_3}$	=	$P(X_2 \mid X_0 = 0)$	=	$\{.5, .5\}$
$\mathbf{p}^{\nu_4}$	=	$P(X_2 \mid X_0 = 1)$	=	$\{.4, .6\}$
$\mathbf{p}^{\nu_5}$	=	$P(X_3 \mid X_0 = 0, X_1 = 1)$	=	$\{.9, .1\}$
$\mathbf{p}^{\nu_6}$	=	$P(X_3 \mid X_1 = 0)$	=	$\{.8, .2\}$
$\mathbf{p}^{\nu_7}$	=	$P(X_3 \mid X_0 = 1, X_1 = 1)$	=	$\{.5, .5\}$
$\mathbf{p}^{\nu_8}$	=	$P(X_4)$	=	$\{.2, .8\}$
$\mathbf{p}^{\nu_9}$	=	$P(X_5 \mid X_4 = 1)$	=	$\{.2, .8\}$
$\mathbf{p}^{\nu_{10}}$	=	$P(X_5 \mid X_4 = 0)$	=	$\{.7, .3\}$
$\mathbf{p}^{\nu_{11}}$	=	$P(X_6 \mid X_4 = 1, X_5 = 1)$	=	$\{.6, .4\}$
$\mathbf{p}^{\nu_{12}}$	=	$P(X_6 \mid \{X_4 = 1, X_5 = 0\} \vee \{X_4 = 0\})$	=	$\{.1, .9\}$
$\mathbf{p}^{\nu_{13}}$	=	$P(X_7 \mid X_4 = X_5)$	=	$\{.5, .5\}$
$\mathbf{p}^{\nu_{14}}$	=	$P(X_7 \mid X_4 \neq X_5)$	=	$\{.2, .8\}$

Table 3.2. One possible PDG-parametrisation of the structure in Fig. 3.6(b) and the probabilistic interpretations of the parameters.

Definition 3.17 (Reach)
Let $D = \langle G, \theta \rangle$ be a PDG over variables $\mathbf{X}$ w.r.t. forest F. A node ν in G labelled with X_i is reached *by $\mathbf{x} \in R(\mathbf{X})$ if*

- *ν is a root, or*
- *$X_i \in ch_F(X_j)$, $\nu' \in V_j$, ν' is reached by $\mathbf{x}$ and $\nu = succ(\nu', X_i, \mathbf{x}[X_j])$.*

Proposition 3.2
Let G be a PDG structure over variables $\mathbf{X}$, then for any joint state $\mathbf{x} \in R(\mathbf{X})$ and any variable $X_i \in \mathbf{X}$, $\mathbf{x}$ reaches a single parameter-node $\nu \in V_i$.

Proof: Proposition 3.2 can be proved by induction in the depth of G. When G has depth 1 only a single parameter-node exists and is then trivially the unique node reached by every $\mathbf{x} \in \mathbf{X}$. Assume Proposition 3.2 is true for structure G. Now, construct structure G' by adding a new variable X_j as leaf under X_i in the forest. Then, for any instance $\mathbf{x} \in R(\mathbf{X})$ a single node ν is reached in V_i, and by the definition of a PDG-structure (Def. 3.14), a single node $\nu' \in V_j$ will be reached by $\mathbf{x}$, namely the node $\nu' = succ(\nu, X_j, \mathbf{x}[X_i])$. □

We denote by $reach(i, \mathbf{x})$ the single parameter-node $\nu \in V_i$ reached by $\mathbf{x}$.

Example 3.8
Consider the PDG-structure of Figure 3.6(b), and the full instantiation $\mathbf{x} = 01100111$ (i.e., $\mathbf{x}[X_0] = 0$, $\mathbf{x}[X_1] = 1$ etc.). $reach(i, \mathbf{x})$ is then:

i	0	1	2	3	4	5	6	7
$reach(i, \mathbf{x})$	ν_0	ν_1	ν_5	ν_3	ν_8	ν_{10}	ν_{12}	ν_{14}

Definition 3.18 (Path)
Let $D = \langle G, \theta \rangle$ be a PDG over variables $\mathbf{X}$. Let $\nu \in V_i$, $pa^*_F(X_i) \subseteq \mathbf{Y} \subseteq \mathbf{X}$. Then

$$Path(\nu, \mathbf{Y}) := \{\mathbf{y} \in R(\mathbf{Y}) : \nu = reach(i, \mathbf{x}) \text{ and } \mathbf{x}[\mathbf{Y}] = \mathbf{y}\}. \tag{3.23}$$

Example 3.9
Consider the PDG-structure of Figure 3.6(b). In this structure we have:

$$Path(\nu_6, \{X_0, X_1\}) = \{(0,0), (1,0)\},$$

by which we see that whether ν_6 is on the path defined by $\mathbf{x}$ only depends on whether $\mathbf{x}[X_1] = 0$, and is independent of the value of any other variable.

We define the real valued function f_G represented by RFG $D = \langle G, \theta \rangle$ as follows:

Definition 3.19
Let $D = \langle G, \theta \rangle$ be an RFG over variables $\mathbf{X}$ w.r.t. forest F, $\nu \in V_i$ and $ch_F(X_i) = \{Y_1, \ldots, Y_l\}$. Define function f^ν_G recursively on $R(\mathbf{X})[de^*_F(X_i)]$ as:

$$f^\nu_G(x_{i,h}, \mathbf{z}_1, \ldots, \mathbf{z}_l) := p^\nu_h \prod_{j=l}^{l} f_G^{succ(\nu, Y_j, x_{i,h})}(\mathbf{z}_j), \tag{3.24}$$

where $x_{i,h} \in R(X_i)$ and $\mathbf{z}_j \in R(\mathbf{X})[ch_F(Y_j)]$. The base case of (3.24) is when X_i is a leaf of F and, therefore, $de^*_F(X_i) = \{X_i\}$ and we get:

$$f^\nu_G(x_{i,h}) := p^\nu_h, \tag{3.25}$$

for $\nu \in V_i$. Define the function f_G:

$$f_G(\mathbf{x}) := \prod_{\nu : \nu \text{ is root}} f^\nu_G(\mathbf{x}). \tag{3.26}$$

Example 3.10
Consider the PDG of Figure 3.6(b) with the parametrisation given in Table 3.2. In this model, we calculate $f^{\nu_9}_G(\mathbf{x})$ where $\mathbf{x}[X_5] = 1$, $\mathbf{x}[X_6] = 0$ and $\mathbf{x}[X_7] = 1$ as:

$$\begin{aligned} f^{\nu_9}_G(\mathbf{x}) &= p^{\nu_9}_2 \cdot f^{\nu_{11}}_G(\{0\}) \cdot f^{\nu_{13}}_G(\{1\}) \\ &= p^{\nu_9}_2 \cdot p^{\nu_{11}}_1 \cdot p^{\nu_{13}}_2 \\ &= 0.8 \cdot 0.6 \cdot 0.5 = 0.24 \end{aligned}$$

Proposition 3.3
Let $D = \langle G, \theta \rangle$ be a PDG model over variables $\mathbf{X}$ w.r.t. variable forest F. Function f_G defines a probability distribution P^D over $\mathbf{X}$.

Proof: We need to show that 1) $0 \leq P^D(\mathbf{x}) \leq 1$ and 2) $\sum_{\mathbf{x}\in R(\mathbf{X})} P^D(\mathbf{x}) = 1$.

1) First, note that as P^D is a product over factors that are all between 0 and 1, hence P^D must be between 0 and 1.

2) Next, notice that:

$$\begin{aligned}\sum_{\mathbf{x}\in R(\mathbf{X})} P^D(\mathbf{x}) &= \sum_{\mathbf{x}\in R(\mathbf{X})} \prod_{\substack{\nu:\text{root}\\ \text{in } D}} f_G^\nu(\mathbf{x}) \\ &= \prod_{\substack{\nu:\text{root}\\ \text{in } D}} \sum_{\substack{\mathbf{x}'\in\\ R(de_F^*(X_i))}} f_G^\nu(\mathbf{x}'),\end{aligned}$$

where variable X_i generating the set $\mathbf{x} \in R(de_F^*(X_i))$ is the variable represented by the single parameter-node ν, and therefore the root of a variable tree. Then, to prove $\sum_{\mathbf{x}\in R(\mathbf{X})} P^D(\mathbf{x}) = 1$ we only need to prove that for any root variable X_i:

$$\sum_{\mathbf{x}\in R(de_F^*(X_i))} f_G^\nu(\mathbf{x}) = 1, \tag{3.27}$$

where $\{\nu\} = V_i$. This can be proved by induction in the depth of the tree. Assume that (3.27) is true for a PDG structure G over variables $\mathbf{X}$. Construct PDG structure G' by adding a new leaf-node X_i to the variable forest underlying G, let $\mathbf{X}' = \mathbf{X} \cup X_i$ and let $|R(X_i)| = k_i$. The sum for $f_{G'}^\nu$ can be constructed as:

$$\begin{aligned}\sum_{\mathbf{x}'\in R(\mathbf{X}')} f_{G'}^\nu(\mathbf{x}') &= \sum_{\mathbf{x}\in R(\mathbf{X})} \left(f_G^\nu(\mathbf{x}) \sum_{x_{i,h}\in R(X_i)} p_h^{\nu'} \right) \\ &= \sum_{\mathbf{x}\in R(\mathbf{X})} f_G^\nu(\mathbf{x}) \cdot 1 \\ &= 1\end{aligned}$$

where $\nu' = succ(reach(j, \mathbf{x}), X_i, \mathbf{x}[X_j])$. □

In addition to the recursive definition of P^D above, Jaeger (2004) provides the following two alternative characterisation of the P^D:

Proposition 3.4
(Jaeger, 2004, Proposition 2.5(A)) Let $D = \langle G, \theta\rangle$ be a PDG over variables $\mathbf{X}$ (w.r.t. forest F), then:

$$P^D(\mathbf{x}) = \prod_{X_i\in\mathbf{X}} p_{\mathbf{x}[X_i]}^{reach(i,\mathbf{x})}. \tag{3.28}$$

Proof: Equation (3.28) follows immediately from equations (3.24) and (3.26). □

Proposition 3.5
(Jaeger, 2004, Proposition 2.5(B)) Let $D = \langle G, \theta \rangle$ be a PDG over random variables $\mathbf{X}$ w.r.t. forest F. Let $G \setminus X_i$ denote the PDG structure obtained from G by removing all nodes labelled with some $X_j \in de^*_F(X_i)$. For any $\nu \in V_i$, and any $\mathbf{x} \in Path(\nu, \mathbf{X})$ then

$$P^D(\mathbf{x}) = f_{G \setminus X_i}(\mathbf{x}[\mathbf{X} \setminus de^*_F(X_i)]) \cdot f^\nu_G(\mathbf{x}[de^*_F(X_i)]). \tag{3.29}$$

Proof: Note that $\mathbf{x}[\mathbf{X} \setminus de^*_F(X_i)]$ will reach exactly the same nodes for $\mathbf{X} \setminus de^*_F(X_i)$ in $G \setminus X_i$ as $\mathbf{x}$ in G. Also, note that when $\mathbf{x} \in Path(\nu, \mathbf{X})$ and $\nu \in V_i$ then $\mathbf{x}[de^*_F(X_i)]$ reaches the same nodes in the sub-graph of G rooted at ν as those reached by $\mathbf{x}$ in G. Therefore:

$$f_{G \setminus X_i}(\mathbf{x}[\mathbf{X} \setminus de^*_F(X_i)]) = \prod_{X_j \in \mathbf{X} \setminus de^*_F(X_i)} p^{reach(j,\mathbf{x})}_{\mathbf{x}[X_j]}, \tag{3.30}$$

and

$$f^\nu_G(\mathbf{x}[de^*_F(X_i)]) = \prod_{X_j \in de^*_F(X_i)} p^{reach(j,\mathbf{x})}_{\mathbf{x}[X_j]}. \tag{3.31}$$

From (3.30) and (3.31) the following can be derived:

$$\begin{aligned} &f_{G \setminus X_i}(\mathbf{x}[\mathbf{X} \setminus de^*_F(X_i)]) \cdot f^\nu_G(\mathbf{x}[de^*_F(X_i)]) = \\ &\prod_{X_j \in de^*_F(X_i)} p^{reach(j,\mathbf{x})}_{\mathbf{x}[X_j]} \prod_{X_j \in \mathbf{X} \setminus de^*_F(X_i)} p^{reach(j,\mathbf{x})}_{\mathbf{x}[X_j]} = \\ &\prod_{X_i \in \mathbf{X}} p^{reach(i,\mathbf{x})}_{\mathbf{x}[X_i]} = P^D(\mathbf{x}), \end{aligned}$$

where the last equality is due to Proposition 3.4. □

3.3.1 The PDG Dependency Model

A PDG structure encodes independence relations that are context specific. A parameter-node ν in a PDG-structure partitions $R(\mathbf{X})$ into $Path(\nu, \mathbf{X})$ and its complement.

Proposition 3.6
(Jaeger, 2004, Proposition 3.2) Let $D = \langle G, \theta \rangle$ be a PDG over discrete random variables $\mathbf{X}$ w.r.t. forest F. Let $\nu \in V_i$, $\mathbf{Y} = pa^*_F(X_i)$. Then for all $\mathbf{y} \in Path(\nu, \mathbf{Y})$:

$$\mathbf{p}^\nu = P^D(X_i | \mathbf{Y} = \mathbf{y}) = P^D(X_i | Path(\nu, \mathbf{Y})) \tag{3.32}$$

Further, we identify the local function f^ν_G defined in (3.31) as:

$$f^\nu_G = P^D(de^*_F(X_i) | \mathbf{Y} = \mathbf{y}) = P^D(de^*_F(X_i) | Path(\nu, \mathbf{Y})) \tag{3.33}$$

Proof: We first prove eq. (3.32) then (3.33).

(3.32): By the fundamental rule of conditional probability we construct $P^D(X_i|\mathbf{Y}=\mathbf{y}) = \frac{P^D(X_i,\mathbf{Y}=\mathbf{y})}{P^D(\mathbf{Y}=\mathbf{y})}$. To construct the joint marginal $P^D(X_i = x_{i,h}, \mathbf{Y}=\mathbf{y})$, we sum over $\mathbf{U} = \{\mathbf{x} \in R(\mathbf{X}) : \mathbf{x}[X_i] = x_{i,h} \text{ and } \mathbf{x}[\mathbf{Y}] = \mathbf{y}\}$:

$$P^D(X_i = x_{i,h}, \mathbf{Y}=\mathbf{y}) = \sum_{\mathbf{x}' \in \mathbf{U}} \prod_{X_j \in \mathbf{X}} p_{\mathbf{x}'[X_j]}^{reach(j,\mathbf{x}')} \tag{3.34}$$

All $\mathbf{x}' \in \mathbf{U}$ reaches the same parameter-node for any $X_l \in \{X_i \cup \mathbf{Y}\}$ as $\mathbf{Y} = pa_F^*(X_i)$. Let this parameter-node be denoted ν_l, we can then extract the common factor $\prod_{X_l \in \{X_i \cup \mathbf{Y}\}} p_h^{\nu_l}$ (where $\mathbf{x}'[X_l] = x_{l,h}$, $\mathbf{x}' \in \mathbf{U}$) from the sum in (3.34), which can then be expressed as:

$$P^D(X_i = x_{i,h}, \mathbf{Y}=\mathbf{y}) = \prod_{X_l \in \{X_i \cup \mathbf{Y}\}} p_h^{\nu_l} \sum_{\mathbf{x}' \in \mathbf{U}} \prod_{\substack{X_k \in \\ \{\mathbf{X} \setminus \{X_i \cup \mathbf{Y}\}\}}} p_{\mathbf{x}'[X_k]}^{reach(k,\mathbf{x}')} \tag{3.35}$$

$$= \prod_{X_l \in \{X_i \cup \mathbf{Y}\}} p_h^{\nu_l}. \tag{3.36}$$

Through a similar derivation, we can show that:

$$P^D(\mathbf{Y}=\mathbf{y}) = \prod_{X_j \in \mathbf{Y}} p_{\mathbf{y}[X_j]}^{reach(j,\mathbf{y})}. \tag{3.37}$$

The division then cancels all factors except from $p_h^{\nu_i}$.

(3.33): Notice that:

$$f_G^{\nu}(\mathbf{x}) = \prod_{de_F^*(X_i)} p_{\mathbf{x}[X_i]}^{reach(i,\mathbf{x})}.$$

Therefore, the proof follows similar arguments as the proof of (3.32) above. □

A set of nodes V_i in a PDG structure over variables $\mathbf{X}$ generates the partitioning consisting of the sets $\{\mathbf{x} \in R(\mathbf{X}) : \mathbf{x} \in Path(\nu, \mathbf{X})\}(\nu \in V_i)$, and we will denote this partition $\mathscr{A}(V_i)$. Using such partitions we characterise the independencies encoded by a PDG structure in Proposition 3.7.

Proposition 3.7
(Jaeger, 2004, Proposition 3.3) The probability distribution P^D represented by a PDG $D = \langle G, \theta \rangle$ satisfies the conditional independence relations:

$$P^D(X_i|\mathbf{X} \setminus de_F^*(X_i)) = P^D(X_i|pa_F^*(X_i)) = P^D(X_i|\mathscr{A}(V_i)). \tag{3.38}$$

A PDG structure G therefore defines the dependency model $M(G)$ including the independence relations:

$$M(G) = \{X_i \perp\!\!\!\perp X_j | \mathscr{A}(V_i) : X_j \in \{\mathbf{X} \setminus de_F^*(X_i)\}, X_i \in \mathbf{X}\}. \tag{3.39}$$

Proposition 3.8
Let F be a variable forest over variables $\mathbf{X}$, and let $X_i, X_j \in \mathbf{X}$ be contained in different trees. Then any PDG model with underlying variable forest F includes the marginal independence $X_i \perp\!\!\!\perp X_j$.

Proof: Let X_k be the root of the tree containing X_i, then by (3.39) we have that $X_k \perp\!\!\!\perp X_j | \mathscr{A}(V_k)$ and $X_i \perp\!\!\!\perp X_j | \mathscr{A}(V_i)$. As X_k is root, $\mathscr{A}(V_k)$ is the trivial partition $\{\mathbf{\Omega}\}$, and therefore $\mathscr{B} = \mathscr{I}(\mathscr{A}(V_k), \mathscr{B})$ for any other partition $\mathscr{B}$. Then, $X_i \perp\!\!\!\perp X_j | \mathscr{I}(\mathscr{A}(V_k), \mathscr{A}(V_i))$ is true, and contraction (Axiom 2.7) then implies:

$$X_i \perp\!\!\!\perp X_j | \mathscr{I}(\mathscr{A}(V_k), \mathscr{A}(V_i)) \wedge X_k \perp\!\!\!\perp X_j | \mathscr{A}(V_k) \Rightarrow \{X_i, X_k\} \perp\!\!\!\perp X_j | \mathscr{A}(V_k). \tag{3.40}$$

Finally, by decomposition $X_i \perp\!\!\!\perp X_j | \mathscr{A}(V_k)$ and as $\mathscr{A}(V_k)$ is the trivial partition, this is a marginal independence: $X_i \perp\!\!\!\perp X_j$. □

Proposition 3.9
Let X_i, X_j and X_k be members of the same tree T in variable forest F, let T branch at X_k and let X_i and X_j be in separate sub-branches underneath X_k. Then any PDG model w.r.t. variable forest F will encode the independence relation: $X_i \perp\!\!\!\perp X_j | \mathscr{I}(\mathscr{A}(X_k), \mathscr{A}(V_k))$.

Proof: For $\mathbf{x} \in R(\mathbf{X})$, membership according to $\mathscr{A}(V_i)$ is independent of the value of X_j as $X_j \in \mathbf{X} \setminus de^*_F(X_i)$, and Proposition 3.9 immediately follows as an instance of Eq. (3.39). □

For a distribution P, any PDG structure G that only encodes independence relations that are also true in P is called an *I-map* of P. This is analogous to the notion of an *I-map* for BN models, discussed earlier (see Section 3.2.1). In addition, any variable forest F that supports a PDG structure G that is an *I-map* of P, is also called an *I-map* of P.

Similarly analogous to BN models, we use the notion of *faithfulness*. When PDG structure G is an *I-map* of P, and P does not contain any more independence relations than those that can be read of G, we say that P is *faithful* to G. We call a distribution P for PDG-*faithful*, iff there exists a PDG structure G such that P is *faithful* to G.

3.3.2 Inference

In this section we present an algorithm for solving inference in a PDG. Central concepts are *in-flow* and *out-flow* of a node ν in a PDG. They are defined as:

Definition 3.20
Let $D = \langle G, \theta \rangle$ be an RFG over random variables $\mathbf{X}$ w.r.t. forest F. Let $\nu \in V_i$ and $G \setminus X_i$ be as in Proposition 3.5. The inflow *of node ν (denoted $ifl(\nu)$) is defined as:*

$$ifl(\nu) := \begin{cases} \sum_{\mathbf{y} \in Path(\nu, \mathbf{X} \setminus de^*_F(X_i))} f_{G \setminus X_i}(\mathbf{y}) & \text{when } \mathbf{X} \setminus de^*_F(X_i) \neq \emptyset, \\ 1 & \text{otherwise.} \end{cases} \tag{3.41}$$

The special case $\mathbf{X} \setminus de^*_F(X_i) = \emptyset$ in eq. (3.41) only happens when F consists of a single tree rooted at X_i.

Definition 3.21
Let $D = \langle G, \theta \rangle$ be a RFG over random variables $\mathbf{X}$ w.r.t. forest F, and $\nu \in V_i$. The outflow of node ν (denoted $ofl(\nu)$) is defined as:

$$ofl(\nu) := \sum_{\mathbf{z} \in R(\mathbf{X})[de^*_F(X_i)]} f^\nu_G(\mathbf{z}). \tag{3.42}$$

Note that when D is a PDG, $ofl(\nu) = 1$ for any ν.

Lemma 3.4
Let $D = \langle G, \theta \rangle$ be an RFG over random variables $\mathbf{X}$ and let ν be a node in D, then:

$$ifl(\nu) ofl(\nu) = \sum_{\mathbf{x} \in Path(\nu, \mathbf{X})} f_G(\mathbf{x}). \tag{3.43}$$

Proof: Equation (3.43) follows immediately from Proposition 3.5 and Definitions 3.20 and 3.21. □

From Lemma 3.4, it follows that when D is a PDG *inflow* of a node is the probability of that node being reached by $\mathbf{x} \in R(\mathbf{X})$ drawn under distribution P^D.

Corollary 3.1
Let D be a PDG over variables $\mathbf{X}$, then:

$$P^D(X_i = x_{i,h}) = \sum_{\nu \in V_i} p^\nu_h ifl(\nu), \tag{3.44}$$

for any $X_i \in \mathbf{X}$.

Lemma 3.5
(Jaeger, 2004, Lemma 4.3 (a)) Let $D = \langle G, \theta \rangle$ be a RFG over random variables $\mathbf{X}$ w.r.t. forest F, and let $\nu \in V_i$ and $k_i = |R(X_i)|$. Then:

$$ofl(\nu) = \sum_{h=1}^{k_i} p^\nu_h \prod_{Y \in ch_F(X_i)} ofl(succ(\nu, Y, x_{i,h})). \tag{3.45}$$

Lemma 3.6
(Jaeger, 2004, Lemma 4.3 (b)) Let $D = \langle G, \theta \rangle$ be a RFG over random variables $\mathbf{X}$ w.r.t. forest F, and $\nu \in V_i$ where X_i is a root of some tree in F. Then:

$$ifl(\nu) = \prod_{\substack{\nu' \neq \nu \text{ and} \\ \nu' \text{ root in } D}} ofl(\nu'). \tag{3.46}$$

Algorithm 3.4 Compute *out-flow* of node ν and all node in the sub-tree rooted at ν in PDG $D = \langle G, \theta \rangle$. Global data-structure *ofl* is used to store *out-flows* and global data-structure π is used to store intermediate results needed for subsequent computation of *in-flow*.

Input: RFG D over variables $\mathbf{X}$ w.r.t. forest F, and a node $\nu \in V_i$

```
1: procedure computeOfl(D, ν)
2:     ofl(ν) := 0
3:     if ch_F(X_i) ≠ ∅ then
4:         for h = 1, ... k_i do
5:             π(ν, h) := 1
6:             for all Y ∈ ch_F(X_i) do
7:                 if ofl(succ(ν, Y, x_{i,h})) has not been computed then
8:                     computeOfl(succ(ν, Y, x_{i,h})))
9:                 π(ν, h) := π(ν, h) · ofl(succ(ν, Y, x_{i,h}))
10:            ofl(ν) := ofl(ν) + p_h^ν · π(ν, h)                ▷ Eq. (3.45)
11:    else
12:        for h = 1 ... k_i do
13:            ofl(ν) := ofl(ν) + p_h^ν
```

Lemma 3.7
(Jaeger, 2004, Lemma 4.3 (c)) Let $D = \langle G, \theta \rangle$ be a RFG over random variables $\mathbf{X}$ w.r.t. forest F, $\nu \in V_i$ where X_i is not a root of F, and $pa_F(X_i) = \{X_j\}$. Then:

$$ifl(\nu) = \sum_{h=1}^{k_j} \sum_{\substack{\nu' \in V_j: \\ \nu = succ(\nu', X_i, x_{j,h})}} [ifl(\nu') p_h^{\nu'} \prod_{Y \in ch_F(X_j) \setminus X_i} ofl(succ(\nu', Y, x_{j,h}))] \tag{3.47}$$

The *out-flow* of all nodes in a RFG can be computed by invoking the procedure `computeOfl` in Algorithm 3.4 on all roots ν of RFG structure G.

Computing outflow for root node ν in a RFG by procedure `computeOfl` (Algorithm 3.4) consists of traversing the structure of D computing (3.45) for each parameter node. For PDG/RFG structure with underlying variable forest F, the complexity is $O(k)$ where:

$$k = \sum_{X_i \in \mathbf{X}} |R(X_i)| \cdot |V_i| \cdot \max(1, |ch_F(X_i)|). \tag{3.48}$$

Computing the *in-flow* of any node and all predecessor nodes in a RFG $D = \langle G, \theta \rangle$ can be done efficiently if *out-flow* of all nodes has first been computed.

Line 14 of Algorithm 3.5 implements eq. (3.47) by using the following relation:

$$\prod_{Y \in ch_F(X_j) \setminus X_i} ofl(succ(\nu', Y, x_{j,h})) = \frac{\prod_{Y \in ch_F(X_j)} ofl(succ(\nu', Y, x_{j,h}))}{ofl(succ(\nu', X_i, x_{j,h}))}, \tag{3.49}$$

where $X_j = pa_F(X_i)$ and $\nu \in V_i$. Recall that we compute the numerator of (3.49) and store it as $\pi(\nu', h)$ during the computation of outflows in line 9 of Algorithm 3.4. Therefore, assuming

Algorithm 3.5 Compute *in-flow* of a node ν in a PDG $D = \langle G, \theta \rangle$. Assumes that *ofl* and π data-structures are updated through invoking `computeOfl` on all roots of G.

Input: RFG $D = \langle G, \theta \rangle$ where structure G is over variables $\mathbf{X}$ w.r.t. forest F, node $\nu \in V_i$

1: **procedure** `computeIfl`(D, ν)
2: **if** ν is root in G **then**
3: $ifl(\nu) := 1$
4: **for all** $\nu' \neq \nu$ and ν' is root in G **do**
5: $ifl(\nu) := ifl(\nu) ofl(\nu')$ ▷ Eq. (3.46)
6: **else**
7: $ifl(\nu) := 0$
8: $X_j := pa_F(X_i)$
9: **for all** $\nu' \in V_j$ **do**
10: **if** $ifl(\nu')$ has not been computed **then**
11: `computeIfl`(D, ν')
12: **for** $h = 1, \ldots, k_j$ **do**
13: **for all** $\nu' \in V_j$ where $succ(\nu', X_i, x_{j,h}) = \nu$ **do**
14: $ifl(\nu) := ifl(\nu) + ifl(\nu') p_h^{\nu'} \frac{\pi(\nu', h)}{ofl(\nu)}$ ▷ Eq. (3.47)

that procedure `computeOfl` has been invoked on all roots and $\pi(\nu', h)$ has been saved for all edges, we can efficiently compute (3.49).

Algorithm 3.6 Compute *in-flow* and *out-flow* of every node in a PDG.

1: **procedure** `computeIflOfl`(D)
2: **for all** roots ν_r of D **do**
3: `computeOfl`(D, ν_r)
4: **for all** leaves ν_l of D **do**
5: `computeIfl`(D, ν_l)

In procedure `computeIflOfl` (Algorithm 3.6) both *in-flow* and *out-flow* are computed for every node in the PDG.

To compute the marginal $P^D(\mathbf{Y} = \mathbf{y})$ of an arbitrary subset of variables $\mathbf{Y} \subseteq \mathbf{X}$ in a PDG $D = \langle G, \theta \rangle$, we first construct a special RFG $D_{\mathbf{Y}=\mathbf{y}}$ from D by inserting evidence $\mathbf{Y} = \mathbf{y}$ described by the simple operations of the `insertEvidence` procedure (Algorithm 3.7).

Constructing evidence RFG $D_{\mathbf{Y}=\mathbf{y}}$ by the `insertEvidence` procedure of Algorithm 3.7 has complexity $O(\sum_{X_i \in \mathbf{Y}} |V_i|)$, assuming that updating parameter vectors is done in constant time instead of the suggested loop construct in line 5.

It is clear that when $D_{\mathbf{Y}=\mathbf{y}}$ is constructed from PDG D by `insertEvidence`$(D, \mathbf{Y}, \mathbf{y})$ (Algorithm 3.7), then for any $\mathbf{x} \in R(\mathbf{X})$ we have:

$$f_{D_{\mathbf{Y}=\mathbf{y}}}(\mathbf{x}) = \begin{cases} P^D(\mathbf{x}) & \text{if } \mathbf{x}[\mathbf{Y}] = \mathbf{y} \\ 0 & \text{otherwise} \end{cases}$$

Algorithm 3.7 Construct evidence RFG from PDG D by inserting evidence $\mathbf{Y} = \mathbf{y}$.

```
1: function insertEvidence(D, Y, y)
2:     D_{Y=y} := copy(D)
3:     for all X_i ∈ Y do
4:         for all ν ∈ V_i do
5:             for all x_{i,h} ∈ R(X_i) do
6:                 if x_{i,h} ≠ y[X_i] then
7:                     set p_h^ν := 0 in D_{Y=y}
8:     return D_{Y=y}
```

If *ofl* has been computed for all roots in $D_{\mathbf{Y}=\mathbf{y}}$, we can then get $P^D(\mathbf{Y} = \mathbf{y})$ by multiplication of root outflows, which is shown in the following derivation:

$$\begin{aligned} P^D(\mathbf{Y} = \mathbf{y}) &= \sum_{\mathbf{x} \in R(\mathbf{X})} f_{D_{\mathbf{Y}=\mathbf{y}}}(\mathbf{x}) \\ &= \sum_{\mathbf{x} \in R(\mathbf{X})} \prod_{\substack{\nu \text{ root} \\ \text{in } D}} f^{\nu}_{D_{\mathbf{Y}=\mathbf{y}}}(\mathbf{x}[de^*_D(X_i)]), \end{aligned} \tag{3.50}$$

where the projection $\mathbf{x}[de^*_D(X_i)]$ is onto descendandt variables of variable X_i that is represented by root parameter-node ν. The equality of (3.50) holds becouse of Proporsition 3.5, and from the definition of out-flows (Definition 3.21) we then have:

$$P^D(\mathbf{Y} = \mathbf{y}) = \prod_{\substack{\nu \text{ root} \\ \text{in } D}} ofl(\nu) \tag{3.51}$$

The complexity of calculating $P^D(\mathbf{Y} = \mathbf{y})$ therefore consists of constructing $D_{\mathbf{Y}=\mathbf{y}}$, calculating *out-flows* in $D_{\mathbf{Y}=\mathbf{y}}$ and the multiplication of root outflows (3.51). Constructing $D_{\mathbf{Y}=\mathbf{y}}$ has complexity $O(\sum_{X_i \in \mathbf{Y}} |V_i|)$ but this is dominated by the complexity for calculating outflows (3.48). The overall complexity therefore remains $O(k)$ where k is computed by (3.48).

The *in-flows* are only necessary for calculating all posterior marginals by equation (3.44). Therefore, computing a specific query on the probability $P^D(\mathbf{Y} = \mathbf{y}|\mathbf{E} = \mathbf{e})$ can be done basically by computing outflows twice, once in $D_{\mathbf{E}=\mathbf{e}}$ to get $P^D(\mathbf{E} = \mathbf{e})$ and once in $D_{(\mathbf{Y},\mathbf{E})=(\mathbf{y},\mathbf{e})}$ to get the joint $P^D(\mathbf{Y} = \mathbf{y}, \mathbf{E} = \mathbf{e})$.

3.3.3 Representation and Effective Size

We have established that general inference in PDG models has linear time complexity in the quantity of (3.48), and we therefore use this measure as the effective size of PDG model D over variables $\mathbf{X}$ w.r.t. variable forest F:

$$size_{\mathit{eff}}(D) = \sum_{X_i \in \mathbf{X}} |R(X_i)| \cdot |V_j| \cdot \max(1, |ch_F(X_i)|). \tag{3.52}$$

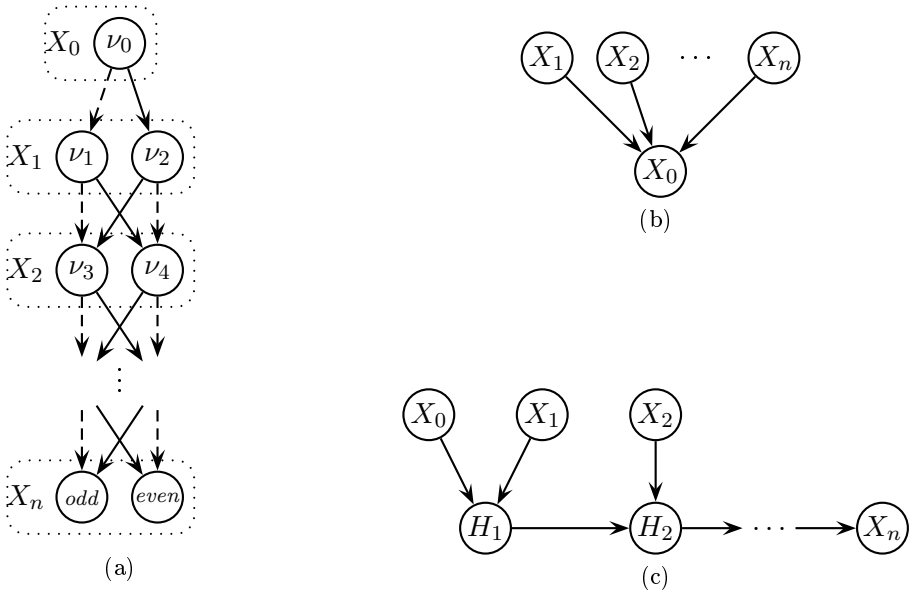


Figure 3.8. The parity distributions PDG (a) and BN (b) representation. Fig. (c) shows a BN representation with linear effective size by allowing auxiliary variables $H_1, \ldots, H_n$ to be included in the network.

We define the representational size of PDG M ($size_{rep}(M)$) as the number of free parameters defined by the model. For PDG model M over variables $\mathbf{X}$, this size measure is computed by:

$$size_{rep}(D) = \sum_{X_i \in \mathbf{X}} (|R(X_i)| - 1)\,|V_i|. \tag{3.53}$$

Therefore, the difference between $size_{\mathit{eff}}(D)$ and $size_{rep}(D)$ depends on the degree of branching of the underlying variable forest F, as:

$$size_{\mathit{eff}}(D) - size_{rep}(D) = \sum_{X_i \in \mathbf{X}} (1 + |R(X_i)|[|ch_F(X_i)| - 1])|V_i|. \tag{3.54}$$

Expressibility of PDGs

The development of the PDG language was initially an attempt to extend the language of binary decision diagrams to represent probabilistic transition systems (Bozga and Maler, 1999) and later generalised to represent discrete probability distribution over sets of variables (Jaeger, 2004). The following Example 3.11 illustrate the expressibility and potential efficiency of the PDG language, using the distribution defined by the logical "parity"-function.

Example 3.11
(Parity) Let $\mathbf{X} = \{X_0, \ldots, X_n\}$ *be a set of binary random variables, and let* P *be the joint distribution over* $\mathbf{X}$ *defining uniform marginals for every* $X_i \in \mathbf{X}$*. Let* $P(\mathbf{X} = \mathbf{x}) = 2^{-(n-1)}$

for any joint configuration $\mathbf{x}$ *with even parity (that is, the sum* $\sum_{X_i \in \mathbf{X}} \mathbf{x}[X_i]$ *is even), and* $P(\mathbf{X} = \mathbf{x}) = 0$ *otherwise. This restriction yields the conditional distributions:*

$$P(X_i = 1 | \mathbf{X} \setminus \{X_i\}) = \left(\sum_{X_i \in \mathbf{X}} X_i \right) \mod 2.$$

The parity distribution is efficiently represented by the PDG-structure over a linear order of the variables depicted in Figure 3.8(a). Two parameter-nodes for each variable summarises the parity of all variables preceding it in the linear ordering. The bottom variable X_n *is now determined exactly depending on parity of the rest of the variables.*

When representing the parity distribution by a BN model, we need a structure like the one in Figure 3.8(b) to capture the parity of every instance $\mathbf{x} \in R(\mathbf{X})$. While the PDG representation has an effective size that is linear in the number of variables $(4(n-1)+2)$, the BN will need exponentially many parameters (2^n) assuming a full tabular representation of the conditional probability distributions. From a modelling perspective, we can produce a more efficient BN model by introducing auxiliary variables, denoted by H_i $(1 \leq i \leq n)$ in Figure 3.8(c). These variables are binary, and collects intermediate parity of the variables, which makes it possible to model the distribution exactly with only $8(n-1)+4$ parameters. In general, there always exists such an efficient transformation from a PDG into a BN representing the same distribution over $\mathbf{X}$, by the introduction of latent auxiliary variables.

Theorem 3.5
(Jaeger, 2004, Theorem 5.3) Let D *be a PDG model over variables* $\mathbf{X} = \{X_0, \ldots, X_n\}$*. Then there exists a BN model* B *such that:*

1. B *is defined over variables* $\mathbf{X} \cup \{H_0, \ldots, H_n\}$,
2. $P^B(\mathbf{X}) = P^D(\mathbf{X})$*, where* P^B *is the joint distribution defined by* B*, and*
3. *there exists a junction tree of size* $O(|D|^2)$*, where* $|D|$ *is the size of* D*.*

From this theorem we can conclude that in theory BNs and PDGs provide representations that have similar efficiency. However, when learning models from data rather than constructing a BN model from a given PDG model, the problem of learning the latent auxiliary variables emerges. In the general setting, not constraining the structure of the BN nor assuming prior knowledge on the existence and cardinality of latent variables, this problem is still widely regarded as open. For solutions to special instances of the problem using more or less restrictive prior knowledge, see Karciauskas et al. (2004); Zhang (2004); Elidan and Friedman (2005).

Theorem 3.5 establishes the ability of BNs to efficiently represent distributions encoded by PDGs. Jaeger (2004) further proves that for any BN model, there exists an efficient transformation into a PDG model representing the same distribution:

Theorem 3.6
(Jaeger, 2004, Theorem 5.1) Let B *be a BN model over variables* $\mathbf{X}$*. Then there exists a PDG* D *over variables* $\mathbf{X}$ *that represents the same distribution as* B*, and* $size_{\mathit{eff}}(D) = O(size_{\mathit{eff}}(B))$*.*

The proof of Theorem 3.6 provided by Jaeger (2004), contains an algorithm that transforms a clique tree constructed from B into an equivalent PDG. This algorithm will be presented in Section 4.6.

3.4 The Naïve Bayes Model

The Naïve Bayes (NB) model represents a joint probability distribution $P(\mathbf{X})$ over random variables $\mathbf{X}$ by introducing a latent variable C that models a set of *components* $R(C)$. The NB model associates to each variable $X_i \in \mathbf{X}$ a conditional distribution $P(X_i|C)$ and to latent variable C a prior distribution $P(C)$. The NB model then represents $P(C, \mathbf{X})$ through the factorisation:

$$P(C, \mathbf{X}) = P(C) \prod_{X_i \in \mathbf{X}} P(X_i|C). \tag{3.55}$$

NB models have traditionally been used mostly for classification and clustering problems. When used for classification, the latent variable C models class membership and C has a fixed number of states, one for each possible class. Each variable $X_i \in \mathbf{X}$ models an attribute (or feature) and has a discrete state-space. The classification problem is the problem of assigning the correct class-label to an instance $E = \langle \mathbf{E}, \mathbf{e} \rangle$, where $\mathbf{E} \subseteq \mathbf{X}$, and $\mathbf{e} \in R(\mathbf{E})$. This problem is solved using a NB model by assigning to E the most likely class label c given E, that is $c = \underset{c' \in R(C)}{argmax}\ P(C = c'|\mathbf{E} = \mathbf{e})$.

In classification, C is not a latent variable outside our domain, but rather C is included in our domain by associating a known (and observed) class label with each component in a one-to-one mapping.

Unsupervised clustering is closely related to classification, but no class-labels exists. The latent C variable then models *cluster* membership, but the number of clusters (components) is typically unknown. The problem is to find the "best" number of clusters (the "best" cardinality of C), and a prior for $P(C)$. What is meant by "best" is usually problem specific, but preference is typically given to models of small cardinality that define few dense clusters.

Many studies have demonstrated the competitiveness of the NB model over more sophisticated and complex models for classification and unsupervised clustering (Cheeseman and Stutz, 1996; Langley et al., 1992; Domingos and Pazzani, 1997; Vilalta and Rish, 2003).

The NB model has recently received some attention in the area of probabilistic inference (Lowd and Domingos, 2005). Applying the NB model for general probabilistic inference and general belief updating is quite different from the two traditional (and successful) applications of the NB model discussed above. In the setting of general probabilistic inference, the learning task is then to construct an NB model with latent cluster variable, that approximates some probability distribution over the set $\mathbf{X}$ of observable variables. Moreover, we are interested in answering arbitrary probabilistic queries over $\mathbf{X}$, and not in the specific clustering provided by the model. Given a specific NB model, we would, therefore, not be interested in the cardinality of C to the extent that inference is still tractable. Nor would we be interested in the priors $P(C)$, rather we would always query the model for a joint marginal or conditional distribution

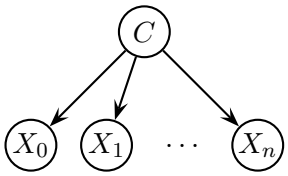


Figure 3.9. The DAG structure capturing the Naïve Bayes dependency model.

that never includes the latent variable C. We will discuss the learning problem in Chapter 4.

3.4.1 The Naïve Bayes dependency model

Definition 3.22
Let N be a NB model over variables $\mathbf{X}$ with latent variable C. The dependency model defined by N is then:

$$M(N) = \{\mathbf{A} \perp\!\!\!\perp \mathbf{B} | C\}, \tag{3.56}$$

where $\mathbf{A}, \mathbf{B} \subseteq \mathbf{X}$.

Interaction between variables are only possible indirectly through C. The cardinality of C dictates how many parameters are to be defined. With $|R(\mathbf{X})|$ components ($|R(\mathbf{X})| = |R(C)|$), there will be enough parameters to independently represent each distinct joint state of $R(\mathbf{X})$.

3.4.2 Inference

From the dependency model defined by the NB model (Def. 3.22), it is clear that the dependencies can be captured graphically by a DAG structure where C is the single parent of all $X \in \mathbf{X}$, see Figure 3.9. Then, we see that in computing the posterior $P(\mathbf{Q}|\mathbf{E} = \mathbf{e})$ for disjoint subsets $\mathbf{Q}$ and $\mathbf{E}$ of $\mathbf{X}$, all variables $\mathbf{B} = \mathbf{X} \setminus \{\mathbf{Q} \cup \mathbf{E}\}$ are barren (by Definition 3.12) and can safely be removed. This then yields the efficient computation of posterior probabilities:

$$P(\mathbf{Q} = \mathbf{q}|\mathbf{E} = \mathbf{e}) = \\ \beta \sum_{c \in R(C)} P(C = c) \prod_{Q \in \mathbf{Q}} P(Q = \mathbf{q}[Q]|C = c) \prod_{E \in \mathbf{E}} P(E = \mathbf{e}[E]|C = c), \tag{3.57}$$

where β is the normalisation constant $P(\mathbf{E} = \mathbf{e})^{-1}$. The problem of belief updating in NB model M given evidence $\mathbf{E} = \mathbf{e}$ then consists of computing (for every $X_i \in \mathbf{X}$):

$$P(X_i, \mathbf{E} = \mathbf{e}) = \sum_{c \in R(C)} P(C = c) P(X_i|C = c) \prod_{E \in \mathbf{E}} P(E = \mathbf{e}[E]|C = c). \tag{3.58}$$

The complexity of (3.58) is $O(|R(C)| \cdot |\mathbf{E}|)$. Constructing all entries $R(X_i)$ in $P(X_i|\mathbf{E} = \mathbf{e})$ requires $|R(X_i)| - 1$ such computations. Then, the overall complexity of performing belief updating in NB models is $O(|R(C)| \cdot |\mathbf{E}| \cdot k)$ where $k = \sum_{X \in \{\mathbf{X} \setminus \mathbf{E}\}} (|R(X)| - 1)$. However, the product $P(C, \mathbf{E} = \mathbf{e}) = P(C) \cdot \prod_{E \in \mathbf{E}} P(E = \mathbf{e}[E]|C = c)$ can be recycled as this same

product is required in all computations of posteriors, and then only adds to the complexity once. We get $O(|R(C)| \cdot (k + |\mathbf{E}|))$, and $(k + |\mathbf{E}|)$ is maximal when $\mathbf{E} = \emptyset$ as all variables then contribute to k.

3.4.3 Representation and Effective Size

From the above discussion, we define the effective size of NB model M over discrete variables $\mathbf{X}$ with latent component variable C as:

$$size_{\mathit{eff}}(M) := |R(C)| \cdot \sum_{X_i \in \mathbf{X}} (|R(X_i)| - 1). \tag{3.59}$$

The number of free parameters that needs to be specified for NB model M, that is the representational size of M ($size_{rep}(M)$), is:

$$size_{rep}(M) = |R(C)| - 1 + |R(C)| \sum_{X_i \in \mathbf{X}} (|R(X_i)| - 1). \tag{3.60}$$

So for NB models, effective size (3.59) and representational size (3.60) is related as:

$$size_{rep}(M) = |R(C)| - 1 + size_{\mathit{eff}}(M). \tag{3.61}$$

Expressibility of the NB model

Recall the parity distribution introduced in Example 3.11. To represent the parity distribution over n variables, the NB model will need the latent variable C to have cardinality 2^n. In this way, C can be seen as representing the joint state of the n variables in $\mathbf{X}$ and the prior $P(C)$ can be configured to be 0 when the given configuration has odd parity. Thus, for each variable X_i we will need 2^n independent parameters, which yields a total effective size of the NB model of $n \cdot 2^n + (2^n - 1)$.

The NB model can represent any discrete distribution over variables $\mathbf{X}$ by fixing the cardinality of the latent variable to $|R(\mathbf{X})|$. However, in general a latent variable of this size would yield intractable inference in the NB model.

3.5 Related Work

In this chapter we have introduced three different probabilistic graphical model languages. We have introduced the independence model encoded by each language and derived complexity of performing belief updating in the models. We introduced the PDG language capable of capturing certain *context-specific* (in)dependencies that are not expressible by the DAG structure of a BN model. Many studies have previously focused on incorporating such *asymmetric* (in)dependencies as an extension to the popular BN language, we will review a few important contributions below.

Boutilier et al. (1996) propose to use a decision tree representation of local distributions in a BN model instead of the more usual full tabular representation. By using such tree

structures context specific independencies are explicitly represented. From such local tree representations, Boutilier et al. (1996) proposes a deterministic decomposition of parents by introducing suitable so-called *multiplexer*-nodes, which effectively reduces the sizes of families in the network. By reducing the size of families, Boutilier et al. (1996) shows that the impact on complexity of inference using clique tree approaches can be significant.

Cano et al. (2000) propose to use tree representations of clique potentials in general clique tree probagation. Here, the aim is not so much to represent context specific independencies that can be identified in local clique potentials, but rather to approximate the potentials by a tree representation. This approach offers a natural tradeoff between accuracy and efficiency of the inference computation: with larger trees, the approximation is more accurate while efficiency is degraded, while smaller trees provides a (potentially) less accurate approximation but faster inference.

Many extensions to the global structure of BN models to represent certain asymmetric independencies has been proposed, e.g., Bayesian Multinets (Geiger and Heckerman, 1996), Mixtures of Bayesian Networks (Thiesson et al., 1997) and Recursive Bayesian Multinets (Peña et al., 2002). Each of these languages defines a decision tree structure that contains at its leaves different BN structures. Each leaf corresponds to a different, and the difference in the above languages reduces to whether the context is decided by one of more variable and whether a latent context-defining variable is allowed. In the Bayesian Multinets proposed by Geiger and Heckerman (1996) a single hypothesis variable defines the context. The Recursive Bayesian Multinets proposed by Peña et al. (2002) defines the context using a set of variables. The framework of Mixtures of Bayesian Networks proposed by Thiesson et al. (1997) uses a latent variable to define the context, and then basically computes an average over a small set of different Bayesian Networks.

Chapter 4

Learning Probabilistic Graphical Models

The problem addressed in this chapter is the following:

> *Let* $\mathbf{X}$ *be a set of discrete random variables w.r.t. a probability space* $\langle \Omega, \mathcal{R}, P \rangle$*. Given a database* $\mathcal{D}$ *of iid samples of* $P(\mathbf{X})$*, construct a PGM* M *over* $\mathbf{X}$ *such that* P^M *provides an* accurate *and* efficient *approximation of* $P(\mathbf{X})$*.*

To assess whether M provides an *accurate* approximation of P we use a distance measure for probability distributions, and the relative distance from P to P^M is then used as a measure of accuracy. By the *efficiency* of the approximation provided by M we understand the complexity of belief updating, that is, computing all posterior marginal distributions from P^M. Both measures are important when selecting models from a single language and also for comparison of different languages for probabilistic graphical modelling.

4.1 Selecting Models and Comparing Languages

Given a specific language of probabilistic graphical models $\mathcal{L}$ and a probability distribution P (or a finite sample $\mathcal{D}$ of P), we are interested in extracting a characteristic of $\mathcal{L}$ that tell us which alternative approximations to P $\mathcal{L}$ has to offer. Such characteristics is also relevant both for comparing different languages and when selecting among alternative models from the same language.

4.1.1 Accuracy and Efficiency

Let M be a probabilistic graphical model and let P be a target distribution, where P and M are defined over the same set of discrete variables $\mathbf{X}$. Let P^M be the distribution defined by M. One standard measure for comparing probability distributions is the Kullback-Leibler

distance (KL-distance) (Cover and Thomas, 1991; Kullbach and Leibler, 1951).[1] KL-distance is an information theoretic measure that assigns a distance from a "true" distribution P to an approximation Q. For discrete distributions, it is defined as:

$$D_{KL}(P||Q) = \sum_{\mathbf{x}\in R(\mathbf{X})} P(\mathbf{x}) \log \frac{P(\mathbf{x})}{Q(\mathbf{x})}, \tag{4.1}$$

where we adopt the convention (following Cover and Thomas (1991)) that $0 \log \frac{0}{q} = 0$ for $0 \leq q \leq 1$ and $p \log \frac{p}{0} = \infty$ for $p \neq 0$, which makes (4.1) well defined for any pair of discrete distributions (not necessarily positive) over the same domain.[2]

Lemma 4.1
Let $\mathbf{X}$ be a set of discrete random variables. Let P be a fixed distribution over $\mathbf{X}$. Then, $D_{KL}(P||\cdot)$ is a function:

$$D_{KL}(P||\cdot) : \mathcal{P}_{\mathbf{X}} \rightarrow [0, \infty], \tag{4.2}$$

where $\mathcal{P}_{\mathbf{X}}$ is the set all distributions over $\mathbf{X}$. $D_{KL}(P||\cdot)$ is a continuous function on $\{Q \in \mathcal{P}_{\mathbf{X}} : Q(\mathbf{x}) = 0 \Rightarrow P(\mathbf{x}) = 0\}$.

Proof: Under the convention that $0 \cdot \log \frac{0}{q} = 0$ for $0 \leq q \leq 1$, continuity of $D_{KL}(P||Q)$ at any $\{Q : Q(\mathbf{x}) = 0 \Rightarrow P(\mathbf{x}) = 0\}$ is immediate. □

$D_{KL}(P||Q)$ is always non-negative, 0 only when $P = Q$, and asymmetrical (hence, (4.1) is not a metric). When logarithms are base 2, the information theoretical interpretation of $D_{KL}(P||Q)$ is the expected extra bits that will be communicated when a coding scheme that is optimal under the distribution of messages Q is used, in a setting where P is the true distribution of messages. From our point of view, we will interpret $D_{KL}(P||P^M)$ as a measure of in-accuracy of model M. When using M for inference, we can express the in-accuracy of the inferred posterior joint distribution $P^M(\mathbf{Q}|\mathbf{E} = \mathbf{e})$ as $D_{KL}(P(\mathbf{Q}|\mathbf{E} = \mathbf{e})||P^M(\mathbf{Q}|\mathbf{E} = \mathbf{e}))$. Then the expected inaccuracy of inferring the joint posterior of variables $\mathbf{Q}$ given that variables $\mathbf{E}$ are observed is:

$$\sum_{\mathbf{e}\in R(\mathbf{E})} P(\mathbf{E} = \mathbf{e}) D_{KL}(P(\mathbf{Q}|\mathbf{E} = \mathbf{e})||P^M(\mathbf{Q}|\mathbf{E} = \mathbf{e})). \tag{4.3}$$

$D_{KL}(P||P^M)$ is an upper bound for (4.3) (Cover and Thomas, 1991, Theorem 2.5.3), and can therefore be used as a conservative estimate for such expected inaccuracy. The entropy of discrete distribution P is defined as:

$$H(P) = -\sum_{\mathbf{x}\in R(\mathbf{X})} P(\mathbf{x}) \log P(\mathbf{x}), \tag{4.4}$$

[1] Kullback-Leibler distance is also sometimes referred to as *information divergence*, *information gain* or *relative entropy*.

[2] The convention of replacing $0 \log \frac{0}{q}$ with 0 makes sense as $\lim_{p\to 0} p \log \frac{p}{q} = 0$, and replacing $p \log \frac{p}{0}$ with ∞ when $p \neq 0$ makes sense because $\lim_{q\to 0} p \log \frac{p}{q} = \infty$ for $p > 0$. However, there exists alternative measures for comparing discrete probability distributions, that does not require paying special attention to zeros, e.g., the *Hellinger's distance*: $D_H(P||Q) = \sum_{\mathbf{x}\in\mathbf{X}} (P(\mathbf{x})^{\frac{1}{2}} - Q(\mathbf{x})^{\frac{1}{2}})^2$.

and $D_{KL}(P||P^M)$ can then be expressed as:

$$D_{KL}(P||P^M) = -H(P) - \sum_{\mathbf{x}\in R(\mathbf{X})} P(\mathbf{x}) \log P^M(\mathbf{x}). \tag{4.5}$$

We usually do not have the "true" distribution P at our disposal, but only a finite sample $\mathcal{D}$ of P.[3] We then use the empirical distribution $P^{\mathcal{D}}$ defined by maximum likelihood estimates under the assumption of multinomial sampling $\mathcal{D}$ (Agresti, 1990):

$$P^{\mathcal{D}}(\mathbf{x}) = \frac{N_{\mathbf{x}}}{|\mathcal{D}|}, \tag{4.6}$$

where $N_{\mathbf{x}}$ is the count of $\mathbf{x}$ in $\mathcal{D}$, that is, the number of instances $d \in \mathcal{D}$ where $d = \mathbf{x}$. Substituting $P^{\mathcal{D}}$ for P in equation (4.5), we then get:

$$\begin{aligned} D_{KL}(P^{\mathcal{D}}||P^M) &= -H(P^{\mathcal{D}}) - \sum_{\mathbf{x}\in R(\mathbf{X})} \frac{N_{\mathbf{x}}}{|\mathcal{D}|} \log P^M(\mathbf{x}) \\ &= -H(P^{\mathcal{D}}) - \frac{1}{|\mathcal{D}|} L(\mathcal{D}|P^M), \end{aligned} \tag{4.7}$$

where $L(\mathcal{D}|P^M)$ is the log-likelihood of $\mathcal{D}$ under P^M, defined by Eq. (2.23). As the right-hand side of (4.7) only depends on M through $L(\mathcal{D}|P^M)$, we can use $L(\mathcal{D}|P^M)$ as a meaningfull measure of accuracy of a model M learned from data $\mathcal{D}$. Furthermore, as $0 \leq D_{KL}(P^{\mathcal{D}}||P^M)$ we see that $-H(P^{\mathcal{D}})$ provides an upper bound on $\frac{1}{|\mathcal{D}|}L(\mathcal{D}|P^M)$.

In Chapter 3, we identified parameters for each of the model language introduced, in which general belief updating will be computable in linear time, making it possible to discriminate between models from different languages based on theoretical efficiency. Popular metrics for assessing the quality of a single model given a database, combines likelihood and a measure of size in a weighted sum. We refer to such metrics as penalised likelihood scores, and they have the general form:

$$S_\lambda(\mathcal{D}, M) := (1-\lambda)L(\mathcal{D}|P^M) - \lambda \cdot size(M), \tag{4.8}$$

where $size(M)$ is some measure of complexity (not always directly related to complexity of inference) and $0 \leq \lambda \leq 1$. Popular penalised likelihood scores for BN models use the representational size (number of free parameters) of the BN model as the measure of complexity. For instance, substituting $size_{rep}(M)$ for $size(M)$ in (4.8), the Bayesian Information Criterion (BIC) (Schwarz, 1978) is proportional to (4.8) with $\lambda = 1 - \frac{\log(|\mathcal{D}|)}{2+\log(|\mathcal{D}|)}$, and the Akaike Information Criterion (AIC) (Akaike, 1974) is proportional to (4.8) for $\lambda = \frac{1}{2}$.

Penalised likelihood metrics are often used to select among alternative models in a learning procedure. It may, however, not be lucrative to settle for a model that optimise the one specific (maybe arbitrarily chosen) λ tradeoff between accuracy and efficiency dictated by the score metric. Depending on the specific application domain, we might want to penalise overly complex models differently. Also, if we are to compare models from different languages, settling for one specific tradeoff may (unintentionally) give favour to models from one language over models from another language.

[3] KL-distance has been and often still is used as a criterion in developing procedures for learning PGM from data, see Chow and Liu (1968) or Beygelzimer and Rish (2003).

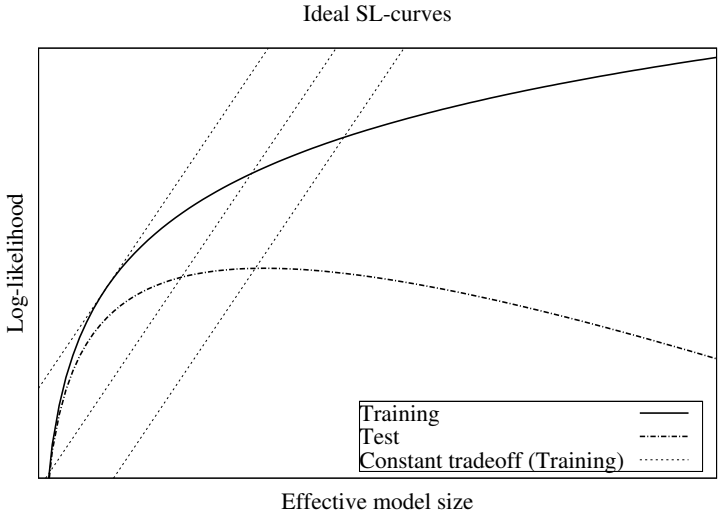


Figure 4.1. Ideal SL-curves. 'Training' plots SL coordinates for non-dominated models where log-likelihood is measured over $\mathcal{D}_A$, and 'Test' plots SL coordinates for the same models where log-likelihood is measured over $\mathcal{D}_B$. The straight lines titled 'constant tradeoff (training)' displays lines constructed by linear extrapolation of SL coordinates that score equally under that specific tradeoff.

4.1.2 SL-Curves

To evaluate our ability to learn a model M from data $\mathcal{D}$ that efficiently and accurately approximates the empirical distribution $P^{\mathcal{D}}$, we will use plots of effective size vs. log-likelihood ($L(\mathcal{D}|P^M)$) of a range models. The range of models will ideally each yield optimal S_λ score for som λ. Figure 4.1 shows idealised plots of effective model size vs. model likelihood for a range of models optimising S_λ (see Eq. 4.8) for different settings of λ. We call such plots SL-curves.

First, in Figure 4.1, the curve titled "Training" plots the likelihood over the data set used for learning (henceforth referred to as $\mathcal{D}_A$) vs. effective size. The curve titled "Test" plots the likelihood of the same models but now computed over a separate test dataset not used in the learning phase (henceforth referred to as $\mathcal{D}_B$). Each of the straight lines titled "Constant tradeoff (Training)" is constructed by extrapolation of a set of models that scores equally under some constant tradeoff. Therefore, when selecting models according to a constant λ, the optimal model can be identified in SL space as the model with SL-coordinates on curve "Training" at which the tangent has slope $\frac{\lambda}{1-\lambda}$.

SL-curves over likelihood obtained from $\mathcal{D}_A$ will show the ability of the specific model language to capture the empirical distribution $P^{\mathcal{D}_A}$. The interpretation of likelihood values obtained over $\mathcal{D}_A$ is non-trivial. While a relatively high value is preferable, any model M that successfully enumerates $\mathcal{D}_A$ (and thereby represents the empirical distribution $P^{\mathcal{D}_A}$ exactly), will receive a maximal likelihood value over $\mathcal{D}_A$ of $L(\mathcal{D}_A|P^M) = -|\mathcal{D}|H(P^{\mathcal{D}_A})$. Any model language that has the ability to represent any distribution over the observed variables is, of course, expected to approach this value asymptotically as the number of free parameters is

increased. Such models are not interesting unless we are confident that the empirical distribution $P^{\mathcal{D}_A}$ and the data generating distribution P are close to indistinguishable. Whether the assumption of $P^{\mathcal{D}_A}$ being close to P is reasonable, depends on the size of $\mathcal{D}_A$, the less data we have the less reasonable the assumption is. As data will always be limited in any practical application, models that enumerate $\mathcal{D}_A$ by capturing $P^{\mathcal{D}_A}$ perfectly, typically suffer from overfitting as any idiosyncrasies of $\mathcal{D}_A$ are captured and as a result does not generalise well to new samples from P. We define the concept of an overfitting model in Definition 4.1.[4]

Definition 4.1
Given a model language $\mathcal{L}$, a dataset $\mathcal{D}$ and partition into training data $\mathcal{D}_A$ and test data $\mathcal{D}_B$. A model $M \in \mathcal{L}$ overfit $\mathcal{D}_A$ if there exists a model $M' \in \mathcal{L}$ such that:

$$L(\mathcal{D}_A|P^M) > L(\mathcal{D}_A|P^{M'}), \text{ and} \tag{4.9}$$

$$L(\mathcal{D}_B|P^M) < L(\mathcal{D}_B|P^{M'}). \tag{4.10}$$

Likelihood values obtained over dataset $\mathcal{D}_B$, can be used to provide some stability to our conclusions and guide selection of models. $L(\mathcal{D}_B|P^M)$ is then typically used to detect overfitting $\mathcal{D}_A$.

When comparing multiple languages using SL-curves we have 2 curves for each language, one for likelihoods over $\mathcal{D}_A$ and one for likelihoods over $\mathcal{D}_B$. For each language $\mathcal{L}$, the model $M = \underset{M' \in \mathcal{L}}{argmax}\, L(\mathcal{D}_B|P^{M'})$ can be identified, and will automatically be the model amongst all models from $\mathcal{L}$ that maximise $L(\mathcal{D}_A|P^M)$ without overfitting $\mathcal{D}_A$ (according to Definition 4.1). We can then compare such optimal models from the different languages w.r.t. dominance and select the dominating model if one exists or select one of the alternatives based on requirements on accuracy or efficiency.

Consider the constructed SL curves in Figure 4.2 for languages PGM1 and PGM2. The upper curves shows log likelihood over $\mathcal{D}_A$ while the lower shows log likelihood over $\mathcal{D}_B$. We use $\mathcal{D}_B$ for guiding the selection amongst alternative models. The models that maximise log likelihood over $\mathcal{D}_B$ is indicated by M1 and M2 for PGM1 and PGM2 respectively. These are the models that would be selected (from the respective language) by a model selecting procedure that uses $\mathcal{D}_B$ to detect overfitting. As M1 and M2 have similar log-likelihood values over $\mathcal{D}_B$, and M1 has higher log-likelihood value over $\mathcal{D}_A$ than M2, then, comparing M1 and M2 we see that M2 has higher likelihood over the entire dataset. This observations should not be hastily interpreted as an indication that M1 provides the more accurate approximation of the data generating distribution. Instead, we can only conclude that M1 and M2 provide an equally accurate approximation, while M2 provides the more efficient approximation. On the level of language comparison, we can make the observation that PGM1 consistently dominates PGM2 in approximating $\mathcal{D}_A$, however, PGM1 suffers accordingly from overfitting $\mathcal{D}_A$ and accuracy on $\mathcal{D}_B$ degrades quickly. When selecting a single model in a specific scenario, this observation is less interesting. However, for a more general comparison of model language performance in a scenario where models are learned from real data, such observations are clearly relevant.

[4]Definition 4.1 is a slightly modified version of a more traditional definition (see (Mitchell, 1997, p. 67)), where log-likelihood over $\mathcal{D}_A$ and $\mathcal{D}_B$ has been substituted for *prediction error* over $\mathcal{D}_A$ and the entire dataset.

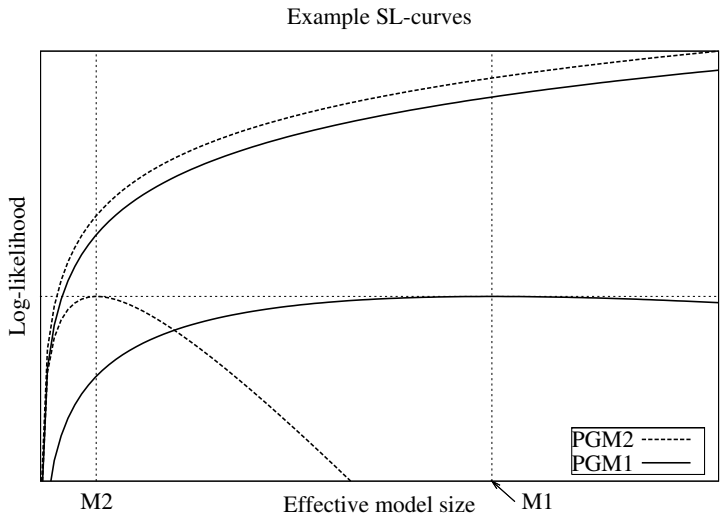


Figure 4.2. Example of SL-curves used to select models from two different languages PGM1 and PGM2. The upper curves are log-likelihood values over $\mathcal{D}_A$ while the lower curves are over $\mathcal{D}_B$.

When comparing the languages PGM1 and PGM2 rather than the models M1 and M2 using Figure 4.2, we would make the observation that PGM1 has less propensity to overfitting than PGM2.

4.1.3 Related Methodologies

Beygelzimer and Rish (2003) use tradeoff curves that display the tradeoff between tree-width and likelihood of BN models. The tree-width of a BN model is a measure of the size of the smallest junction tree representation, and is therefore equivalent to our notion of effective size of BN models. The curves used by Beygelzimer and Rish (2003) are equivalent to our SL-curves, but the motivation for the analysis is somewhat different from our analysis. Beygelzimer and Rish (2003) aims at identifying the so-called approximabillity of probability distributions by BN models. That is, a measure of how effective a BN approximation of a given distribution can be. In the present study, we aim at a comparison of different languages of probabilistic graphical models using SL-curves with likelihoods for both $\mathcal{D}_A$ and $\mathcal{D}_B$. When only considering a single language, our SL-curves (for $\mathcal{D}_A$) tell exactly the same story as the tradeoff-curves of Beygelzimer and Rish (2003).

SL-curves are closely related to curves showing *prediction error* against *complexity*, which are commonly used in machine learning for the assessment of *generalisation performance* in both unsupervised and supervised model selection (Hastie et al., 2001; Mitchell, 1997). A standard learning procedure then increases the complexity by adding parameters to the model, and eventually selects the model that minimises the prediction error on $\mathcal{D}_B$. It is natural to view log-likelihood over $\mathcal{D}_B$ as a bound on the expected accuracy in predicting new instances sampled from the generative distribution, and the model yielding maximal log-likelihood over $\mathcal{D}_B$ is then the same model that has minimal prediction error on $\mathcal{D}_B$. A slight difference, how-

ever, is that we explicitly use *effective size* that is proportional to computational complexity of general inference in the model, instead of the more common *representational complexity* typically used for such analyses.

4.2 Parameter Estimation

In this section we discuss the problem of estimating parameters of a model given a dataset of observations. Assume that for model structure M over discrete variables $\mathbf{X}$, we need to find a good parametrisation for M. Let $\mathcal{D}$ be a dataset of iid samples of joint distribution $P(\mathbf{X})$. Assume that after observing data $\mathcal{D}$ we can construct the posterior density $P(\Theta|\mathcal{D})$, effectively assigning a conditional probability to any parametrisation θ given the observed samples $\mathcal{D}$. A Bayesian approach to estimation would then select the mean of $P(\Theta|\mathcal{D})$, that is:

$$\theta' = E[\Theta|\mathcal{D}] = \int_{\Theta} \theta P(\theta|\mathcal{D}) d\theta. \tag{4.11}$$

Another Bayesian approach is the *maximum a posterior* (or MAP) estimation, where the parametrisation attaining the maximum posterior probability is selected:

$$\theta' = \underset{\theta}{argmax}\, P(\theta|\mathcal{D}). \tag{4.12}$$

The posterior $P(\theta|\mathcal{D}) = P(\mathcal{D}|\theta)P(\theta)/P(\mathcal{D})$ can be simplified by assuming that any sequence of observations is equally likely a priori, corresponding to a uniform prior $P(\mathcal{D})$ which can then be disregarded when comparing posteriors. Further, if we assume a uniform prior on parameters, the posterior $P(\theta|\mathcal{D})$ becomes proportional to the likelihood of data $P(\mathcal{D}|\theta)$. Then (4.12) becomes the popular *maximum likelihood* estimator:

$$\theta' = \underset{\theta}{argmax}\, P(\mathcal{D}|\theta). \tag{4.13}$$

If we assume multinomial sampling, the ML estimate for the conditional probability $P(\mathbf{Y} = \mathbf{y}|\mathbf{U} = \mathbf{u})$ from data $\mathcal{D}$ is given by the fraction:

$$P(\mathbf{Y} = \mathbf{y}|\mathbf{U} = \mathbf{u}) = \frac{N_{\mathbf{y},\mathbf{u}}}{N_{\mathbf{u}}}, \tag{4.14}$$

where $N_{\mathbf{y},\mathbf{u}}$ is the number of data instances $d \in \mathcal{D}$ for which $d[\mathbf{Y}, \mathbf{U}] = (\mathbf{y}, \mathbf{u})$, and $N_{\mathbf{u}} = \sum_{\mathbf{y} \in R(\mathbf{Y})} N_{\mathbf{y},\mathbf{u}}$. Therefore, when data $\mathcal{D}$ is complete (i.e., fully observed), ML estimates can be computed in closed form by simple proportions of counts. When data is incomplete, we can not compute this estimate directly and must rely on methods such as the EM algorithm, that produces an ML estimate using expected counts. For NB models, we face the problem even for complete data. The difficulty arises from estimating parameters in the presence of the latent variable C for which no observations exit. We will discuss the solution provided by the EM algorithm in dealing with the problem of incomplete data and latent variables in Section 4.4.1. For now, we will focus on the simpler task of ML estimation in BN and PDG models from complete data.

Algorithm 4.1 The procedure scores a smoothing parameter α by a cross-validation method.

Input: Model M, smoothing value α, fully observed data $\mathcal{D}$.

Output: A score for smoothing value α.

```
1: function CVScore(M, α, D)
2:     Randomly divide D into n equal size disjoint folds D_1, D_2 ..., D_n
3:     s := 0.0
4:     for all folds D_i do
5:         Let θ_i be α-smoothed ML-parameters for M estimated from D \ D_i.
6:         s := s + L(D_i|θ_i)
7:     return s/n
```

Smoothing

Pure ML estimation of parameters are often not desired, as a count of zero will yield a zero probability configuration in the model. As data is always limited, considering any event which is not observed in the data as an impossible event is never justifiable (in theory) as either the event (or the data-sample) may just be particularly unlikely in nature.

A standard method to avoiding such zero counts is to use smoothed ML-parameters, which amounts to adding a *smoothing factor* (or *pseudo count*) α to the count when calculating the estimate of $P(\mathbf{Y} = \mathbf{y}|\mathbf{U} = \mathbf{u})$:

$$P(\mathbf{Y} = \mathbf{y}|\mathbf{U} = \mathbf{u}) = \frac{N_{\mathbf{y},\mathbf{u}} + \alpha}{N_{\mathbf{u}} + \alpha \cdot |R(\mathbf{Y})|} \tag{4.15}$$

We will denote parameters calculated from eq. (4.15) α-smoothed ML-parameters.[5]

The larger the α, the more aggressive the smoothing and parameters will approach uniformity and the counts from data will vanish. Choosing α too small may not provide sufficient smoothing to cancel out the unlikely events observed in the data. A good value for α is therefore very dependent on the nature of data. By "a good value" we understand a value for which α-smoothed parameters yields a closer and more accurate approximation of the generating distribution than pure un-smoothed ML parameters.

For a given parameterised model M representing distribution P^M, the likelihood of separate test dataset $\mathcal{D}_B$ may be used as valid measure of accuracy of the approximation provided by P^M. Alternatively, instead of leaving out a subset of the dataset for validation purposes only, we can use a cross-validation approach to estimate the accuracy of an approximation. We will employ a cross-validation approach in assessing the quality of a smoothing value α. Function `CVScore` of Algorithm 4.1 assesses the quality of a α-value by cross-validation.

We will assume that `CVScore`$(M, \alpha, \mathcal{D})$ defines a unimodal function in the α argument. Empirical observations has shown that this is not an unreasonable assumption. Figure 4.3 shows `CVScore`$(M, \alpha, \mathcal{D}_A)$ and $L(\mathcal{D}_B|M)$ for a PDG model M over the variables observed in a real dataset. Not only does this plot support our assumption of unimodal `CVScore` in α,

[5] Equation (4.15) corresponds to MAP estimation of parameters with prior $P(\theta)$ following a Dirichlet distribution with parameter α for each dimension, see Heckerman (1995).

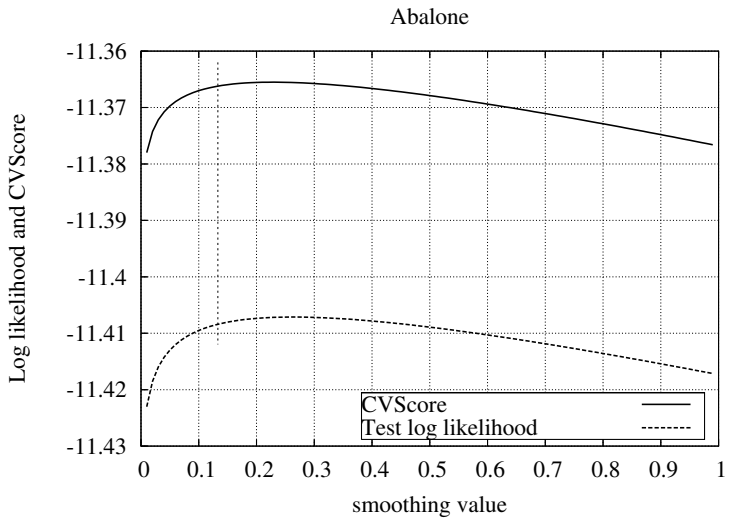


Figure 4.3. The plot shows $\texttt{CVScore}(M, \mathcal{D}_A, \alpha)$ for fixed model M and fixed data $\mathcal{D}_A$ depicted by the solid line. The dashed line plots the value of $L(D_{test}|M)$ for a separate data-sample D_{test} and for M with α-smoothed ML parametrisation. The dataset used is the Abalone dataset with $|\mathcal{D}_A| = 3758$ and $|\mathcal{D}_B| = 419$.

but also we see that $L(\mathcal{D}_B|M)$ and `CVScore` attains their maximum value in the same region of smoothing values α.

Accepting the assumption of unimodality, we will use a simple search procedure to estimate α that yields maximal `CVScore`. The procedure `tuneSmooth` (Algorithm 4.2) optimises an α using a simple narrowing search. The result of `tuneSmooth` is plotted in Figure 4.3 as a vertical dashed line.

The vertical line in the plot in Figure 4.3 shows the α value resulting from our implementation of the `tuneSmooth` procedure (Algorithm 4.2).

4.3 Learning Bayesian Network Models

This section is concerned with the problem of learning BN models from data. The recent book by Neapolitan (2003) serves both as an excellent introduction to the topic and a comprehensive reference containing many important results that have emerged over the past 10-15 years of intensive research in this specific field of automated learning.

In this section we propose an algorithm for learning BN models from data. In short, the procedure performs a stochastic search in the space of equivalence classes of BN models. Major parts of the material presented in this section is based on the ideas previously published in (Nielsen et al., 2003).

Our proposed procedure, the k-greedy Equivalence Search (or KES) procedure, is a generalisation of the Greedy Equivalence Search (or GES) procedure, first proposed by Meek

Algorithm 4.2 Given a dataset and a model M this algorithm optimises a smoothing factor by using the cross-validation score, `CVScore`.

Input: Dataset $\mathcal{D}$ and model M
Output: Optimal smoothing parameter α.

```
1: function tuneSmooth(D, M)
2:     l := 0
3:     u := α_max
4:     repeat
5:         if CVScore(M, l + ε, D) > CVScore(M, l, D) then
6:             l := l + ε
7:         if CVScore(M, u + ε, D) > CVScore(M, u, D) then
8:             u := u − ε
9:     until neither u nor l changed, or u − l is small enough.
10:    if CVScore(M, u, D) > CVScore(M, l, D) then
11:        return u
12:    else
13:        return l
```

(1997).

4.3.1 Selecting Optimal BN Models

We say that a distribution P is representable by BN dependency model $M(G)$ iff G is an *I-map* of P, which then implies that for some parametrisation θ, BN model $B = \langle \theta, G \rangle$ represents distribution $P^B = P$. We will by $B_G^{\mathcal{D}}$ denote the BN model with DAG structure G and ML parameters θ estimated from data $\mathcal{D}$.

Definition 4.2 (Local (Inclusion) Optimality)
A BN dependency model $M(G)$ is inclusion optimal w.r.t. distribution P iff P is representable by $M(G)$ and no model $M(G')$ strictly (distributionally) included in $M(G)$ exists for which P is representable.

Definition 4.3 (Global (Parameter) Optimality)
A model $M(G)$ is said to be parameter optimal w.r.t. distribution P iff P is representable by $M(G)$ and no other model with fewer free parameters is P representable.

Proposition 4.1
Let P be a distribution faithful to DAG G, then the model $M(G)$ is the unique global optimal model w.r.t. P.

Proof: As P is faithful to G, for any other model $M(H) \neq M(G)$ that can represent P it must be the case that $M(G) \subset_D M(H)$. For any such model $M(H)$, DAG H can be constructed from DAG G by a series of covered edge reversals and single edge additions (by

Definition 3.11 and Theorem 3.4). It can easily be shown that reversing a covered edge can not change the number of free parameters in the model defined by the DAG, see (Chickering, 1995). However, edge additions always will increase the number of free parameters. Therefore $M(H)$ must necessarily contain more free parameters than $M(G)$, which proves unique global parameter optimality of $M(G)$. □

For learning procedures that traverse the space of equivalence classes representing each equivalence class by a DAG, it is desirable that the score function does not discriminate between equivalent DAGs, and instead assign the same score to equivalent models. We call such score functions *score equivalent*.

Definition 4.4 (Score Equivalence)
Score function S is score equivalent *iff for any pair of DAGs G and H where $G \approx H$ it is the case that $S(\mathcal{D}, B_G^{\mathcal{D}}) = S(\mathcal{D}, B_H^{\mathcal{D}})$.*

Generic score functions like S_λ (equation (4.8)) discussed in Section 4.1 are typically used to assess the quality of BN models. For recovering a model that represents the data generating distribution, consistency of the score function is important.

Definition 4.5 (Consistent Score Functions)
Let $\mathcal{D}$ be a dataset of iid samples of a positive discrete probability distribution $P(\mathbf{X})$. A score function for BN models S is then consistent if, asymptotically as $|\mathcal{D}| \to \infty$, the following holds:

1. *If DAG G is an* I-map *of P while H is not, then $S(\mathcal{D}, B_G^{\mathcal{D}}) > S(\mathcal{D}, B_H^{\mathcal{D}})$.*
2. *If both G and H are* I-maps *of P but $size_{rep}(M(G)) < size_{rep}(M(H))$, then $S(\mathcal{D}, B_G^{\mathcal{D}}) > S(\mathcal{D}, B_H^{\mathcal{D}})$.*

For learning procedures, that traverse the space of DAGs by local transformations such as single edge addition and removal operations, the requirement of local consistency is important.

Definition 4.6 (Locally Consistent Score Functions)
Let $\mathcal{D}$ be a dataset of iid samples of a positive discrete probability distribution $P(\mathbf{X})$. Let G be a DAG over $\mathbf{X}$ and let G' be the DAG constructed from G by adding the edge $X_i \to X_j$. A score function for BN models S is then locally consistent if, asymptotically as $|\mathcal{D}| \to \infty$, (4.16) and (4.17) below hold:

$$X_i \not\perp\!\!\!\perp X_j | pa_G(X_j)[P] \Rightarrow S(\mathcal{D}, B_{G'}^{\mathcal{D}}) > S(\mathcal{D}, B_G^{\mathcal{D}}) \tag{4.16}$$

$$X_i \perp\!\!\!\perp X_j | pa_G(X_j)[P] \Rightarrow S(\mathcal{D}, B_{G'}^{\mathcal{D}}) < S(\mathcal{D}, B_G^{\mathcal{D}}) \tag{4.17}$$

Assuming DAG-faithfulness of the generative distribution, the inclusion boundary neighbourhood ensures asymptotic optimality, as shown by Castelo and Kočka (2003).

Theorem 4.1
(Castelo and Kočka, 2003, Theorem 4) Let $\mathcal{D}$ be a fully observed dataset of iid samples from a discrete joint probability distribution P. Let P be faithful to DAG structure G and let S be

Algorithm 4.3 The k-greedy Equivalence Search procedure (`KES`). S is any locally consistent score criterion and $IB^+(\cdot)$ is the set defined in (4.18).

Input: Data $\mathcal{D}$; $0 \leq k \leq 1$

Output: DAG structure of local optimal BN model.

1: **procedure** `KES`($\mathcal{D}$, k)
2: $G :=$ empty DAG model over observed variables in $\mathcal{D}$
3: $\mathbf{B} := IB^+(G, \mathcal{D})$
4: **while** $\mathbf{B} \neq \emptyset$ **do**
5: $\mathbf{C} :=$ random subset of $\mathbf{B}$ of size $\max(1, k|\mathbf{B}|)$
6: $G := \underset{G' : M(G') \in \mathbf{C}}{argmax} \; S(\mathcal{D}, B^{\mathcal{D}}_{G'})$
7: $\mathbf{B} := IB^+(G, \mathcal{D})$
8: **return** G

a locally consistent score function. Then, as $|\mathcal{D}| \to \infty$, for any DAG $H \not\approx G$ with probability 1 there is a model $M(H') \in IB(M(H))$ s.t. $S(\mathcal{D}, B^{\mathcal{D}}_H) < S(\mathcal{D}, B^{\mathcal{D}}_{H'})$.

4.3.2 Greedy and k-greedy Model Selection

The GES algorithm for selecting optimal BN models was proposed by Meek (1997), and the optimality was later proved by Chickering (2002). A generalisation of the GES algorithm was proposed by Nielsen et al. (2003), the k-greedy Equivalence Search (KES). Algorithm 4.3 gives a simple high-level formulation of the `KES` procedure. With $k = 1$, `KES` effectively reduces to GES.

We define the set $IB^+(G)$ as:

$$IB^+(G, \mathcal{D}) := \{M(G') \in IB(M(G)) \text{ s.t. } S(\mathcal{D}, B^{G'}_{\mathcal{D}}) > S(\mathcal{D}, B^{G}_{\mathcal{D}})\}. \tag{4.18}$$

where S is a locally consistent and score equivalent score function.

We will by `GES` refer to `KES` with $k = 1$.

Theorem 4.2
(Nielsen et al., 2003, Theorem 3) Let $\mathcal{D}$ be a dataset of fully observed iid samples of discrete joint probability distribution P, let P be faithful to DAG G and let $0 \leq k \leq 1$. Then, asymptotically for $|\mathcal{D}| \to \infty$, with probability 1, `KES`($\mathcal{D}$, k) returns DAG $H \approx G$.

Proof: Theorem 4.2 follows almost immediately from Theorem 4.1. As the `KES` procedure of Algorithm 4.3 at each iteration moves to a model in the inclusion boundary of the current model, that has higher score than the current model, by Theorem 4.1 `KES` will only terminate when the global optimal model G is reached. As the number of dependency models for any finite set of variables is finite, `KES` will eventually terminate and return G. □

The original formulation of the GES algorithm by Meek (1997) implemented a two-phased search. In the first phase only the upper inclusion boundary $UIB(G)$ was used in the generation

$P(X,Y)$

	x_0	x_1	x_2	x_3
y_0	0.22	0.03	0.22	0.03
y_1	0.03	0.22	0.03	0.22

$P(Y,Z)$

	y_0	y_1
z_0	0.35	0.15
z_1	0.15	0.35

$P(X,U)$

	x_0	x_1	x_2	x_3
u_0	0.22	0.22	0.03	0.03
u_1	0.03	0.03	0.22	0.22

$P(U,Z)$

	u_0	u_1
z_0	0.35	0.15
z_1	0.15	0.35

Table 4.1. Marginal joint distributions for the undirected selection-four-cycle distribution.

of $IB^+(G,\mathcal{D})$ (eq. (4.18)), and in the second phase only the lower inclusion boundary $LIB(G)$ was used. The original formulation could lead to superfluous addition of edges in the first (forward) phase that would then be removed in the second phase. Our formulation uses the full inclusion boundary in each step and, thereby, may avoid some superfluous additions, while leading to the same theoretical results.

The assumption of DAG faithfulness in Theorem 4.2 is a strong assumption to make on a joint probability distribution. We will give an example of a distribution for which DAG faithfulness is not satisfied and which exhibits multiple local maxima.

Example 4.1
(Nielsen et al., 2003, Example 1) Let $\mathbf{X} = \{X, Y, U, Z\}$ be a set of discrete random variables where X has 4 states and Y, U and Z are all binary. Let P be a probability distribution over $\mathbf{X}$ that satisfies the conditional independencies $X \perp\!\!\!\perp Z | \{Y, U\}[P]$ and $Y \perp\!\!\!\perp U | \{X, Z\}[P]$, and with marginal joint probability distributions given in Table 4.1. The (in)dependencies of this distribution are perfectly captured by the undirected graph in Fig. 4.4(a). This UDG is not decomposable (that is, not triangulated) and therefore no equivalent DAG model exists (Andersson et al., 1997, Corollary 4.1). Two distinct BN dependency models are inclusion optimal w.r.t. P, the DAG structures in Fig. 4.4(b) and (c) represents these models. The model in Fig. 4.4(b) contains 19 independent parameters while the model in Fig. 4.4(c) requires 23 independent parameters, therefore the global optimal model is the model in Fig. 4.4(b).

As one last note, the model in Figure 4.4(d) is a directed model that captures the distribution by including the latent selection variable S. Variable S is a special variable that will always be in one unique state for all observations, but is never included in the observations itself. It can be seen as a variable that selects the observations that are observed.

We will denote this distribution the undirected selection-four-cycle distribution.

Random parametrisation of the selection-four-cycle distribution of Example 4.1 was used for experiments by Chickering and Meek (2002) in experimenting on GES performance in the presence of multiple inclusion optimal models. The specific parametrisation we bring here was manually designed to guide a greedy search to a suboptimal model.

Meek (1995) investigates some aspects of the assumption of DAG faithfulness and first proves existence of faithful (discrete) distributions for *any* DAG structure. Furthermore, if parameters are selected at random over a uniform distribution of legal parameters, with probability 1, parameters will yield a probability distribution faithful to G (Meek, 1995).

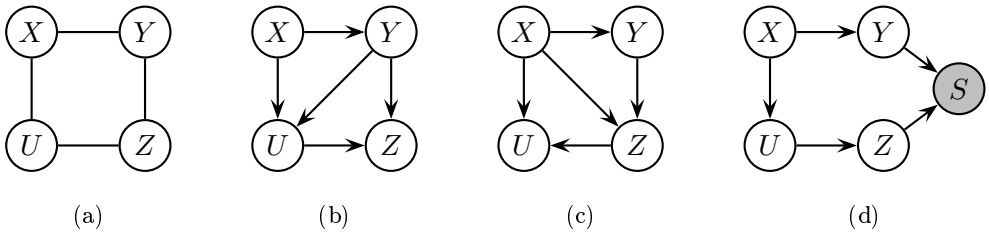


Figure 4.4. Four models that can represent the undirected select-four-cycle distribution.

However, these theoretical results are of little importance to the practical problem of learning BN models from data. It is not hard to imagine situations where this assumption is invalidated. The existence of a relationship like the one described by the DAG in Figure 4.4 (where S is the hidden selection variable), clearly invalidates the assumption of DAG faithfulness. This observation prompted Chickering and Meek (2002) to propose its replacement by the weaker assumption of satisfaction of the composition property assumption.

The composition property (or composition axiom of independence (Pearl, 1988)) is defined as:

Definition 4.7 (Composition property)
A discrete joint probability distribution P over variables $\mathbf{X}$, satisfies the composition property iff for any $X \in \mathbf{X}$ and any nonempty disjoint subsets $\mathbf{U}, \mathbf{W}$ of $\mathbf{X}$ and subset $\mathbf{Z}$ of $\mathbf{X}$:

$$X \perp\!\!\!\perp \mathbf{U}|\mathbf{Z}[P] \wedge X \perp\!\!\!\perp \mathbf{W}|\mathbf{Z}[P] \Rightarrow X \perp\!\!\!\perp \{\mathbf{U} \cup \mathbf{W}\}|\mathbf{Z}[P]. \tag{4.19}$$

Sometimes the contra-positive of (4.19) is easier to apply when working with a specific example:

$$X \not\perp\!\!\!\perp \{\mathbf{U} \cup \mathbf{W}\}|\mathbf{Z}[P] \Rightarrow X \not\perp\!\!\!\perp \mathbf{U}|\mathbf{Z}[P] \vee X \not\perp\!\!\!\perp \mathbf{W}|\mathbf{Z}[P]. \tag{4.20}$$

The distribution of Example 4.1 satisfies the composition property. The composition property is a less restrictive assumption than the assumption of DAG-faithfulness, as distributions that are DAG-faithful automatically satisfies the composition property (Pearl, 1988). The converse is not true, which the distribution in Example 4.1 exemplifies. That the model in Example 4.1 satisfies the composition property can be seen by the fact that no pairs of conditional independence relations from the model fits the left-hand side of equation (4.19). Therefore, the composition property is trivially fulfilled.

Still, the class of distributions that satisfies the composition property may be too restrictive. For instance, one relevant distribution that does not satisfy the composition property is the parity distribution (see Example 3.11). To realise this, let P be the parity distribution over binary variables $\mathbf{X}$ and let $\mathbf{Y} = \{X_i, X_j, X_l\} \subset \mathbf{X}$. We then have that $X_i \not\perp\!\!\!\perp \{X_j, X_l\}|\mathbf{X} \setminus \mathbf{Y}[P]$ but neither $X_i \not\perp\!\!\!\perp X_j|\mathbf{X} \setminus \mathbf{Y}[P]$ nor $X_i \not\perp\!\!\!\perp X_l|\mathbf{X} \setminus \mathbf{Y}[P]$, and the implication of (4.20) is then not satisfied.

Substituting the assumption of satisfying the composition property for the assumption of DAG faithfulness, Chickering and Meek (2002) prove inclusion optimality of GES. This result extends to KES which we formally state in Theorem 4.3. The proof of Theorem 4.3 proceeds in the same manner as the proof for GES inclusion optimality provided by Chickering and Meek (2002).

Theorem 4.3
(Nielsen et al., 2003, Theorem 4) Let $\mathcal{D}$ be a dataset of fully observed iid samples from a joint probability distribution P that satisfies the composition property, and let $M(H)$ be inclusion optimal w.r.t. P. Then, for any $0 \leq k \leq 1$ and $|\mathcal{D}| \rightarrow \infty$, with probability 1 KES($\mathcal{D}$,k) return DAG $G \approx H$.

Proof: We will prove Theorem 4.3 by contradiction. Assume KES($\mathcal{D}$, k) returns DAG G that is not inclusion optimal w.r.t. P. That KES returns G implies that there is no DAG $G' : M(G') \in IB(M(G))$ s.t. $S(\mathcal{D}, B_G^{\mathcal{D}}) < S(\mathcal{D}, B_{G'}^{\mathcal{D}})$. That G is not inclusion optimal w.r.t. P implies that $M(G)$ does not include P, and G is therefore not an *I-map* of P. Then, for some X_i in G it must be true that $X_i \not\perp\!\!\!\perp \{nd_G(X_i) \setminus pa_G(X_i)\} | pa_G(X_i)[P]$. By repeated application of (4.20), a singleton $X_j \in \{nd_G(X_i) \setminus pa_G(X_i)\}$ can be identified for which $X_i \not\perp\!\!\!\perp X_j | pa_G(X_i)[P]$. Adding the edge $X_j \rightarrow X_i$ to G will produce graph H, and as $X_j \in nd_G(X_i)$, H will remain a DAG. By the definition of locally consistent score functions (Definition 4.6) we get $S(\mathcal{D}, B_G^{\mathcal{D}}) < S(\mathcal{D}, B_H^{\mathcal{D}})$. By Theorem 3.4, $M(H) \in IB(M(G))$, which contradicts the assumption that KES could return G. □

Theorem 4.3 establishes inclusion optimality of KES. For a distribution satisfying the composition property, the number of inclusion optimal models may be exponential in the number of variables, while only a single (or a some small subset) of these models may be global parameter optimal. The distribution presented in Example 4.1 is an example of this. We can construct a distribution by including n copies of the undirected selection-four-cycle of Figure 4.4(a). For each such copy, 2 distinct inclusion optimal models exists, only one of which is global parameter optimal. Therefore, the distribution over all $4n$ variables would exhibit 2^n distinct local inclusion optimal models while still only one unique model is the global parameter optimal model.

The greedy traversal of the neighbourhood implemented by GES is not guaranteed to recover the global parameter optimal model. However, by relaxing greediness and choosing $k < 1$ we introduce some randomness into the search and thereby may explore a larger area of the search space. Configuring KES for maximal randomness (by setting $k = 0$) we are able to recover any inclusion optimal model.

Theorem 4.4
(Nielsen et al., 2003, Theorem 5) Let $\mathcal{D}$ be a dataset of fully observed iid samples from a discrete joint probability distribution P that satisfies the composition property. Let G be a DAG representing a BN model $M(G)$ that is inclusion optimal w.r.t. P. Then, as $|\mathcal{D}| \rightarrow \infty$, with non-zero probability, KES($\mathcal{D}$, 0) will return G.

Proof: Let $M(G)$ be any inclusion optimal model w.r.t. P. We can then prove Theorem 4.4 by constructing a sequence of models $M(G_0), \ldots, M(G_e)$, where G_0 is the empty DAG and $M(G_e) = M(G)$, and each model $M(G_i) \in IB^+(G_{i-1}, \mathcal{D})$ for $1 \leq i \leq e$.

Consider the sequence of DAGs $G_0, \ldots, G_e$, where G_0 is the empty DAG, each DAG is constructed from the immediately preceding DAG by a single edge addition, and $G_e = G$. For all $0 \leq i < e$ it is clear that $M(G_{i+1}) \in UIB(G_i)$, hence we only need to show that $S(\mathcal{D}, B^{\mathcal{D}}_{G_i}) < S(\mathcal{D}, B^{\mathcal{D}}_{G_{i+1}})$ to prove $M(G_{i+1}) \in IB^+(G_i, \mathcal{D})$. As every model in the sequence is in the *upper inclusion boundary* of the immediately preceding model, $M(G_i) \subset_P M(G_j)$ for all $0 \leq i < j \leq e$, in particular $M(G_i) \subset_P M(G)$ for any $0 \leq i < e$. As $M(G)$ is inclusion optimal w.r.t. P, no model strictly included in $M(G)$ (and therefore no model in our sequence) can represent P. For any model $M(G_i)$ where $i < e$, G_i is therefore not an *I-map* of P, and then, for some variable X:

$$X \not\perp\!\!\!\perp \{nd_{G_i}(X) \setminus pa_{G_i}(X)\} | pa_{G_i}(X)[P]. \tag{4.21}$$

However, as G is an *I-map* of P, for the same X we have:

$$X \perp\!\!\!\perp \{nd_G(X) \setminus pa_G(X)\} | pa_G(X)[P]. \tag{4.22}$$

As G_i is a subgraph of G, it is clear that $\{nd_G(X) \setminus pa_G(X)\} \subset \{nd_{G_i}(X) \setminus pa_{G_i}(X)\}$. It therefore follows from (4.21) and (4.22) (by the *block independence lemma* (2.20)), that:

$$X \not\perp\!\!\!\perp \{pa_G(X) \setminus pa_{G_i}(X)\} | pa_{G_i}(X)[P]. \tag{4.23}$$

We can then (using (4.20)) identify a singleton $Y \in \{pa_G(X) \setminus pa_{G_i}(X)\}$ for which $X \not\perp\!\!\!\perp Y | pa_{G_i}(X)[P]$. Adding the edge $Y \to X$ to G_i producing G_{i+1} will (asymptotically for $|\mathcal{D}| \to \infty$) yield a score improvement for any locally consistent score function, hence $M(G_{i+1}) \in IB^+(M(G_i))$ for all $0 \leq i < e$. □

4.3.3 Implementation

In this section we discuss some important issues relating to the implementation of `KES` (Alg. 4.3). In particular we will prove consistency od the general penalised likelihood score function and discuss our approach to generating $IB^+(G, \mathcal{D})$ (4.18).

The λ-score for BN models

For BN models, we will use S_λ as a score function with $size_{rep}$ as penalty. Let B be the parametrised BN model, then we define the score:

$$S^{BN}_\lambda(\mathcal{D}, B) = (1 - \lambda) L(\mathcal{D}|P^B) - \lambda size_{rep}(B). \tag{4.24}$$

Lemma 4.2
S^{BN}_λ is score equivalent for BN models for $0 \leq \lambda \leq 1$.

Proof: Chickering (1995) proves that for equivalent DAGs G and H, $L(\mathcal{D}|B_G^{\mathcal{D}}) = L(\mathcal{D}|B_H^{\mathcal{D}})$ and $size_{rep}(M(G)) = size_{rep}(M(H))$. As S_λ^{BN} is the sum of two quantities that are equivalent for equivalent models, S_λ^{BN} is itself equivalent. □

A score function for BN models is decomposable if it can be expressed as a sum over terms, each of which is only a function of one variable and its parents in the DAG structure of the BN model. As both terms of S_λ^{BN} decompose into such terms, we see that S_λ^{BN} is itself decomposable for BN models.

Lemma 4.3
S_λ^{BN} is a consistent score for BN models when $0 < \lambda < 1$.

Proof: Let $\mathcal{D}$ be iid samples from the discrete distribution $P(\mathbf{X})$. Then, with probability 1, $P^{\mathcal{D}} \to P$ when $|\mathcal{D}| \to \infty$. We prove each of the requirements of Definition 4.5 in the following:

1. Consider two DAGs G and H, and let G be an *I-map* of the generative distribution P while H is not. We then need to prove that as $|\mathcal{D}| \to \infty$:

$$S_\lambda^{BN}(\mathcal{D}, B_G^{\mathcal{D}}) - S_\lambda^{BN}(\mathcal{D}, B_H^{\mathcal{D}}) > 0. \tag{4.25}$$

Combining (4.24) and (4.25) we get:

$$\begin{aligned} S_\lambda^{BN}(\mathcal{D}, M(G)) - S_\lambda^{BN}(\mathcal{D}, M(H)) =& (1-\lambda)[L(\mathcal{D}|P^{B_G^{\mathcal{D}}}) - L(\mathcal{D}|P^{B_H^{\mathcal{D}})})] \\ & - \lambda[size_{rep}(B_G^{\mathcal{D}}) - size_{rep}(B_H^{\mathcal{D}})] \\ >& 0. \end{aligned}$$

Then, by (4.7) we get:

$$(1-\lambda)(-|\mathcal{D}| \cdot [D_{KL}(P||P^{B_G^{\mathcal{D}}}) - D_{KL}(P||P^{B_H^{\mathcal{D}}})]) > c, \tag{4.26}$$

where $c = size_{rep}(M(G)) - size_{rep}(M(H))$. For $|\mathcal{D}| \to \infty$, with probability 1 $P^{B_G^{\mathcal{D}}} \to P$ (and, therefore, $D_{KL}(P||P^{B_G^{\mathcal{D}}}) \to 0$). (4.26) is then asymptotically satisfied if:

$$(1-\lambda)|\mathcal{D}|D_{KL}(P||P^{B_H^{\mathcal{D}}}) > c, \tag{4.27}$$

for some $c > 0$. Consider the set $\mathcal{H}$ of probability distributions representable by $M(H)$. Now, construct the non-empty set $\mathcal{H}_r = \{Q \in \mathcal{H} : D_{KL}(P||Q) \leq r\}$ for some $r < \infty$.[6] By continuity of $D_{KL}(P||\cdot)$ (Lemma 4.1), $\mathcal{H}_r$ is a compact set. Then, a well known result from topology (Apostol, 1974, Theorem 4.25) guarantees that there exists a minimal element $Q' = \underset{Q \in \mathcal{H}_r}{argmin}\, D_{KL}(P||Q)$. Recall that H is not an *I-map* of P. Then $D_{KL}(P||Q')$ is positive (non-zero) and (4.27) is then satisfied for $|\mathcal{D}| \to \infty$ and $\lambda < 1$.

[6] That $\mathcal{H}_r$ will be non-empty for some $r < \infty$ is easily realised by the fact that any DAG can represent a uniform distribution, and for uniform distribution Q, $D_{KL}(P||Q) < \infty$ for any P.

2. The second requirement for consistency can be proved to be satisfied by somewhat similar arguments. When both G and H are *I-maps* of P, for $|\mathcal{D}| \to \infty$ with probability 1 $P^{B_G^{\mathcal{D}}} \to P$ and $P^{B_H^{\mathcal{D}}} \to P$, and the difference in likelihood $L(\mathcal{D}|P^{B_G^{\mathcal{D}}}) - L(\mathcal{D}|P^{B_H^{\mathcal{D}}})$ will approach 0. Then, as $0 < \lambda$, we have:

$$size_{rep}(M(G)) < size_{rep}(M(H)) \Rightarrow S_\lambda^{BN}(\mathcal{D}, B_G^{\mathcal{D}}) > S_\lambda^{BN}(\mathcal{D}, B_H^{\mathcal{D}}). \tag{4.28}$$

□

From Lemma 4.3, Corollary 4.1 immediately follows:

Corollary 4.1
Let $\mathcal{D}$ be a dataset of iid samples from joint probability distribution P, and let P be faithful to DAG G. Then, asymptotically for $\mathcal{D} \to \infty$ and any $H \not\approx G$:

$$S_\lambda^{BN}(\mathcal{D}, G) > S_\lambda^{BN}(\mathcal{D}, H). \tag{4.29}$$

Lemma 4.3 then establishes global consistency for selecting BN models according to S_λ^{BN}.

Lemma 4.4
S_λ^{BN} for BN models is a locally consistent score function.

The proof for Lemma 4.4 follows similar arguments as the proof for local consistency of the Bayesian score (Chickering, 2002, Lemma 7).

Proof: As S_λ^{BN} is decomposable, the difference $S_\lambda^{BN}(\mathcal{D}, B_G^{\mathcal{D}}) - S_\lambda^{BN}(\mathcal{D}, B_{G'}^{\mathcal{D}})$ is invariant for all pairs of DAGs that only differs in the single adjacency $X_i \to X_j$. We are, therefore, free to choose the structure common to G and G'. Let G' be a fully connected DAG. Then, $M(G') = \emptyset$ and $M(G) = \{X_i \perp\!\!\!\perp X_j | pa_G(X_j)\}$. As $M(G')$ can represent any distribution, $M(G')$is trivially an *I-map* for P. If $X_i \not\perp\!\!\!\perp X_j | pa_G(X_j)[P]$, then $M(G)$ is not an *I-map* of P and by consistency of S_λ^{BN} the implication of (4.16) is true. If $X_i \perp\!\!\!\perp X_j | pa_G(X_j)[P]$, then both $M(G')$ and $M(G)$ are *I-maps* of P and $size_{rep}(M(G)) < size_{rep}(M(G'))$, and by consistency of S_λ^{BN}, implication (4.17) is true. □

On the Choice of Size Measure

It may seem more natural (or even more fair) to use the effective size as the penalty term in the lambda score of (4.24) instead of the representational size. Especially when considering that in Chapter 5 we are going to base our comparative analysis on effective sizes. However, our reasons for not doing so are mainly the complications connected with computing the increase/reduction of the effective size locally given a local modification like addition or removal of an edge. Having a decomposable score is preferable from a practical point of view, as it yields a straightforward way of reusing computations by caching locally computed scores. Also, the theoretical results of Section 4.3.1 and Section 4.3.2 very much depends on the score being decomposable and locally consistent.

Algorithm 4.4 Given a DAG G, this algorithm produces a representative DAG for a random member of $IB(G)$

Input: DAG G
Output: Random member of $IB(G)$

```
1:  function sampleIB(G)
2:      H := G
3:      r := random integer between 0 and |E|
4:      for r times do
5:          reverse a random covered arc in H
6:      (X, Y) := random pair of nodes in H
7:      if Y ∈ adj_H(X) then
8:          Remove the adjacency (X, Y) from H.
9:      else
10:         Introduce the adjacency (X, Y) with random orientation into H without introducing a cycle.
11:     return H
```

Obviously, rebuilding a full clique tree representation whenever computing the change in score implied by a modification is not a local operation. Instead, we could consider building the clique tree incrementally during the BN learning procedure. Incremental construction and maintenance of a clique tree representation was studied by Flores et al. (2003). Given a clique tree model and a structural modification of the original BN model (add/remove a link), the procedure of Flores et al. (2003) identifies small sub-graphs (Maximal Prime Sub-graphs) of the clique tree that needs re-triangulation. In practise, this can be much simpler than rebuilding the full clique tree representation, but in the worst case it still may turn out to be equivalent to a full global re-triangulation.

We are not aware of any reliable locally (and efficiently) computable estimates for the increase in effective size resulting from a local modification to the BN model structure. For these reasons, we choose to use the representational size as the penalty in our score function for BN models.

Generating the Inclusion Boundary

Theorems 3.3 and 3.4 suggest a simple way of sampling a random member of the inclusion boundary of any DAG G. By reversing covered edges and adding or removing a single edge we will generate a DAG G' that represents a model in $IB(G)$. The function `sampleIB` (Algorithm 4.4) gives a high-level formulation of this procedure.

The `sampleIB` function of Algorithm 4.4 is able to sample any member of $IB(G)$. First, by Theorem 3.3 the sequence of r random covered edge reversals (line 5) can generate any member $G' \in \mathcal{E}(G)$. Next, by Theorem 3.4 the random addition/removal (lines 8 and 10) can generate any member of $IB(G)$. However, the sampling of DAG models equivalent to G in line 5 is not uniform, as "close" DAGs requiring only a few covered arc reversals are more

likely to be sampled than "distant" DAGs requiring more covered arc reversals. The intuition behind this observation is that only a few of the edges that needs to be reversed to get from DAG G to distant (equivalent) DAG H may be covered in G. After reversing covered edge e in G producing G', the set of covered edges will then typically have changed between G and G', but one edge remains covered in both, namely e. Therefore, there is a chance that in G', e is reversed again, effectively producing G again from G'.

The implementation of $IB^+(G, \mathcal{D})$ is based on the `sampleIB` function, which means that instead of exhaustively enumeration of $IB(G)$, we sample from $IB(G)$ sufficiently many times. The `sampleIB` procedure performs sampling with replacement from the set $IB(G)$. Let $X^{(R)}$ be the number of distinct models in a random sample of size R. That is, assuming that we draw (with replacement) R models from $IB(G)$, $X^{(R)}$ then is the number of distinct models drawn. Assuming uniform sampling, the expectation of $X^{(R)}$ is:

$$E[X^{(R)}] = \sum_{i=1}^{R-1} \left(\frac{N-1}{N} \right)^{(i-1)}, \tag{4.30}$$

where N is the size of $IB(G)$ (see Appendix B for the proof). The mean percentage of $IB(G)$ that will be represented in a sample of size R is then $\frac{E[X^{(R)}]}{N}$. Therefore, if we want to generate a random sample of average size $k \cdot N$ from $IB(G)$, we can simply draw R samples, where $\frac{E[X^{(R)}]}{N} = k$. We can not solve (4.30) directly, instead we expand the sum one term at a time and check if we are within some small error ϵ of k. Allowing for an error of ϵ is necessary for any computer implementation as otherwise we would expand the sum with infinitely many terms for $k = 1.0$. Figure 4.5 shows $\frac{E[X^{(R)}]}{N}$ for $N = 100$ against R. In the plot of Figure 4.5 we have indicated corresponding k (that is $\frac{E[X^{(R)}]}{N}$) and R values for $\epsilon = 0.001$. For example, we see that for $k = 0.8$ we will sample $R = 162$ times, and for $k = 0.9$ we will sample $R = 231$.

In `KES`, however, we need to sample $IB^+(G)$ rather than $IB(G)$. For simplicity, we first sample $IB(G)$ by the method outlined above, and then select from this sample the model with highest score. If no such model was found in the first sample, a new sample is drawn, and so forth. Eventually, we terminate the search when the full $IB(G)$ has been sampled.

This reversal of operations only has an impact on the implementation of `KES`, none of the theoretical properties of `KES` is affected by this.

The above proposed method still lacks efficient computation of $N = |IB(G)|$. This value is difficult to obtain without exhaustive enumeration. In our implementation, we approximate N by the number of edges that can be added to the empty graph over variables $\mathbf{X}$, i.e., $|\mathbf{X}|^2 - |\mathbf{X}|$.[7] This approximation is justified by the fact that any model in the inclusion boundary of DAG model G has one more or one less edge than G. On one hand this is an underestimate as more than one unique equivalence class may exist for which the same connection has been added/removed from G. On the other hand it is an overestimate as not all node connections are possible as some connections may result in cycles. In practise we have found this estimate to be adequate.

[7] Chickering (2002) proposed this estimate.

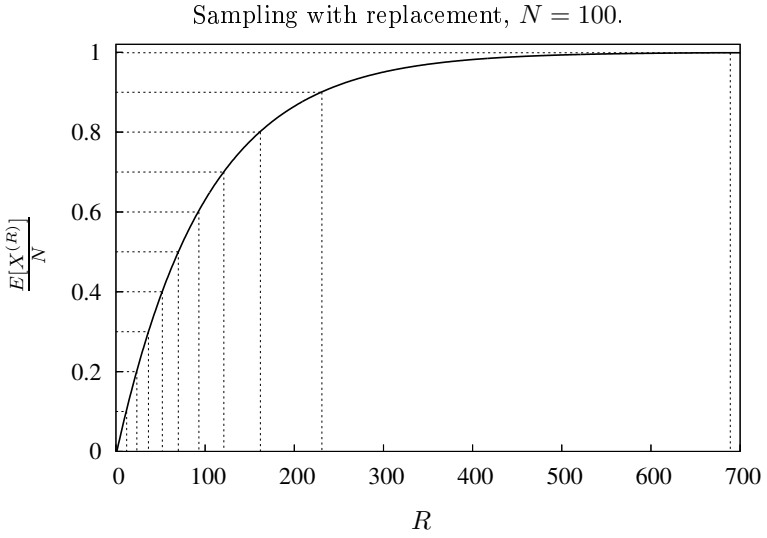


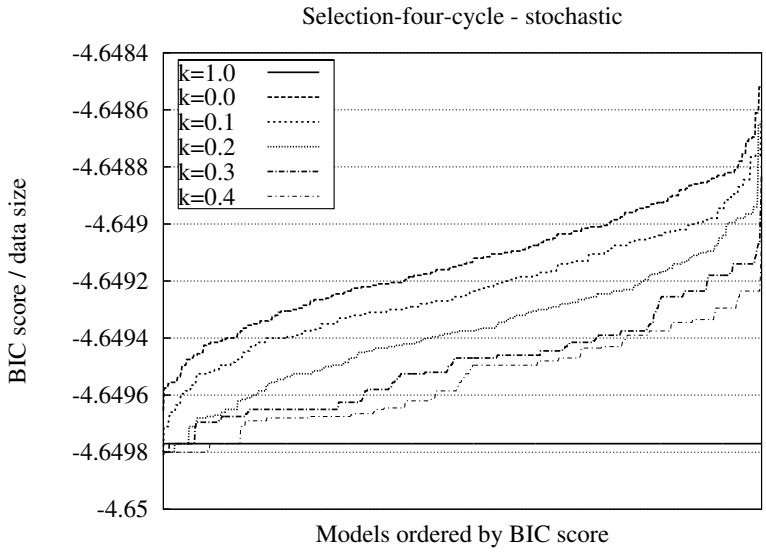
Figure 4.5. The expected fraction of distinct models sampled from a set of 100 elements, when sampling R elements with replacement. That is, $\frac{E[X^{(R)}]}{N}$ for $N = 100$ as a function of R.

4.3.4 Testing the BN Learning Procedure

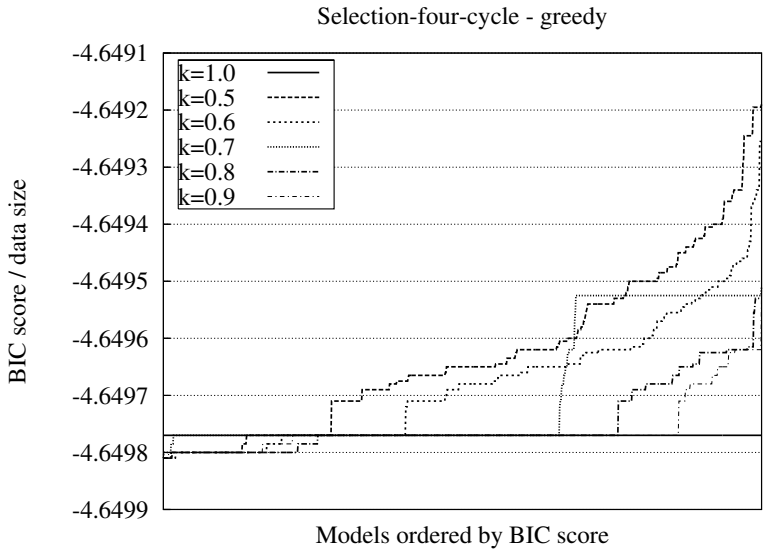
One motivation for developing a procedure that allows trading off greediness for randomness was the identification of distributions with multiple local inclusion optimal models. An example of such a distribution was the selection-four-cycle distribution used by both Chickering and Meek (2002) and Nielsen et al. (2003) and repeated here in Example 4.1.

To investigate the performance of both greedy and stochastic heuristics in search spaces containing numerous local optima, we first construct a distribution exhibiting numerous local inclusion optimal DAG models. We can construct a model representing a distribution exhibiting 2^n local inclusion optimal DAG models, by constructing the UDG model consisting of n copy's of the selection-four-cycle of Example 4.1. In this experiment, we use a model constructed in this way with $n = 10$ which then is a model over 40 random variables that exhibits 1023 local inclusion optimal DAG models and a single global optimal DAG model. We then sample 20000 instances from this model and invoked the `KES` procedure (Algorithm 4.3) using this data. We use 11 different settings of $k \in \{0.0, 0.1, \ldots, 1.0\}$, and for each setting of k, the `KES` procedure was restarted 1000 times, and we used the BIC score in all experiments. Results are displayed in the plots of Figure 4.6(a) and (b). First, Fig. 4.6(a) show the lowest settings of k yielding the more stochastic search. We have also included the deterministic and maximally greedy version with $k = 1.0$ (corresponding to the GES procedure of Chickering and Meek (2002)) for comparison. We observe that all models learnt for $k \in \{0.0, 0.1\}$ and most models for $k \in \{0.2, 0.3, 0.4\}$ attain higher BIC score than the single model obtained by GES. For $k \in \{0.4, 0.5, \ldots, 0.9\}$ (Fig. 4.6(b)), we again observe that for $k < 1.0$ we are able to recover models that attain higher BIC score than GES.

The results reported above shows that GES may gets trapped in a low quality local in-



(a)



(b)

Figure 4.6. Result of 1000 restarts of `KES` learning from data sampled from the selection-four-cycle distribution (see Example 4.1). Models are sorted in ascending order of BIC score.

clusion optimal model. This is not surprising, as the distribution from which the data was sampled is a manually constructed distribution specifically designed to trap GES. The inclusion optimal model recovered by GES is (asymptotically) the lowest scoring inclusion optimal model over the 1024 different inclusion optimal models.

To investigate the effect of learning from data sampled from a DAG faithful distribution,

we use data sampled from 2 different standard BN models:[8]

- The Alarm model represents medical knowledge of relationships between findings and diagnoses in the domain of patient care in an operating room (Beinlich et al., 1989). It contains 37 discrete variables and 46 arcs, and the structure of the model is displayed in Figure 4.10 and the table contains descriptive names for the node indexes.
- The Hailfinder model was developed by Abramson et al. (1996) as a weather forecasting system. It combines meteorological data and expert knowledge in forecasting of severe weather conditions in Northeaster Colorado. The model contains 56 discrete random variables and 66 arcs, the structure is displayed in Figure 4.11.

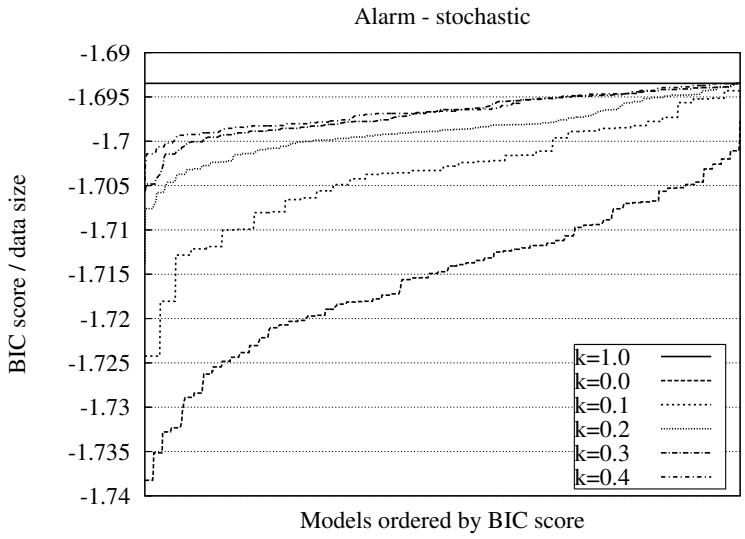
For each of the two above models, we generated databases by sampling 20000 instances. The results of the 1000 restarts of the `KES` procedure is plotted in Figure 4.7 and 4.8.

Figure 4.7(a)-(b) shows the results of learning from Alarm-sampled data. As expected, we observe that the model recovered by GES (`KES` for $k = 1.0$) is the highest scoring model, and the greedier the `KES` procedure, the better models are recovered on average over the 1000 restarts. In addition, from the plots corresponding to $k < 1.0$ we can observe that a lot of local inclusion optimal models still exists in the data. Recall that we use a score-equivalent score-function, and therefore any two models attaining different score are not equivalent. Therefore, for every different score-value in the plots of Figure 4.7 there exists a distinct inclusion optimal model. From Theorem 4.1 we see that only a single inclusion optimal model exists in the limit of large data, therefore this observation is explained by the fact that our data-sample is of limited size.

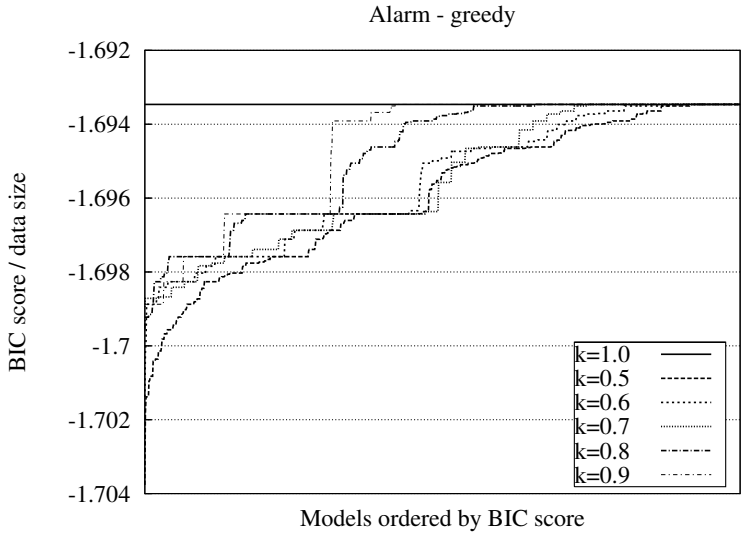
Figure 4.8(a)-(b) shows the results of learning from Hailfinder-sampled data. From Figure 4.8(b) we observe that the model recovered by the GES procedure is not the highest scoring model over all the different settings of k. This is explained by the fact that optimality of GES is an asymptotic property, and for any finite dataset we then are not guaranteed to optimality. In fact, as reported by Nielsen et al. (2003), in any practical application of `KES` using real world datasets, we often recover better models by $k < 1$. In addition, this experiment shows us that even in the cases where DAG faithfulness is a safe assumption, limited data may yield suboptimal result of GES.

Lastly, Figure 4.9 shows the average learning times for `KES` with different settings of k. It is notable that the learning time increases dramatically from an almost constant level at approximately 2.2 seconds for $k = 0.0, 0.1, \ldots, 0.9$ to approximately 4.9 seconds for $k = 1.0$. The reason is found in the way we sample the inclusion boundary and the exponential nature of (4.30), see Figure 4.5. For $k = 0.0$ up to $k = 0.9$ there are only moderate increases in the actual number of models sampled, while for $k = 1.0$ we need to increase the number of models sampled much more than for any other increase in k. This also explains why we do not see a clear increase in execution time for $k = 0.0$ to $k = 0.9$, as these execution times are all dominated by the final steps of the algorithm. In any final step we need to sample the full inclusion boundary to guarantee there are no models in the boundary of better score.

[8] Both models are obtainable from many on-line repositories, see for example `http://genie.sis.pitt.edu/networks.html`.



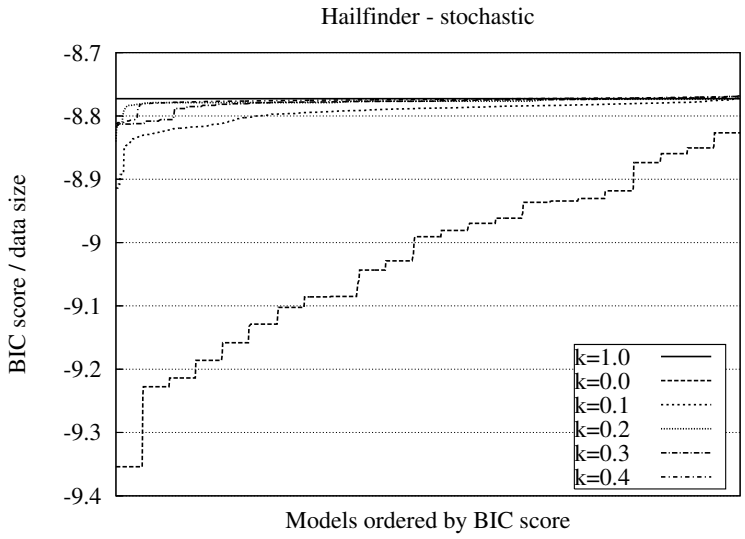
(a)



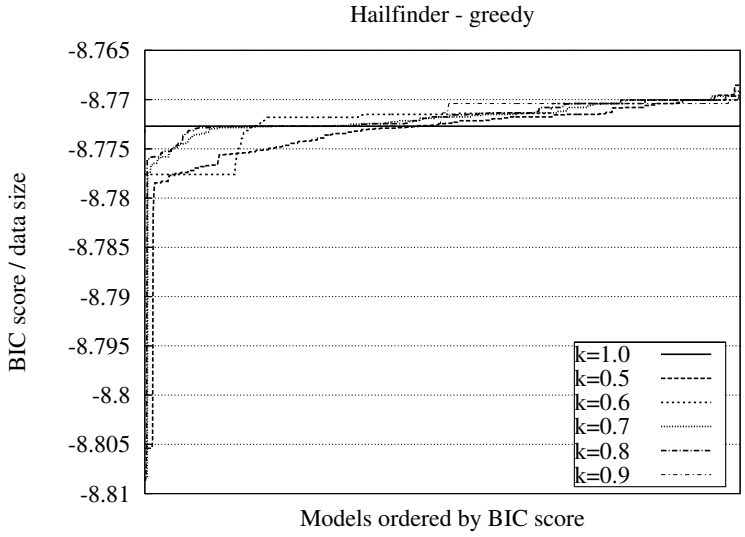
(b)

Figure 4.7. Result of 1000 restarts of `KES` learning from data sampled from the Alarm BN model. Results are sorted in ascending order of BIC score.

As an initial test, these experiments show us that our implementation of `KES` and, in particular, the sampling of the inclusion boundary (as discussed in the previous subsection), performs as expected on synthetic datasets.



(a)



(b)

Figure 4.8. Result of 1000 restarts of `KES` learning from data sampled from the Hailfinder BN model. Results are sorted by ascending order of BIC score.

4.3.5 Related Work

One of the earliest works on learning BN models include the work by Chow and Liu (1968) on learning tree structured BN models. Restricting the search to only include tree structures reduces the size of the search space dramatically from exponential in the number of variables (the case for unrestricted DAG structures) to quadratic. Chow and Liu (1968) proposes a procedure that produces a tree structured BN model that has maximal weight, where the

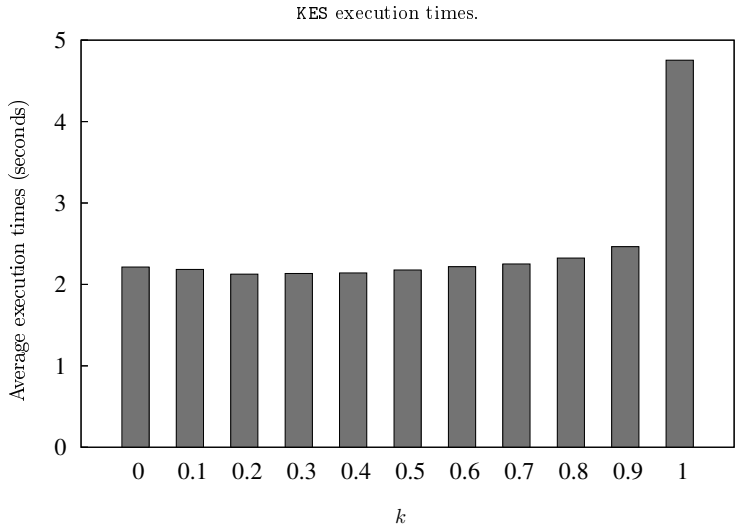
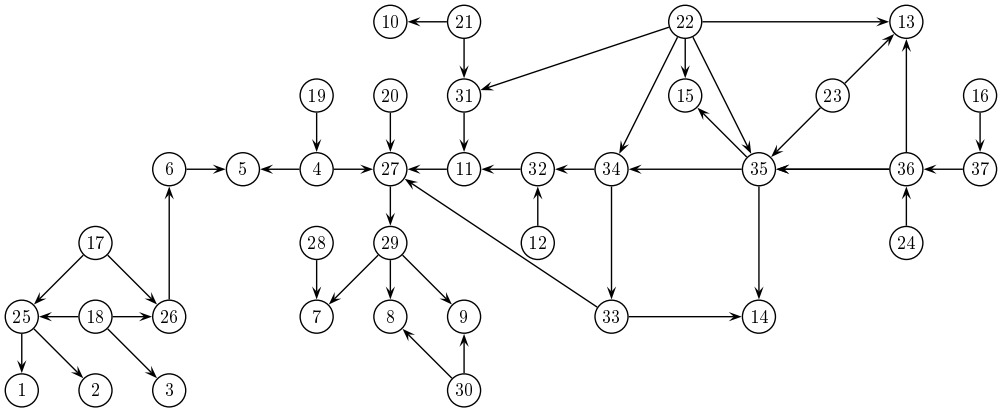


Figure 4.9. Average learning times of the KES algorithm applied to data sampled from the Alarm model for 11 different settings of k.



1	central venous pressure
2	pulmonary capillary wedge pressure
3	history of left ventricular failure
4	total peripheral resistance
5	blood pressure
6	cardiac output
7	heart rate obtained from blood pressure monitor
8	heart rate obtained from electrocardiogram
9	heart rate obtained from oximeter
10	pulmonary artery pressure
11	arterial-blood oxygen saturation
12	fraction of oxygen in inspired gas
13	ventilation pressure
14	carbon-dioxide content of expired gas
15	minute volume, measured
16	minute volume, calculated
17	hypovolemia
18	left-ventricular failure
19	anaphylaxis

20	insufficient anesthesia or analgesia
21	pulmonary embolus
22	intubation status
23	kinked ventilation tube
24	disconnected ventilation tube
25	left-ventricular end-diastolic volume
26	stroke volume
27	catecholamine level
28	error in heart rate reading due to low cardiac output
29	true heart rate
30	error in heart rate reading due to electrocautery device
31	shunt
32	pulmonary-artery oxygen saturation
33	arterial carbon-dioxide content
34	alveolar ventilation
35	pulmonary ventilation
36	ventilation measured at endotracheal tube
37	minute ventilation measured at the ventilator

Figure 4.10. The DAG structure of the Alarm BN model and the table of labels for each node. The effective size of the Alarm model is 771.

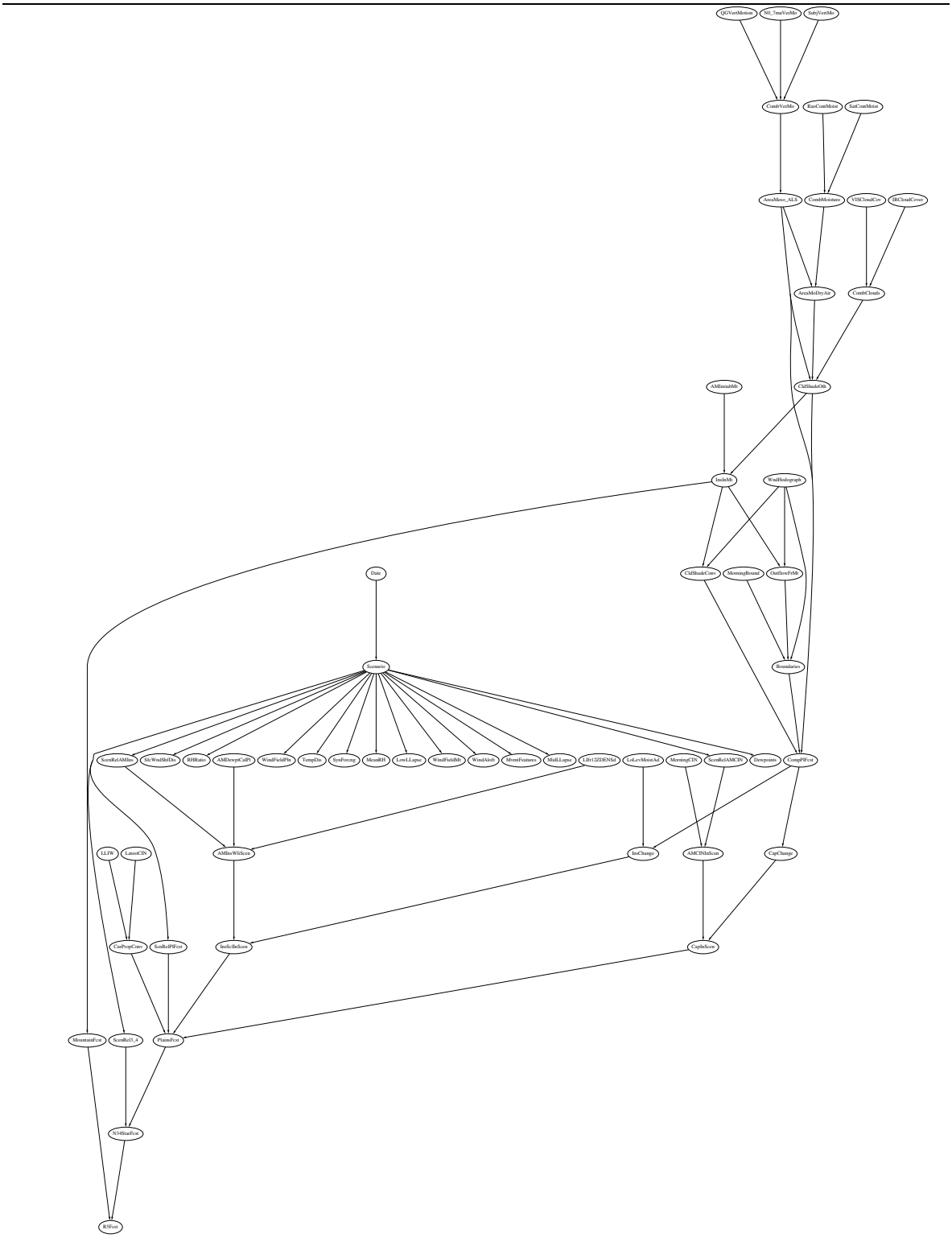


Figure 4.11. The Hailfinder network for severe weather forecasting, developed by Abramson et al. (1996). The effective size of the Hailfinder BN model is 9406.

weight of the tree is the accumulated mutual information between child-parent variable pairs in the tree, which corresponds to minimising KL-distance (4.1).

Later works, where the restriction on structure is relaxed to include general DAG structures, include the SGS algorithm (see Spirtes et al. (2000)). This algorithm performs statistical tests of conditional independence, and incrementally builds a DAG structure entailing *d-separation* properties corresponding to the conditional (in)dependencies that are verified from data. One problem with this approach is that it assumes a reliable way of testing conditional independence. Using a statistical hypothesis tests we are always running the risk of the test failing by chance while the hypothesis is in fact true. This problem becomes increasingly important when multiple such tests are needed, which is typically the case for the SGS algorithm. However, it can be proved that the SGS algorithm returns the optimal model if given a reliable test of conditional (in)dependence. Approaches following the general recipe of explicitly inducing a structure that entails correct (in)dependence relations is usually referred to as *constraint based search* approaches.

Apart from the already mentioned GES algorithm (Meek, 1997), another important early work on learning general BN models is the work by Cooper and Herskovits (1992). Here, the K2 procedure is proposed for recovering a BN structure by a heuristic search for finding the most probable structure. The K2 procedure requires an ordering of the variables as input and in addition an upper bound on the number of parents that a node may have. Cooper and Herskovits (1992) shows promising results by learning from datasets sampled from the Alarm network (Beinlich et al., 1989). The K2 procedure employs a greedy search for the best parents for each node, choosing the parent that increases a local score (based on likelihood of the model) the most without violating the ordering or the threshold for number of parents. Approaches to learning BN models that aims at optimising some score function are usually called *score based search* approaches.

The result of multiple restarts of `KES` was used by Peña et al. (2004) in assisting the user in the interpretation of a BN model learnt from data. Specifically, after a sequence of restarts of `KES`, a special graph can be constructed over the variables where each edge is annotated with a relative frequency of existence of the edge in the set of inclusion optimal models recovered in the sequence of restarts of `KES`.

It was shown by Cowell (2001) that *constrain based search* and *score based search* are identical approaches when learning BN models under the assumptions that: 1) an ordering of the variables is given; 2) data is complete; 3) the statistical test is based on cross entropy, and; 4) the score metric is based on maximising log-likelihood (possibly with some penalty). Under these assumptions, *score based* and *constraint based* approaches will have identical preference between models and should therefore only be viewed as different interpretations of the same approach.

The justification for using restarts of the `KES` algorithm was based on the fact that there may be exponentially many local optima and in such settings, restarts of the stochastic search procedure enables `KES` to investigate a larger area of the search space. Gomes and Selman (1997) investigates cost profiles of search procedures in a more general combinatorial problem. They show that when the cost profile is characterised by a heavy tailed distribution, the average

performance of a sequence of search procedures can be improved dramatically by introducing random restarts of the search. We can view the event that `KES` recovers a local optimal model and not the global optima as a heavy tail of the cost profile of `KES`, and therefore, the restarts can be seen as a similar way of exploiting the heavy-tailed behaviour as the random restarts used by Gomes and Selman (1997).

4.4 Learning Naïve Bayes Models

The problem of leaning a NB model from data reduces to the problem of learning the cardinality of the latent component variable C, a prior distribution over latent components $P(C)$ and marginal conditional distributions $P(X_i|C)$ for all variables X_i observed in data.

4.4.1 Estimating Parameters from Incomplete Data: The EM-Algorithm

As the latent component variable of the NB model is never observed in data, maximum likelihood estimation from Equation (4.14) is not possible, as we lack the possibility to count observations of C in $\mathcal{D}$. The standard approach to estimating parameters in the presence of incomplete data and latent variables is the Expectation-Maximisation (EM) algorithm (Dempster et al., 1977; McLachlan and Krishnan, 1997; Lauritzen, 1995). The EM algorithm alternates between two steps, the E-step and the M-step. The E-step amounts to computing expected counts for the missing observations, while the M-step uses these expected counts as if they were observed in the efficient computation of maximum likelihood parameters. By iterating over these two steps, the EM algorithm converges to a parameterisation that is defines a local maximum of the likelihood function. Assuming some initial (typically random) parameterisation θ_0 of NB model M, EM is then implemented by the two steps:

E-step: augment each instance $d \in \mathcal{D}$ by a vector of fractional counts for C of $P^M(C|\mathbf{X} = d)$, where M is the current NB model with parameters θ_n. In this way, we can construct the expected counts:

$$N_c^* = \sum_{d \in \mathcal{D}} P^M(C = c|\mathbf{X} = d), \tag{4.31}$$

$$N_{c,x_{i,h}}^* = N_{x_{i,h}} \cdot P^M(C = c, X_i = x_{i,h}). \tag{4.32}$$

M-step: construct parameters θ_{n+1} by ML estimation using expected counts as if they were actual observed counts. This amounts to updating conditional distributions $P^M(X_i|C)$ for every $X_i \in \mathbf{X}$ and prior $P^M(C)$ as:

$$P^M(C = c) = \frac{N_c^*}{N}, \tag{4.33}$$

$$P^M(X_i = x_{i,h}|C = c) = \frac{N_{c,x_{i,h}}^*}{N_c^*}. \tag{4.34}$$

Algorithm 4.5 Simple algorithm for learning a range of NB models from data.

Input: Fully observed data $\mathcal{D}$.

Output: Range of NB models of increasing effective size.

```
1: procedure LearnNB(D)
2:     initialise NB model M with k_min latent components
3:     repeat
4:         estimate parameters of M by EM
5:         output M
6:         prune low weight components of M
7:         split large components of M
8:         add k new components to M
9:     until stopping criteria is met
```

The EM algorithm iterates between these two steps until a termination criterion is met. Common termination criteria include convergence in parameters, and setting a threshold on the number of iterations allowed.

The EM algorithm is only guaranteed to converge to a local maximum, and there may be many such local maxima where only a small fraction are close to the global maximum. The common strategy used to mitigate the problem of poor EM estimates is to perform multiple restarts of EM with different random starting points.

4.4.2 Learning the Cardinality of the Latent Component Variable

We aim at learning NB models for approximation of a probability distribution and for performing belief updating inference task using the model. This aim is somewhat different from most previous applications of the NB model, as mentioned in Section 3.4. Typically, the learning of a NB model with latent component variable is aimed at discovering hidden structure among the variables or to attain a soft clustering of instances. In both cases, it is preferable to keep the cardinality of the latent variable from growing unbounded, as too many clusters can make it hard for users to use the clustering for understanding latent structure in the domain. However, for general probabilistic inference, bounding the cardinality of the latent variable is only relevant from the point of view of bounding the loss of efficiency.

Lowd and Domingos (2005) proposes the NBE algorithm for learning NB models for general probabilistic inference. In the NBE algorithm, the cardinality of the latent component variable is optimised by basically repeating the three steps: 1) increase the current cardinality, 2) estimate parameters by EM, and 3) prune low weight components.[9] Low weight components are components with relatively low prior probability. As a termination criterion, the NBE algorithm uses a separate hold-out dataset and measures likelihood over this dataset. The failure to improve likelihood then makes the algorithm terminate returning the model of max likelihood over the hold-out dataset. We adopt the NBE algorithm of Lowd and Domingos

[9] In our application of the EM algorithm we do not employ any heuristic in order to escape local optima, such as random restarts.

(2005) with minor modifications. To obtain a range of NB models of different efficiency and accuracy rather than a single model, we do not need a holdout dataset to decide on termination. Instead we will continue to increase the cardinality to get more and more complex models. Algorithm 4.5 gives a high-level description of our `LearnNB` procedure.

In `LearnNB`, when the cardinality has been increased from m to k, we initialise the prior of each of the $k-m$ new components to $\frac{1}{k}$. While we have no theoretical justification to choose exactly $\frac{1}{k}$ as the initial prior of new components instead of any other initialisation, it seems at least reasonable to choose a uniform prior for all new components. Also, subsequently, the EM procedure will be applied to estimate better parameters. To ensure that $P(C)$ remains normalised, the m old priors are scaled by $\frac{m}{k}$. For all variables $\mathbf{X}$, probabilities $P(X_i|C=c_{new})$ is initialised by a randomly drawn instance $d \in \mathcal{D}$ as follows:

$$P(X_i = x_{i,h}|C = c_{new}) := \begin{cases} \frac{1+0.1\cdot P'(x_{i,h})}{1.1} & \text{if } x_{i,h} = d[X_i], \\ \frac{0.1\cdot P'(x_{i,h})}{1.1} & \text{otherwise,} \end{cases} \tag{4.35}$$

where $P'(x_{i,h}) = \frac{N_{x_{i,h}}}{|\mathcal{D}|}$. This way of initialising new components is intuitive if we view the learning of an NB model as the process of discovering unlabelled natural groups within the dataset, i.e., latent clusters. By initialising a new component by a randomly selected data instance d, we then initialise a new latent cluster with centre at d.

The pruning of low weight components (line 6 of Algorithm 4.5) imposes an implicit upper bound on the cardinality of the latent component variable, and thereby on the complexity of the NB model, in the following way: all components c' with a prior $P(C=c') \leq \frac{1}{w}$ are removed from the model, for some integer w. This automatically yields a maximum cardinality $|R(C)|$ of w, and in our implementation we use $w = 1000$.

All aspects of the `LearnNB` procedure introduced so far are adopted directly from the NBE algorithm of Lowd and Domingos (2005). One new addition to the proposal of Lowd and Domingos is the introduction of component splitting. If we view the learning of an NB model as the process of discovering clusters within the data, the splitting of components (or clusters) is the substitution of one existing cluster for two new clusters. This makes sense when a single component captures two (or more) clusters. To select components that captures more than one component, it would be natural to select components $c \in R(C)$ for which the joint conditional distribution $P(\mathbf{X}|C=c)$ is inhomogeneous, that is, low entropy. The exact computations of the conditional entropy $H(\mathbf{X}|C=c) = \sum_{\mathbf{x}\in R(\mathbf{X})} P(\mathbf{X}=\mathbf{x}|C=c)\log P(\mathbf{X}=\mathbf{x}|C=c)$ requires a sum of $|R(\mathbf{X})|$ terms. A more efficient approach would be to approximate the distribution $P(\mathbf{X}|C=c)$ by simulation or sampling techniques (see (Neal, 1993) or (Castillo et al., 1997, Section 9.3)). However, we use a much more simple heuristic for choosing components for splitting which simply chooses a component c if the prior $P(C=c)$ has captured the majority of the total probability mass. The reasoning behind choosing components of high prior is that splitting such components has the largest potential for increasing the overall accuracy. In our implementation, we will split component c when $P(C=c) \geq 0.9$. The splitting of components (line 7 of Algorithm 4.5) is performed for large component c_l by replacing c_l with 2 new components c_l' and c_l'', each with prior $P(C=c_l') = \frac{1}{2}P(C=c_l)$, where $P(C=c_l)$ is the prior of the component c_l before splitting. Each

conditional $P(X_i|C = c'_l)$ and $P(X_i|C = c''_l)$ is initialised as a copy of the old $P(X_i|C = c_l)$ exposed to a random perturbation.

4.4.3 Related Work

Learning the cardinality of the latent component variable of a NB model is a problem that has received considerable attention. One direct approach would be to perform an exhaustive search over a range of possible cardinalities, choosing the one that results in a NB model that attains maximal score (Cheeseman and Stutz, 1996). However, to score each model, parameters needs to be estimated by EM which may be too time consuming considering also that multiple restarts of EM for each cardinality may be required.

Elidan and Friedman (2001) proposes an approach to learning the cardinality of hidden variables in BN models that avoids expensive EM algorithm in the search for a good cardinality. They work with hard assignments of instances in the data to each latent state of the hidden variable, that is, each instance in the data is at any point in time associated with a single latent state of the hidden variable. Initially, the hidden variable has a relatively large cardinality and in each iteration, states are merged to reduce the cardinality and the model is scored using the current hard assignment of instances. Eventually, the cardinality can not be reduced further and the best cardinality encountered during the search is returned.

The operation of splitting components in learning the cardinality of C in a NB model has previously been proposed (see eg. Karčiauskas (2005) or Elidan (2004)). The heuristic for choosing components for splitting used by Karčiauskas (2005) is an exhaustive search over all possible splits, choosing the one that yields the model of maximal score.

4.5 Learning Probabilistic Decision Graph Models

In this section we will address the problem of learning PDG models that optimise score function S_λ (see Eq. 4.8) for some λ. Major parts of the material presented in this section is based on ideas previously published in (Jaeger et al., 2004). It was established in Section 4.2, that the problem of estimating ML parameters from complete data with no latent variables is computable by taking fractions of counts. By (3.28), the likelihood function for PDG model M over variables $\mathbf{X}$ is :

$$l(\mathcal{D}|M) = \prod_{d\in\mathcal{D}} \prod_{X_i\in\mathbf{X}} p_{d[X_i]}^{reach(i,d)}. \tag{4.36}$$

Then, the log likelihood of data $\mathcal{D}$ given PDG model M over variables $\mathbf{X}$ is:

$$\begin{aligned} L(\mathcal{D}|M) &= \sum_{d=\mathcal{D}} \sum_{X_i\in\mathbf{X}} \log p_{d[X_i]}^{reach(i,d)} \\ &= \sum_{X_i\in\mathbf{X}} \sum_{h=0}^{k_i} \sum_{\nu_i\in V_i} N_h^{\nu_i} \log p_h^{\nu_i}, \end{aligned} \tag{4.37}$$

where $N_h^{\nu_i}$ is the number of instances $d \in \mathcal{D}$ reaching $\nu_i \in V_i$ for which $d[X_i] = x_{i,h}$, and $k_i = |R(X_i)|$. For a given PDG structure G, the ML estimate $\hat{\mathbf{p}}^{\nu_i}$ for parameters attached to parameter node $\nu_i \in V_i$ is:

$$\hat{p}_h^{\nu_i} = \frac{N_h^{\nu_i}}{N^{\nu_i}}, \tag{4.38}$$

where $N^{\nu_i} = \sum_{h=0}^{k_i} N_h^{\nu_i}$. We can then express (4.37) as:

$$L(\mathcal{D}|M) = \sum_{X_i \in \mathbf{X}} \sum_{h=0}^{k_i} \sum_{\nu_i \in V_i} N_h^{\nu_i} \log \frac{N_h^{\nu_i}}{N^{\nu_i}}, \tag{4.39}$$

Then the general penalised log-likelihood score S_λ for PDG model M become:

$$\begin{aligned} S_\lambda(\mathcal{D}, M) &= (1-\lambda) L(\mathcal{D}|M) - \lambda size_{\mathit{eff}}(M) \\ &= (1-\lambda) \sum_{X_i \in \mathbf{X}} \sum_{h=0}^{k_i} \sum_{\nu_i \in V_i} N_h^{\nu_i} \log \frac{N_h^{\nu_i}}{N^{\nu_i}} - \lambda \sum_{X_i \in \mathbf{X}} (\max(1, |ch_G(X_i)|) \cdot |V_i| \cdot k_i) \\ &= \sum_{X_i \in \mathbf{X}} \left((1-\lambda) \sum_{h=0}^{k_i} \sum_{\nu_i \in V_i} N_h^{\nu_i} \log \frac{N_h^{\nu_i}}{N^{\nu_i}} - \lambda \left(\max(1, |ch_G(X_i)|) \cdot |V_i| \cdot k_i\right) \right) \end{aligned} \tag{4.40}$$

For the rest of this section, we will focus on the search for a structure G that optimises (4.40). For a given domain $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ there exists $n!$ distinct orderings of the elements, so $n!$ is a (conservative) lower bound on the number of distinct forest structures. For each forest structure, the number of distrinct PDG structures is at least exponential in the number of variables contained in the tree of maximal depth in the forest. The cardinality of the search space therefore makes exhaustive structure search intractable, and we will resort to heuristic procedures for learning structures.

4.5.1 Structural Learning in PDGs

We will divide the search for good PDG structures into two conceptually disjoint tasks:

1. learning a good forest structure over the variables, and
2. learning a PDG structure w.r.t. that forest.

This decomposition is motivated by the following points:

- Conceptually, this decomposition is natural, while in practice they are not completely independent components of the learning task.
- Considering efficiency of the learning procedure, fixing a variable forest structure F effectively reduces the space of possible PDG structures to be considered by the learning procedure.

Algorithm 4.6 The procedure `LearnPDGs` that learns a set of PDG models from a fully observed dataset $\mathcal{D}$. The two conceptually distinct phases are implemented by the `LearnForest` procedure of Algorithm 4.12 and the `LearnPDG` of Algorithm 4.7.

Input: $\mathcal{D}$: fully observed dataset; $\mathbf{\Lambda}$: list of values from $[0, 1]$; $\mathbf{T}$: list of values from $[0, 1]$. $\lambda \in \mathbf{\Lambda}$.

1: **function** `LearnPDGs`$(\mathcal{D}, \mathbf{\Lambda}, \mathbf{T})$
2: $\quad \mathbf{F} := \emptyset$ ▷ Population of forest structures
3: $\quad \mathbf{G} := \emptyset$ ▷ Population of PDGs
4: $\quad$ **for all** $t \in \mathbf{T}$ **do** ▷ Phase I
5: $\quad\quad \mathbf{F} := \mathbf{F} \cup \{$`LearnForest`$(\mathcal{D}, t)\}$
6: $\quad$ **for** λ from $\lambda_{max} \ldots \lambda_{min}$ in $\mathbf{\Lambda}$ **do** ▷ Phase II
7: $\quad\quad$ **for all** $F \in \mathbf{F}$ **do**
8: $\quad\quad\quad \mathbf{G} := \mathbf{G} \cup \{$`LearnPDG`$(F, \lambda, \mathcal{D})\}$
9: $\quad\quad$ output $\underset{G \in \mathbf{G}}{argmax}\, S_\lambda(\mathcal{D}, G)$
10: $\quad\quad$ prune low forests yielding low scoring PDGs from $\mathbf{F}$
11: $\quad\quad \mathbf{G} := \emptyset$

For the reasons mentioned above, we decompose our structural learning algorithm into two phases. In Phase I, a variable forest is induced from data. By performing suitable statistical tests of conditional independence relations, we build a tree structure that only entails independencies that were verified through the test. In the second phase (Phase II) we then optimise a PDG structure w.r.t. the variable forest from the Phase I, for the score function of (4.40).

Algorithm 4.6 contains a pseudo-code description of the top-level learning procedure `LearnPDGs`. The two phases are implemented in lines 4-5 and 6-11 respectively. We incorporate a population based search for good forest structures. That is, a population $\mathbf{F}$ of variable forests is constructed in Phase I. Next, this population is pruned by removing forest structures for which we fail to build high-scoring PDG structures in Phase II.

We will postpone the detailed description of Phase I, and in the following assume that a PDG forest have already been constructed.

Optimising the PDG-structure: Phase II

Algorithm 4.7 describes the `LearnPDG` procedure. The `LearnPDG` procedure optimises a PDG structure w.r.t. variable forest F for (4.40).

Initially a minimal PDG structure is build, and this structure is then repeatedly exposed to a sequence of local score optimising structural transformations, until the score converges.

In the following, we describe the procedures `splitNodes`, `mergeNodes` and `redirectEdges`, that implements local operations for score optimisation.

Splitting nodes The `splitNodes` procedure introduces new parameters by replacing existing parameter nodes with a set of new parameter nodes. The structural transformation of

Algorithm 4.7 This procedure searches for a optimal (w.r.t. (4.40)) PDG structure w.r.t. a variable forest F. The local procedures `splitNodes` (Alg. 4.8), `mergeNodes` (Alg. 4.9) and `redirectEdges` (Alg. 4.10) are used to optimise the score function.

```
1: procedure LearnPDG(F, λ, D)
2:     G := minimal PDG for F
3:     repeat
4:         for all trees T of F do
5:             X_r := root of T
6:             splitNodes(V_r, λ, D)
7:             mergeNodes(V_r, λ, D)
8:             redirectEdges(V_r, λ, D)
9:     until S_λ(D, G) did not improve
10:    return G
```

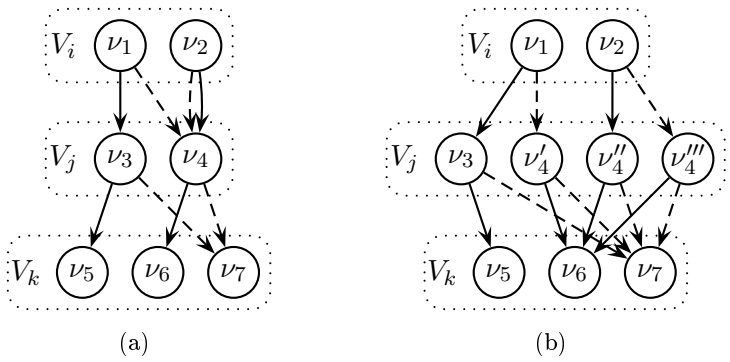


Figure 4.12. The structural modification performed when splitting node ν_4 by the `split` operation is shown. (a) shows the local structure before the split, and (b) shows the resulting structure. Only the relevant section of the PDG shown.

splitting a parameter-node by the `split` operation can be seen in Figure 4.12.

When splitting a parameter-node ν_i having an in-degree of n, we replace ν_i with n new parameter-nodes, one for each incoming edge. The set of children of ν_i are copied to each of the n new parameter-nodes. Parameters for new nodes needs to be re-estimated, while counts for no other nodes in the PDG will change as a result of the `split` operation. Denote by $new(\nu_j, l, \nu_i)$ the node that would be created for edge $\nu_j \xrightarrow{l} \nu_i$ when splitting ν_i, and let $inc(\nu)$ be the set of edges incoming to ν. We can then express the score gain associated with splitting

$\nu_i \in V_i$ in PDG structure G as:

$$\begin{aligned} S_\lambda(\mathcal{D}, M_2) - S_\lambda(\mathcal{D}, M_1) =& (1-\lambda)[L(\mathcal{D}|M_2) - L(\mathcal{D}|M_1)] - \lambda[size_{\mathit{eff}}(M_2) - size_{\mathit{eff}}(M_1)] \\ =& (1-\lambda) \left[\sum_{\substack{l \\ \nu_j \rightarrow \nu_i \\ \in inc(\nu_i)}} \left(\sum_{h=0}^{k_i} N_{lh}^{\nu_j} \log \hat{p}_h^{new(\nu_j,l,\nu_i)} \right) - \sum_{h=0}^{k_i} N_h^{\nu_i} \log p_h^{\nu_i} \right] \\ & - \lambda[(|pa_G(\nu_i)| - 1) \cdot \max(1, |ch_G(X_i)|) \cdot k_i] \end{aligned} \tag{4.41}$$

where M_1 is the PDG model before splitting ν_i and M_2 is the model after the split. $\hat{p}^{new(\nu_j,l,\nu_i)}$ in (4.41) is the ML estimates for $p^{new(\nu_j,l,\nu_i)}$:

$$\hat{p}_h^{new(\nu_j,l,\nu_i)} = \frac{N_{lh}^{\nu_j}}{N_l^{\nu_j}}, \tag{4.42}$$

where $N_{lh}^{\nu_j}$ is the number of instances $d \in \mathcal{D}$ reaching ν_j for which $d[X_j] = x_{j,l}$ and $d[X_i] = x_{i,h}$.

If we assume ML parameters, we can recover the counts for data instances reaching node ν_i by $N^{\nu_i} = |\mathcal{D}| \cdot \mathit{ifl}(\nu_i)$. Then counts N_h^ν can also easily be recovered from ML parameter p_h^ν by (4.38). However, $N_{lh}^{\nu_i}$ is not easily reconstructed without accessing the data.

To avoid data access needed to extract count $N_{lh}^{\nu_j}$ in (4.42) necessary for computing the exact score gain through (4.41), we will instead focus on a heuristic score for selection of nodes.

Let $\nu_i \in V_i$ and $pa_F(X_i) = X_j$. The potential for positive contribution to the score by splitting ν_i very much depends on the number of data instances reaching ν_i. Denote by $\gamma(e)$ the probability mass flowing into ν_i via edge e, that is:

$$\gamma(\nu_j \xrightarrow{h} \nu_i) = \mathit{ifl}(\nu_j) \cdot p_h^{\nu_j}. \tag{4.43}$$

The relative distribution of contributions to the *inflow* over incoming edges is also important to the potential of splitting a node. Even for a relatively high $\mathit{ifl}(\nu_i)$, if most of the probability mass flows into ν_i via a single edge, the possible accuracy gain will be low, as a split would basically produce one node identical to ν_i and a number of "low income" nodes that, therefore, can not impact the total accuracy significantly. For this reason, we prefer nodes for which the distribution of incoming probability mass $\{\gamma(e) : e \in inc(\nu_i)\}$ is less peaked and, hence, has high entropy.

From the above discussion, we arrive at the heuristic score given in equation (4.44):

$$\texttt{splitPotential}(\nu_\texttt{i}) = \mathit{ifl}(\nu_i) \cdot \frac{H(\{\frac{\gamma(e)}{\mathit{ifl}(\nu_i)} : e \in inc(\nu_i)\})}{\log(|inc(\nu_i)|)}, \tag{4.44}$$

where $H(\cdot)$ is the entropy function, and $\log(|inc(\nu_i)|)^{-1}$ then normalise $H(\cdot)$.

Algorithm 4.8 describes the `splitNodes` procedure that selects nodes for splitting using the `splitPotential` measure of (4.44). The `splitNodes` procedure may cause the PDG to be fully expanded by splitting all nodes top down. To avoid this, in our implementation we simply disallow the splitting of nodes that have one or more parents that was split in the current traversal of the structure.

Algorithm 4.8 This procedure randomly selects node for splitting by the `split` operation, biasing the selection towards nodes with relatively high `splitPotential`. The aggressiveness of the selection is controlled by the λ value, the larger the λ, the more aggressive the selection will be. The `split` operation of line 6 performs the structural modification of the split (see Fig. 4.12).

```
1: procedure splitNodes(V_i, λ)
2:     if pa_F(X_i) ≠ ∅ then
3:         for all ν ∈ V_i do
4:             rnd := random number from [0, 1)
5:             if (1 − λ) · splitPotential(ν) > rnd then          ▷ See (4.44).
6:                 split(ν)
7:     for all X_j ∈ ch_F(X_i) do                                 ▷ The top-down traversal
8:         splitNodes(V_j, λ)
```

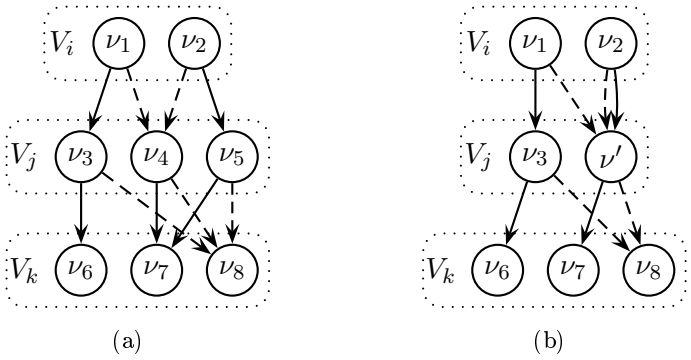


Figure 4.13. The structural modification performed when merging nodes ν_4 and ν_5 by the `merge` operation is shown. (a) shows the local structure before the merge, and (b) shows the resulting structure. Only the relevant section of the PDG shown.

Merging Nodes Redundant parameters that do not contribute significantly to the accuracy of the model but only contributes to the size-penalty should be removed from the model. The `merge` procedure obtains this by merging parameter-nodes. Figure 4.13(a)-(b) shows the structural modification of merging nodes ν_4 and ν_5. In Figure 4.13(a), ν_4 and ν_5 have identical children, and this removes the problem of deciding which child to keep, had the children not been identical $succ(\nu_4, X_k, h) \neq succ(\nu_5, X_k, h)$. We will require of two nodes being considered for merging that they have identical children. Then the score gain of merging two nodes $\nu_{i_1}, \nu_{i_2} \in V_i$ can be computed as:

$$S_\lambda(\mathcal{D}, M_2) - S_\lambda(\mathcal{D}, M_1) = \\ (\lambda - 1)[L(\mathcal{D}|M_2) - L(\mathcal{D}|M_1)] - \lambda[size_{\mathit{eff}}(M_2) - size_{\mathit{eff}}(M_1)], \quad (4.45)$$

where M_1 is the PDG model before the merge and M_2 is the model after the merge. It is clear that the effective size is reduced by $max(1, |pa_F(X_i)|) \cdot |R(X_i)|$ when merging two nodes $\nu_{i_1}, \nu_{i_2} \in V_i$. For computing the possible loss in accuracy, we need to compute the ML estimate $\hat{\mathbf{p}}^{\nu_{i_{1+2}}}$ for the node $\nu_{i_{1+2}}$ created by merging ν_{i_1} and ν_{i_2}. This estimate is:

$$\begin{aligned}\hat{p}_h^{\nu_{i_{1+2}}} &= \frac{N_h^{\nu_{i_1}} + N_h^{\nu_{i_2}}}{N^{\nu_{i_1}} + N^{\nu_{i_2}}} \\ &= \frac{\hat{p}_h^{\nu_{i_1}} \cdot ifl(\nu_{i_1}) + \hat{p}_h^{\nu_{i_2}} \cdot ifl(\nu_{i_2})}{\sum_{x_{i,l} \in R(X_i)} \hat{p}_l^{\nu_{i_1}} \cdot ifl(\nu_{i_1}) + \hat{p}_l^{\nu_{i_2}} \cdot ifl(\nu_{i_2})}.\end{aligned} \tag{4.46}$$

Please note that only existing values of *ifl* and ML estimates of the parameters $\mathbf{p}$ for the nodes ν_{i_1} and ν_{i_2} are used to compute (4.46), and no data access is necessary. The loss in accuracy can be expressed as:

$$\begin{aligned}L(\mathcal{D}|M_2) - L(\mathcal{D}|M_1) &= \sum_{x_{i,h} \in R(X_i)} \left((N_h^{\nu_{i_1}} \log p_h^{\nu_{i_1}} + N_h^{\nu_{i_2}} \log p_h^{\nu_{i_2}}) - (N_h^{\nu_{i_1}} + N_h^{\nu_{i_2}}) \log \hat{p}_h^{\nu_{i_{1+2}}} \right) \\ &= \sum_{x_{i,h} \in R(X_i)} \left(N_h^{\nu_{i_1}} (\log p_h^{\nu_{i_1}} - \log \hat{p}_h^{\nu_{i_{1+2}}}) + N_h^{\nu_{i_2}} (\log p_h^{\nu_{i_2}} - \log \hat{p}_h^{\nu_{i_{1+2}}}) \right) \\ &= \sum_{\nu \in \{\nu_{i_1}, \nu_{i_2}\}} N^{\nu} D_{KL}(\mathbf{p}^{\nu} || \hat{\mathbf{p}}^{\nu_{i_{1+2}}}),\end{aligned} \tag{4.47}$$

where Equality 4.47 assumes ML parameters $\mathbf{p}^{\nu_1}$ and $\mathbf{p}^{\nu_2}$. By (4.46) these are obtainable without accessing data. As $N^{\nu} = ifl(\nu) \cdot N$, for comparing (4.47) for different pairs of nodes, we can use the inflows of the nodes involved. We then arrive at the general score `mergeScore` of (4.48):

$$\texttt{mergeScore}(\nu_{i_1}, \nu_{i_2}) = \sum_{\nu \in \{\nu_{i_1}, \nu_{i_2}\}} ifl(\nu) \cdot D_{KL}(\mathbf{p}^{\nu} || \hat{\mathbf{p}}^{\nu_{i_{1+2}}}). \tag{4.48}$$

It is clear that (4.48) is computable without accessing data. Algorithm 4.9 shows the bottom-up merging of nodes. Nodes are selected for merge based on a λ-weighted sum of the `mergeScore` and number of parameters that will be removed from the model.

Redirecting Edges The local structural transformation of redirection of edge $\nu_j \rightarrow \nu_i$ assigns a new head ν_i' for the edge. We will need the following notation: For data $\mathcal{D}$ and PDG model M over variables observed in $\mathcal{D}$, we will denote by $\mathcal{D}^{\nu_i}$ (where $\nu_i \in V_i$) the subset of data instances $\{d \in \mathcal{D} : reach(i, d) = \nu_i\}$, i.e., the part of $\mathcal{D}$ that reaches ν_i. Maintaining $\mathcal{D}^{\nu_i}$ for all nodes is possible for limited sized $\mathcal{D}$. Each parameter-node ν_i can efficiently represent $\mathcal{D}^{\nu_i}$ by a list of pointers to instances in a static version of $\mathcal{D}$. For each variable $X_i \in \mathbf{X}$ every instance $d \in \mathcal{D}$ reaches a unique node, so in total we will need to store $|\mathcal{D}||\mathbf{X}|$ pointers in addition to the static data $\mathcal{D}$. The number of pointers is then invariant to the structure of the PDG model, and the storage requirement is therefore static for a given database $\mathcal{D}$. In addition, we will by $\mathcal{D}_h^{\nu_i}$ denote the set $\{d \in \mathcal{D}_h^{\nu_i} : d[X_i] = x_{i,h}\}$.

Returning to the redirection of edges, let $\nu_j \in V_j$, $\nu_i \in V_i$ and $pa_F(X_i) = X_j$. Recall that every parameter-node defines a marginal distribution over descendant variables in the variable

Algorithm 4.9 The `mergeNodes` procedure merges parameter-nodes by the `merge` operation (see Fig. 4.13) in a top down traversal of a PDG structure.

Input: V_i : set of parameter-nodes representing X_i in a PDG structure G w.r.t. variable forest F; λ : value from $[0, 1]$

Output: Valid PDG structure

procedure `mergeNodes`(V_i, λ)
  for all j such that $X_j \in ch_G(X_i)$ **do**
    `mergeNodes`$(V_j,\ \lambda)$
  for all $\{\nu_{i_1}, \nu_{i_2}\} \in V_i$ s.t. $\nu_{i_1} \neq \nu_{i_2}$ **do**
    if ν_{i_1} and ν_{i_2} have the same children **then**
      if then$(1-\lambda) \cdot$ `mergescore`$(\nu_{i_1}, \nu_{i_2}) < \lambda \cdot k_i \cdot \max(1, |pa_F(X_i)|)$
        `merge`(ν_{i_1}, ν_{i_2})

Algorithm 4.10 The `redirectEdges` procedure performs fine grained optimisation on a PDG structure, by redirecting edges in optimising (4.41).

Input: V_i : set of nodes; λ : value from $[0, 1]$.

procedure `redirectEdges`(V_i, λ)
  for all $X_j \in ch_G(X_i)$ **do**
    `redirectEdges`$(V_j,\ \lambda)$
  for all $\nu_i \in V_i$ **do**
    for all $x_{i,h} \in R(X_i)$ **do**
      for all $X_j \in ch_G(X_i)$ **do**
        $\nu_j := succ(\nu_i, X_j, x_{i,h})$
        $\nu_j^* := \underset{\nu \in V_j \setminus \nu_j}{argmax}(L(\mathcal{D}_h^{\nu}|f_G^{\nu}))$
        if $L(\mathcal{D}_h^{\nu_i}|f_G^{\nu_j^*}) > LL(\mathcal{D}_h^{\nu_i}|f_G^{\nu_j})$ **then**
          redirect $\nu_i \xrightarrow{h} \nu_j$ to new head node ν_j^*
  Remove any orphan nodes

forest defined by the recursive function $f_G^{\nu_i}$ (see Def. 3.19). Therefore, when selecting a new head node $\nu_i' \in V_i$ for edge $\nu_j \xrightarrow{h} \nu_i$, we prefer a node $\nu_i' \in V_i$ for which data $\mathcal{D}_h^{\nu_j}$ is more likely under $f_G^{\nu_i'}$ than under $f_G^{\nu_i}$.

The log-likelihood of $\mathcal{D}_h^{\nu_i}$ under $f_G^{\nu_i}$ is:

$$L(\mathcal{D}_h^{\nu_i}|f_G^{\nu_i}) = \sum_{d \in \mathcal{D}_h^{\nu_i}} \log f_G^{\nu_i}(d[de_G^*(X_i)]). \tag{4.49}$$

Algorithm 4.10 shows the `redirectEdges` procedure which performs redirections bottom-up in a PDG-structure, maximising (4.49).

The structural transformation of the redirection operator can result in some nodes being orphaned. As a result, a set of parameter-nodes (potentially more nodes than the orphan nodes) may become unreachable by any directed path from the root parameter-node. After

all redirections have been performed, we remove such nodes from the PDG-structure.

Complexity Let M be a PDG model of structure G w.r.t. variable forest F over variables $\mathbf{X}$. The `splitNodes` procedure (Alg. 4.8) computes `splitPotential` by eq. (4.44) for every parameter-node in M with more than one parent. The complexity of computing (4.44) for node ν is linear in the number of incoming edges $O(|inc(\nu)|)$. In general, $|inc(\nu)|$ can be exponential in the $|\mathbf{X}| - 1$ when F contains a single linear tree and the sets of parameter-nodes are maximal for all but the leaf variable that contains a single node. As explained earlier, we do not consider node ν for splitting if a parent of ν has already been split in the same traversal. Also, in-between consecutive invocations of the `splitNodes` procedure, we merge nodes through the `mergeNodes` procedure (Alg. 4.9), which further reduces the risk of experiencing exponential blowup. The complexity in practice is therefore expected to be sub-exponential, and indeed the `splitNodes` procedure exhibits tractable execution times in practise.

In the `mergeNodes` procedure (Alg. 4.9), we compute the `mergeScore` (eq. (4.48)) for every pair of parameter nodes $\{\nu_{i_1}, \nu_{i_2}\}$ in each node set V_i. Therefore, the complexity is quadratic in the largest set V_i of parameter-nodes $O(|V_i|^2)$. This size can again in theory be exponential in the number of variables, given suitable sequences of splits. However, as explained above, the aggressiveness of the `splitNodes` procedure is efficiently suppressed, making the procedure tractable in practice.

For the `redirectEdges` procedure (Alg. 4.10), for every edge $\nu_i \xrightarrow{h} \nu_j$ where $\nu_j \in V_j$ and $\nu_i \in V_i$, the marginal likelihoods are computed through (4.49) for every node $\nu'_j \in v_j \setminus \{\nu_j\}$. In general, this yields quadratic complexity in the largest set V_i, i.e., $O(|V_i|^2)$. By arguments similar to those above, we expect that even though $|V_i|$ can be exponential in the number of variables, in practice the size of $|V_i|$ is sub-exponential. Computing (4.49), however, is not free. Rather, it is an expensive procedure, as it includes accessing the data $\mathcal{D}^{\nu_i}$. For this reason, in our implementation of the `LearnPDG` procedure (Alg. 4.7), we invoke the `redirectEdges` procedure less often than the `splitNodes` and `mergeNodes` procedures.

Inducing the variable forest: Phase I

The type of conditional independence relation that are encoded in a PDG model D w.r.t. a variable forest F, are based on partitions of the state-space defined by sets of parameter-nodes V_i:

$$\begin{aligned} &P^D(X_i|\mathbf{X} \setminus de^*_F(X_i)) = P^D(X_i|pa^*_F(X_i)) = P^D(X_i|\mathscr{A}(V_i)) \\ \Rightarrow &X_i \perp\!\!\!\perp pa^*_F(X_i)|\mathscr{A}(V_i)[P^G] \end{aligned} \tag{4.50}$$

On the variable level, the partition $\mathscr{A}(V_i)$ is defined by the value of $pa_F(X_i)$, and the only conditional independence that are identifiable from the variable forest without inspecting the PDG structure are $X_i \perp\!\!\!\perp \mathbf{X} \setminus \{pa^*_F(X_i) \cup de^*_F(X_i)\}|pa^*_F(X_i)$. Variables that are members of different trees in the variable forest F will be marginally independent in any distribution represented by a PDG model w.r.t. forest F. Therefore, when learning the variable-forest, we wish to organise variables as follows:

1. Marginally independent variables are assigned to different trees, and marginally dependent variables to the same tree.

2. Within trees, the structure will branch at variable X_k such that for all pairs $\{X_i, X_j\} \subseteq ch_F(X_k)$ it is the case that $X_i \perp\!\!\!\perp X_j | \{pa^*_F(X_k) \cup X_k\}$.

On Testing for Conditional Independence To decide on marginal and conditional independence relations amongst the variables we use a χ^2-test of independence (DeGroot, 1986). We will construct the X^2 (or Pearson) statistic for the test. The X^2 statistics is:

$$X^2 = \sum_{B \in \mathscr{B}} \sum_{h=1}^{k_i} \sum_{l=1}^{k_j} \frac{(N^B_{hl} - E[N^B_{hl}])^2}{E[N^B_{hl}]}, \tag{4.51}$$

where $\mathscr{B}$ is the conditioning partitioning, N^B_{hl} is the observed count of instances $d \in \mathcal{D}$ where $d[X_i, X_j] = (x_{i,h}, x_{j,l})$ and $d \in B$, and $E[N^B_{hl}]$ is the *expected* count N^B_{hl} under the assumption that $X_i \perp\!\!\!\perp X_j | \mathscr{B}$ is true. This expectation is then computed as:

$$E[N^B_{hl}] = |\mathcal{D}| \frac{N^B_{h+} \cdot N^B_{+l}}{|\mathcal{D}^B|^2}, \tag{4.52}$$

where $N^B_{h+} = \sum_{l=1}^{k_j} N^B_{hl}$ and $N^B_{+l} = \sum_{h=1}^{k_i} N^B_{hl}$ and $\mathcal{D}^B = \{d \in \mathcal{D} : d \in B\}$. For marginal independence tests, the conditioning partitioning will be trivial partition $\mathscr{B} = \{\mathbf{\Omega}\}$.

When the tested independence holds true, then statistic X^2 will be χ^2 distributed with $|\mathscr{B}| \cdot (k_i - 1) \cdot (k_j - 1)$ degrees of freedom. The degrees of freedom is the number of free parameters that needs to be estimated, see Agresti (1990) (pages 174–175) for a discussion of the χ^2-test and degrees of freedom. We will reduce the degrees of freedom by one for each cell count of zero, which is a common approach (Spirtes et al., 2000).

As mentioned above, we wish to build a variable tree such that the tree branches at variable X_k and $X_i \perp\!\!\!\perp X_j | \{pa^*_F(X_k) \cup X_k\}$ for all pairs of children $\{X_i, X_j\}$ of X_k. The cardinality of the conditioning set $\{pa^*_F(X_k) \cup X_k\}$ is exponential in the size of the set. Therefore, it is very likely that data is too limited for us to perform reliable tests. However, the actual conditional independence relation encoded by the PDG structure is typically not based on the full $\mathscr{A}(\{pa^*_F(X_k) \cup X_k\})$ as conditioning partition, but rather a more coarse grained partition. That is, direct children of X_k will be independent in a PDG structure conditional on $\mathscr{I}(\mathscr{A}(V_k), \mathscr{A}(R(X_k)))$, which is typically not as fine grained as $\mathscr{A}(\{pa^*_F(X_k) \cup X_k\})$. We therefore, in addition to building the underlying variable trees, also induce a simple PDG structure. As will become apparent soon, we can do this by interleaving incremental building of variable trees through tests of independence, by an induction of a partial PDG structure over the variables currently included in the trees. We will then only need to estimate atmost as many parameters as the full partition generated by all predecessor variables, and in practise the number of parameters will be much smaller.

We need to have a strategy for handling situations where the amount of data is too limited to provide reliable estimates for the X^2 statistics of (4.51). For simplicity, we will only perform the test when we have more than 5 data instances (on average) per parameter for estimation

Algorithm 4.11 The `Grow` procedure grows a partially build PDG structure by increasing the depth by one more level. The `depGraph` function builds a dependency graph over variables by performing pairwise tests of conditional independence, using a χ^2 test and significance level t.

Input: T : partially build PDG structure; t : significance level from $[0, 1]$.

1: **procedure** `Grow`(T, t)
2: **for all** leaves V_i of T where $below(X_i) \neq \emptyset$ **do**
3: $\mathscr{B} := \mathscr{I}(\mathscr{A}(V_i), \mathscr{A}(X_i))$
4: $H :=$ `depGraph`$(below(X_i), \mathscr{B}, t)$
5: **for all** connected components $\mathbf{C}$ in H **do**
6: $X_j :=$ random variable from $\mathbf{C}$
7: $V_j := \{\nu_j\}$
8: $ch_F(X_i) := ch_F(X_i) \cup X_j$
9: $below(X_j) := \mathbf{C} \setminus X_j$

in computing the X^2 statistic. This is a commonly used rule-of-thumb (see eg. Spirtes et al. (2000) (pages 94–95)). When the cardinality of the conditioning partition becomes less than 5 instances we will assume the independence relation to be true without performing the test. Statistically, of course, this is an unjustified assumption, however, we will still use this heuristic to promote simpler models with fewer parameters and thereby the ability to obtain more reliable estimates for the parameters.[10,11]

Growing Variable Trees Algorithm 4.11 describes the `Grow` procedure, which is the central procedure in learning the variable forest. The `Grow` procedure extends the underlying variable tree of a partially build PDG-structure by adding another level of variables to the leafs of the tree. Each leaf X_l has an associated (possibly empty) set $below(X_l)$ of variables that are to be included in the subtree rooted at X_l. The `depGraph`$(\mathbf{Y}, \mathscr{B}, t)$ function returns a dependency graph over variables $\mathbf{Y}$ where $X_i, X_j \in \mathbf{Y}$ are connected if $X_i \not\perp\!\!\!\perp X_j | \mathscr{B}$ tests positive by a statistical test for conditional independence, using significance level t.

Figure 4.14 depicts an example of the structural transformations performed by the `Grow` procedure. Figure 4.14(a) depicts the initial situation. The partially build PDG structure already contains the variables X_6, X_2 and X_4, and variables $below(X_4) = \{X_1, X_3, X_7, X_5\}$ will the members of the subtree rooted at X_4. The next step, depicted in Figure 4.14(b), then builds a dependency graph over variables $below(X_4)$. The third and last step, depicted in Figure 4.14(c), then initialises a separate branch rooted at X_4 for each connected component in the dependency graph over variables $below(X_4)$. A branch is initialised by choosing a variable X_i at random as the root of the branch, and then placing the remaining variables from

[10] An alternative approach could be to use a score function instead of a statistical test to evaluate conditional independence when data is limited. Such approaches was investigated by Abellán et al. (2006).

[11] It should be mentioned that Fisz (1980) (pages 439–440) considers the necessary amount of data for the χ^2 test to give reliable results, and Fisz (1980) mentions the work of Vessereau (1958). Vessereau (1958) shows that when the expected frequencies are constant, one only needs a single data instance per parameter in (4.51). However, in our case, the expected frequencies are not necessarily constant as the partitions does not necessarily partition the data uniformly.

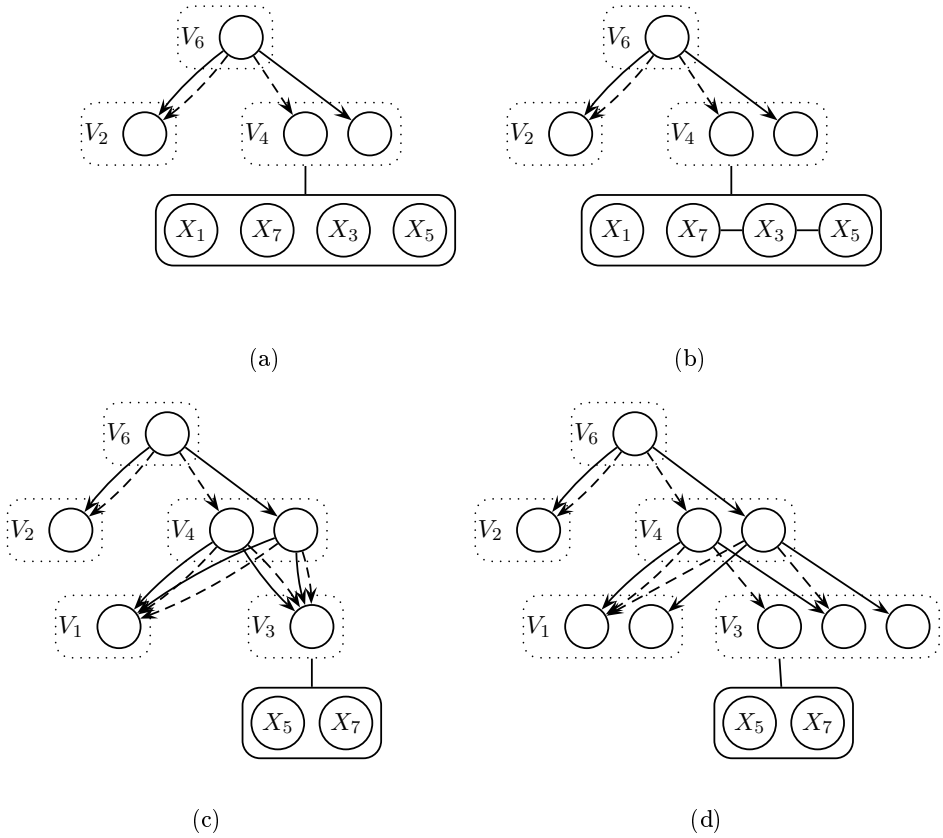


Figure 4.14. Snapshots of the procedure for growing variable forests. In this example, a tree is being build over 7 variables X_1 to X_7. The sets $below(\cdot)$ is indicated by the solid box attached underneath leafs.

the connected component $\{\mathbf{C} \setminus X_i\}$ in the set $below(X_i)$. Figure 4.14(d) depicts the partially build PDG after having been exposed to local structural transformations implemented in the `LearnPDG` procedure of Algorithm 4.7.

Building Variable Forests Algorithm 4.12 describes the procedure `LearnForest`. This procedure builds a full variable forest over variables $\mathbf{X}$ by first building a dependency graph over $\mathbf{X}$, using the trivial partitioning as conditioning partitioning, that is, marginal independence tests (line 4). Then, for each connected component in this dependency graph, we grow a tree using the `Grow` procedure described above (see Alg. 4.11).

In line 11 of Algorithm 4.12, trees are grown by alternating between the `Grow` procedure and the `LearnPDG` procedure that optimises the partially build PDG structure returned from `Grow`. A tree is fully grown when no leaf V_i has a non-empty $below(X_i)$ set.

Algorithm 4.12 The `LearnForest` procedure builds a variable forest by growing each tree through alternating between the `Grow` procedure and the `LearnPDG` procedure.

```
1: function LearnForest(D, t, λ_max)
2:     X := variables from D
3:     F := ∅
4:     H :=depGraph(X, {Ω}, t)
5:     for all connected components C in H do
6:         X_i :=rndVar(C)
7:         V_i := {ν_i}
8:         below(X_i) := C \ X_i
9:         T_i := tree w. V_i as root
10:        F := F ∪ {T_i}
11:        repeat
12:            Grow(T_i, t)
13:            LearnPDG(F, λ_max)
14:        until T_i is full-grown
       return F
```

4.5.2 Testing the PDG Learner

To perform initial quality checks of the PDG learning procedure of Algorithm 4.6, we experimented with several different databases consisting of iid samples from distributions represented by a PDG models. We performed two distinct experiments:

1. learning PDG structures with the correct variable forest given as a starting point, and

2. learning the PDG structure including the induction of a variable forest.

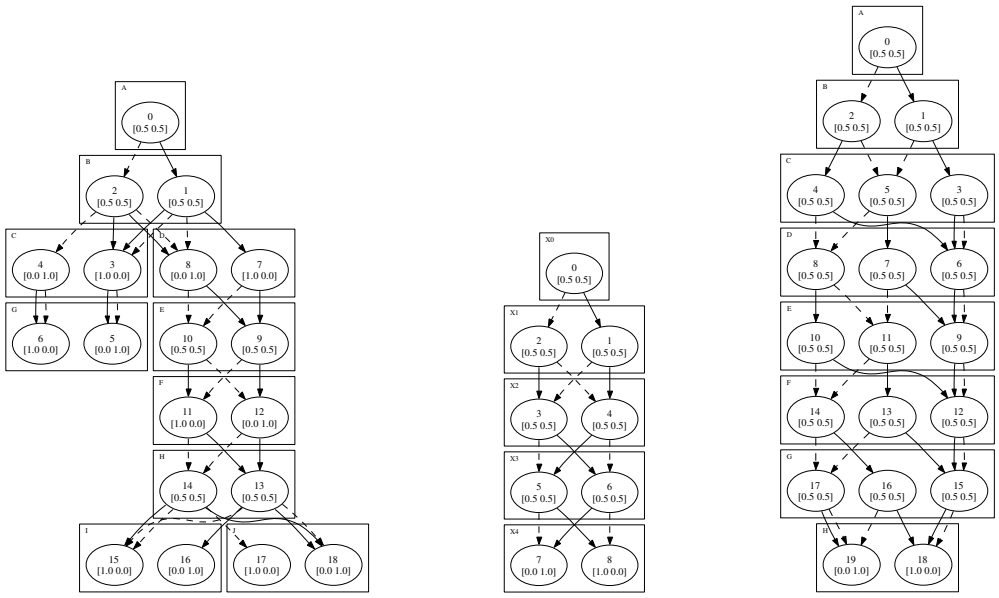
Clearly, the latter is both the harder and the more relevant test, the former was mainly performed as an initial sanity check of the `LearnPDG` procedure.

PDG sampled data

The merits of the PDG model is most clearly visible when representing logical relations as demonstrated by the parity distribution in Example 3.11. It is therefore natural to include manually constructed models that represents certain logical relationships.

We used 5 different PDG models, 3 of which were manually constructed (shown in Figure 4.15) and 2 randomly generated (shown in Figure 4.16). We sampled full instances from each model to get a fully observed dataset. This dataset was then partitioned into $\mathcal{D}_A$ and $\mathcal{D}_B$, where $|\mathcal{D}_A| = 10000$ and $|\mathcal{D}_B| = 5000$.

The 3 manually constructed PDG models (Logic1, Logic2 and Logic3) and the procedures for generating the 2 random PDG models (Rnd15 and Rnd20) are described below.



(a) (b) (c)

Figure 4.15. PDG structures: Logic1 (a) encodes a distribution containing the logical relationships listed in Table 4.2; Logic2 (b) encodes the parity distribution over 10 binary variables (see Example 3.11); Logic3 (c) encodes a relation where one variable assumes the value defined by the disjunction of pairwise conjunctions of the remaining variables (see Eq. (4.53)).

Variable	C	D	F	G	I	J
Truth-value	$A \vee B$	$A \wedge B$	$D \oplus E$	$\neg C$	$\neg(F \vee H)$	$\neg(F \wedge H)$

Table 4.2. Logical functions encoded in model Logic1. Variables A, B, E and H models input bits with a uniform $(\frac{1}{2}, \frac{1}{2})$ prior.

Logic1 This PDG model (depicted in Fig. 4.15(a)) represents a distribution over 9 binary random variables, 4 of which models input bits with a uniform $(\frac{1}{2}, \frac{1}{2})$ prior, while the others are determined by the logical relations listed in Table 4.2.

Logic2 This PDG model (depicted in Fig. 4.15(b)) encodes the parity distribution described in Example 3.11 over 5 binary variables $X_0, \ldots, X_4$.

Logic3 The last manually constructed PDG model (depicted in Fig. 4.15(c)) represents a distribution over the binary variables. Each variable, except a special variable H, has a uniform $(\frac{1}{2}, \frac{1}{2})$ prior, while H is determined by a disjunction of pairwise conjunctions of the rest of the variables, expressed as:

$$H = \bigvee_{i=0}^{n} (X_i \wedge X_{i+1}). \tag{4.53}$$

For the concrete Logic3 model we included 8 binary variable in total.

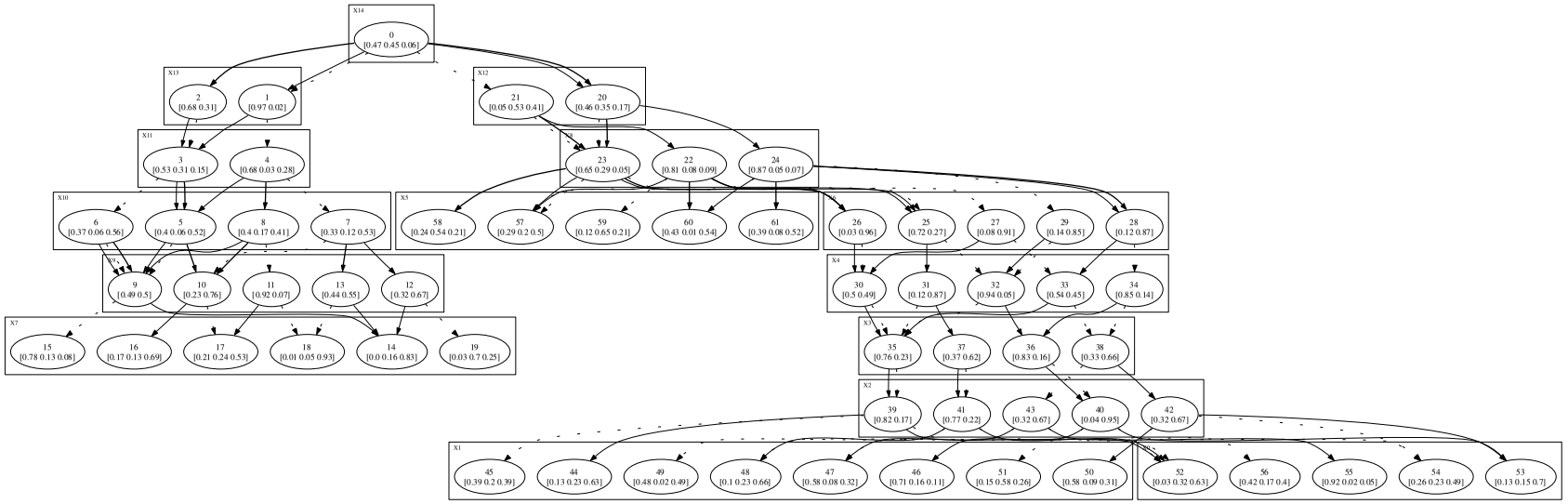
Random PDG Models In the last two experiments, we used randomly generated PDG models. Parameters were randomly generated, following the method proposed in (Caprile, 2001). The structures were forced to be single tree forests as underlying variable forests and the cardinality of variables were randomly selected to be either 2 or 3 for simplicity.

Figure 4.16(a) shows the Rnd15 model over 15 discrete random variables and with an effective size 182. Figure 4.16(b) shows the Rnd20 model over 20 discrete random variables and with an effective size 233.

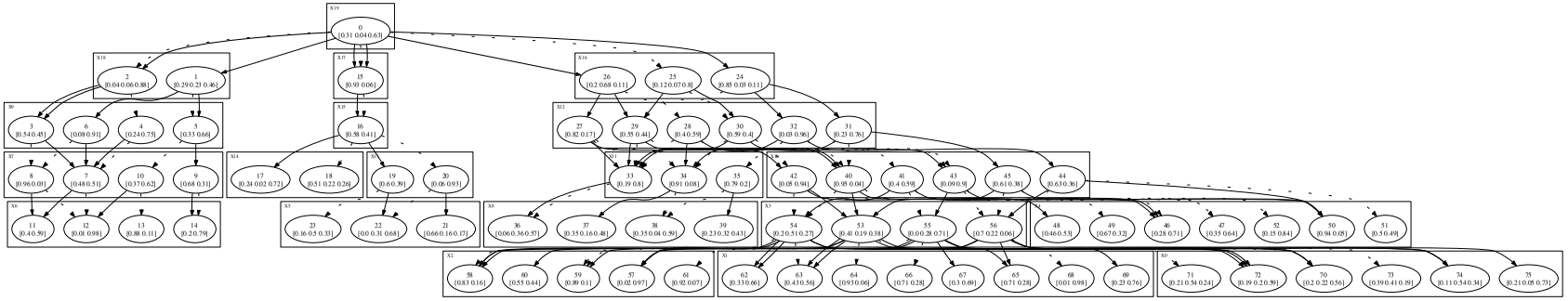
Results

The results of applying the PDG learning algorithm on the PDG-sampled data are summarised in Table 4.3. Also in Table 4.3 we list the initial size of the population of forest structures ($\#F$) and the number of λ-values for which a model was optimised ($\#\lambda$). For each dataset we report the SL-coordinates (effective size and accuracy on $\mathcal{D}_A$ and $\mathcal{D}_B$) of the model selected for optimal accuracy over test data, that is $M = \underset{M'}{argmax}\, L(\mathcal{D}_B|M')$. Figure 4.17(a) shows the learning times for both experiments measured in seconds. Figure 4.17(b) shows the effective sizes relative to the effective sizes of the true models.

Recovering Logical Models From results of Experiment 1 we observe that the true models are matched in SL-space by the learned models for Logic1-3. From the more relevant Experiment 2 where induction of the variable forest is included in the learning task, we are still successful in recovering an approximation as accurate as the true model for Logic1-3, however, only for Logic2 are we able to recover the approximation at the same effective size as the true model.



(a)



(b)

Figure 4.16. Random PDG models used in experiments. (a) shows model Rnd15 which is a randomly generated PDG model over 15 discrete random variables. The Rnd15 model has effective size 182. (b) shows model Rnd20 which is generated over 20 discrete random variables and has effective size 233.

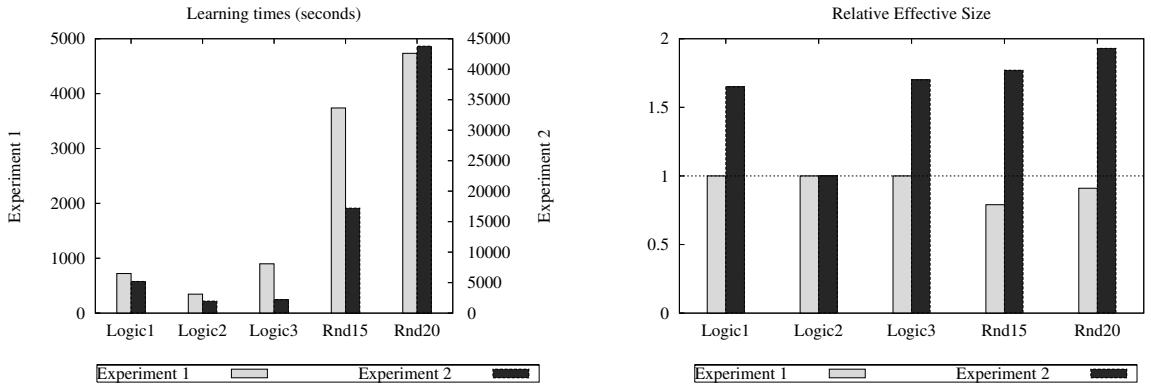
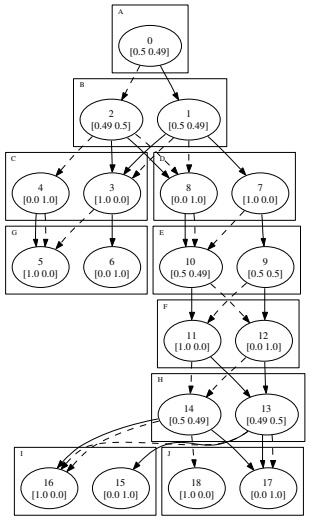


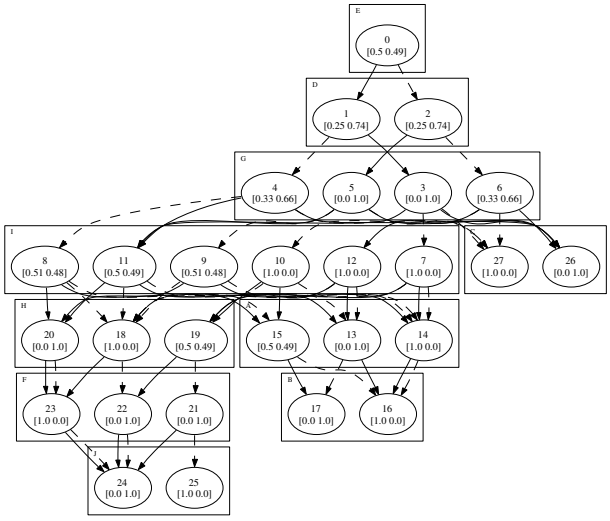
Figure 4.17. Plots showing characteristics from applying the `LearnPDG` procedure (Algorithm 4.6) to data sampled from artificial PDG models. (a) shows learning times (in seconds) of experiment 1 on the left y-axis and of experiment 2 on the right y-axis. (b) shows the effective size of the model selected in each of the experiments, relative to the effective size of the true PDG model.

			Experiment 1			Experiment 2			True model		
	$\#F$	$\#\lambda$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$
Logic1	30	22	46	-4.000	-4.000	76	-4.000	-4.000	46	-4.000	-4.000
Logic2	30	10	18	-4.000	-4.000	18	-4.000	-4.000	18	-4.000	-4.000
Logic3	30	21	40	-6.998	-7.001	68	-6.998	-7.001	40	-7.000	-7.000
Rnd15	30	16	143	-14.860	-14.859	323	-14.959	-15.037	182	-14.852	-14.833
Rnd20	30	21	211	-18.088	-18.102	449	-18.684	-18.714	233	-18.082	-18.081

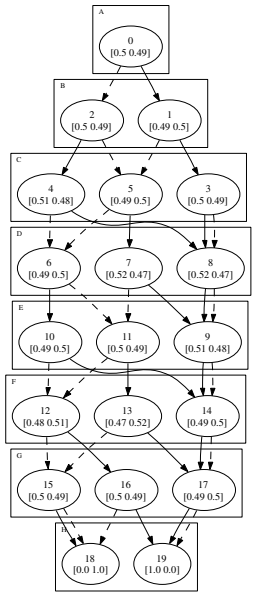
Table 4.3. Summary of our experiments on PDG sampled data. Column '$\#F$' contains the size of the initial population of variable forests, which is only relevant for Experiment 2. Column '$\#\lambda$' contains the number of lambda values for which a model was optimised. Experiment 1 and Experiment 2 refers to experiments using the correct variable forest as a starting point, and experiments where the forest is automatically induced, respectively. Columns $L(\mathcal{D}_A)$ and $L(\mathcal{D}_B)$ lists log-likelihood values for training and test data respectively (per data instance).



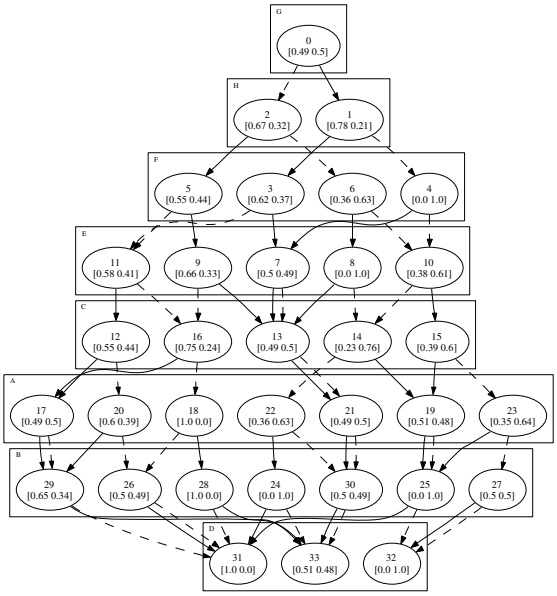
(a) Logic1, Exp. 1



(b) Logic1, Exp. 2



(c) Logic3, Exp. 1



(d) Logic3, Exp. 2

Figure 4.18. Models learned from data sampled from Logic1 ((a) and (b)) and Logic3 ((c) and (d)) models. (a) and (c) shows the model selected from Experiment using the correct variable forest structure as a starting point. (b) and (d) shows the model selected from the second experiment where no variable forest is given as a starting point.

Figure 4.18(a) and (b) shows the models selected from Experiment 1 and 2 respectively, using the Logic1 sampled data, while Figure 4.18(c) and (d) shows the models selected from experiments using the Logic3 sampled data. We observe that the recovered models in Figure 4.18(a) and (c) only differs from the corresponding true models (Figure 4.15(a) and (c)) by a few local transformations that are of no significance to the representation. Both models successfully represents the correct logical relations by assigning probability 0 to all and only the joint configurations that are false. For the models in Figure 4.18(b) and (d), the correct logical formula was not represented as some false joint configurations were assigned a non-zero probability. For the Logic2 sampled data, the correct model representing the correct logical formula was recovered in both experiments.

Recovering Random Models The results of using data sampled from the Rnd15 and Rnd20 models are quite similar, and we will discuss them in the following. For the first experiment we are not able to obtain an approximation of the same accuracy as the true models, but the selected models have smaller effective size than the true models, and they are then not dominated by the true models. For the second experiment the selected models are both less accurate and has larger effective size than the true models.

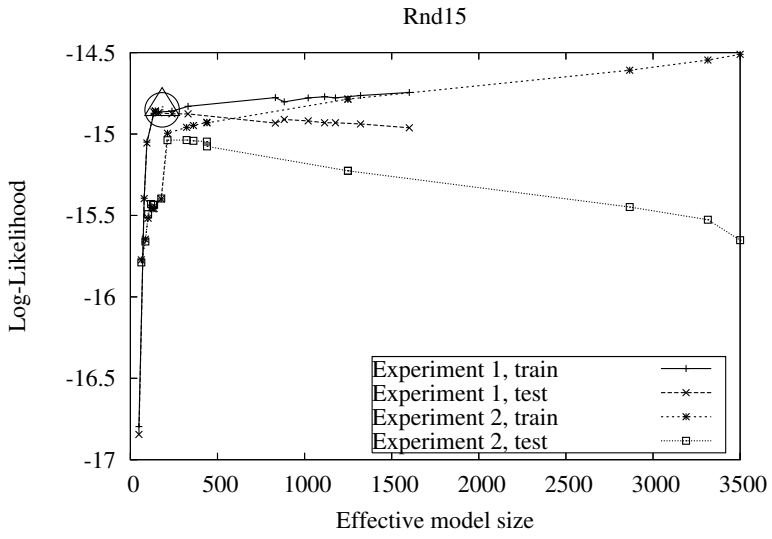
Figure 4.19(a)-(b) shows SL-curves for the four distinct experiments involving Rnd15 and Rnd20 sampled data respectively. First, from the SL-curve Figure 4.19(a) we observe that for the first experiment, the attainable level of likelihood seems to be close to the level of the true model. That is, using the correct variable forest as a starting point we do not gain much from increasing the size beyond the size of the true model. For the second experiment, where the learning procedure was not restricted to the correct variable forest, models of better accuracy over $\mathcal{D}_A$ are recovered. However, as we have already observed, these models offer a poor accuracy over $\mathcal{D}_B$.

Similar observations were made from the experiments using Rnd20 sampled data. In the corresponding SL-curves shown in Figure 4.19(b), discrepancies between the two experiments are more clear than for Rnd15 sampled data.

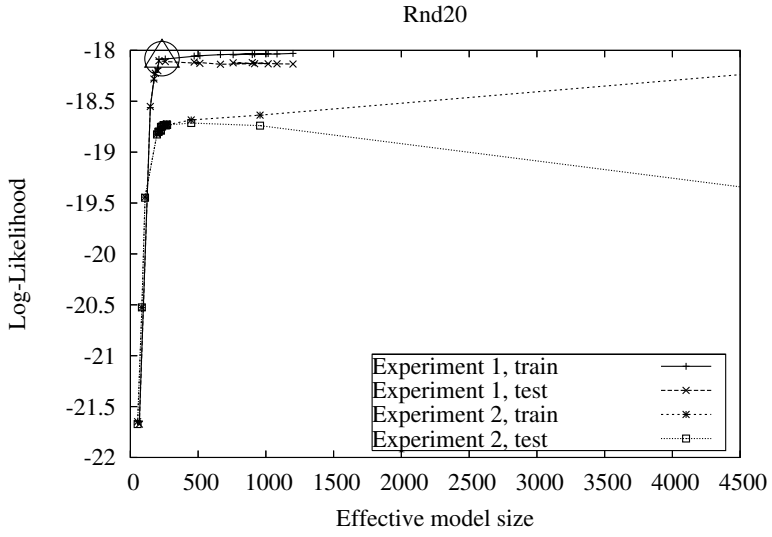
Discussion

From the observations made from the results of these preliminary experiments, we conclude that the induction of a good variable forest as a basis for the PDG learner is the harder task. It is of great importance to the quality and efficiency of the final PDG model, as we clearly observed for Rnd15 and Rnd20 sampled data. It is not surprising that the underlying variable forest can have a huge impact on the learning procedure. Any independence encoded in a variable forest is also imposed on any PDG model with that forest as underlying structure. However, if the forest fails to capture important independence relations, these must then be encoded either numerically in the parameters or in the PDG structure. Our experiments show that without the correct forest, we may need much larger structure than the correct structure to compensate for the suboptimal underlying variable forest.

On the positive side, when a good variable forest is given, our PDG learning procedure is very successful in finding good models by the local transformations. Even though relying on



(a)



(b)

Figure 4.19. SL-curves for both experiments using Rnd15 and Rnd20 sampled data. In both plots the SL-coordinates of the true model is marked with a circle (for $\mathcal{D}_A$ likelihood) and a triangle (for $\mathcal{D}_B$ likelihood). Log likelihoods are per data-instance, that is, divided by the size of the dataset.

heuristics for traversing the space of PDG models, the `merge`, `split` and `redirect` operations successfully recover high scoring models.

4.5.3 Related Work

A recent framework that is closely related to PDGs is the that of *case-factor diagrams* (CFDs) of McAllester et al. (2004). The CFD language is (like the PDG language) inspired by binary decision diagrams, and also supports computation of belief updating in time linear in the size of the representation. The structural constraints of CFD models differ from the structural constraints of PDG language in two key points: 1) CFD models do not allow undirected cycles, which means that reuse of parameters in a similar natural way as in the PDG language is not possible; and, 2) in two different paths through the CFD model, variables may occur in different orderings, which is not possible in PDG models. McAllester et al. (2004) does not propose learning procedures for CFDs, and, to our knowledge, no study on learning CFDs has been published.

A framework that is very closely related to PDGs (and CFDs) is the Independency Tree (IT) model, investigated by Flores et al. (2006). Flores et al. (2006) proposes a procedure for learning ITs from data, and reports initial and promising results when using the IT model in for clustering.

Probability estimation trees (PETs) represent a conditional probability distribution for a target variable given a set of conditioning variables, see e.g. (Provost and Domingos, 2003; Liang et al., 2006). Learning of PETs usually follow a traditional procedure for learning Decision Trees (eg. the popular ID3 algorithm (Quinlan, 1986)) with few modifications. The PET is then used to give a ranking in form of probabilities of class membership conditional on attribute variables, and CSI relations can easily be represented in a compact way. The PET framework, however, is not able to efficiently represent a joint probability distribution over a domain of variables, and therefore does not offer a natural and efficient way to perform belief updating in a domain.

Many studies have focused on using local CSI relations to improve learning of BN models. Boutilier et al. (1996) propose to use a PET representation for each local conditional distribution in a BN model. These local PETs are then used to guide a decomposition of the BN model in which auxiliary multiplexer variables are introduced to reduce the size of clique potentials in the associated Clique Tree representation. Finally, this then yields faster clique tree inference in the decomposed BN model. Thus, the local PET representation is only used as a preprocessing step to obtain a simpler BN model.

Chickering et al. (1997) use a Decision Graph (DG) representation of the local conditional distributions in a BN model, and propose an algorithm for obtaining both the BN model and local DG representations simultaneously. The learning procedures of the local DG representations proposed by Chickering et al. (1997) contains splitting and merging operators that resemble the operators presented here for PDG learning. However, the heuristics for choosing nodes for splitting and merging employed by Chickering et al. (1997) is purely random, and not guided by the gain in score as is the case for our application. Also, Chickering et al. (1997) only consider leaf nodes and not internal nodes for splitting and merging. Using the local DG structure, Chickering et al. (1997) show how to further simplify the global structure of the BN model.

The Recursive Bayesian Multinets (RBM) of Peña et al. (2002) capture CSI relations by a

decision tree over a set of distinguished variables. Each leaf ot the decision tree then contains a BN model over the variables that was not included on the path from the root to the leaf. Concerning computational complexity, RBMs aims at representing a complex domain in with many CSI relations, by a few simpler models, one for each relevant context. In the study of Peña et al. (2002), the leaf BN models are constrained to certain classes of NB models.

4.6 Combining BN and PDG Learning: A Hybrid Learning Approach

In the previous section, we observed that the variable forest induction is often the "Achilles heel" of our PDG learning procedure. Motivated by this observation, we will introduce an alternative way to handle the construction of variable forest. The material presented in this section is based on ideas previously published in (Jaeger et al., 2006).

As previously stated in Theorem 3.6, there exists an efficient translation from a clique tree model into an equivalent PDG model. Given that a clique tree model for some domain exists, we can then convert this model into an equivalent PDG model, and thereby evading the direct induction of a variable forest. This PDG model can then be exposed to the score optimising local transformations of the `LearnPDG` procedure (Alg. 4.7), and we will denote this approach as the hybrid approach.

Jaeger (2004) proposes an algorithm for performing such a conversion, and we will review this algorithm in the following. We need the following definition:

Definition 4.8 (Fully Expanded PDG)
A PDG D over variables $\mathbf{X}$ w.r.t. forest F is said to be fully expanded iff any parameter node ν has only a single parent.

From Definition 4.8, it follows that $|V_i| = |R(pa^*_F(X_i))|$ for any set of parameter nodes V_i in a fully expanded PDG D over variable forest F.

Lemma 4.5
Let $\mathbf{X}$ be a set of discrete random variables. A fully expanded PDG structure D w.r.t. any linear ordering of $\mathbf{X}$ can represent any probability distribution over $\mathbf{X}$.

Proof: Let D be a fully expanded PDG w.r.t. a linear order $X_0, X_1, \ldots, X_n$ of variable $\mathbf{X}$, that is, for the underlying variable forest F the relation $pa^*_F(X_i) = \{X_0, \ldots, X_{i-1}\}$ holds for any $X_i \in \mathbf{X}$. Furthermore, as D is fully expanded, $Path(\nu, pa^*_F(X_i))$ contains a single element from $R(pa^*_F(X_i))$ for any $\nu \in V_i$. Denote this element $\mathbf{y}$. Then by Propositions 3.4 and 3.6 $\mathbf{p}^\nu = P^D(X_i|pa^*_F(X_i) = \mathbf{y})$, and P^D factorises as:

$$P^D(\mathbf{X}) = \prod_{X_i \in \mathbf{X}} P^D(X_i|X_{i+1}, \ldots, X_n). \tag{4.54}$$

By the chain-rule of conditional distributions (2.11), any multivariate distribution factorise according to (4.54), and therefore D can represent any multivariate distribution over $\mathbf{X}$. □

Algorithm 4.13 Transforms a directed clique tree into an equivalent PDG. The underlying variable tree is build by the `buildVariableTree` procedure of Algorithm 4.14.

Input: J: clique tree.
Output: D: PDG model equivalent to J.

1: **procedure** `cliqueTreeToPDG`(J)
2: Let C_r be the root of J
3: T :=`buildVariableTree`(C_r, J)
4: Let D be an empty PDG-structure w.r.t. variable tree T
5: `buildPDGFromCliques`(C_r, J, D)
6: **return** D

Algorithm 4.14 A variable tree is build from a directed clique tree J at from clique-node C and all clique-nodes below C.

Input: C: clique of clique tree J
Output: T: variable tree representing variables of clique C and all cliques below C in J

1: **function** `buildVariableTree`(C)
2: Let T be a linear tree over variables $new(C)$
3: Let X_l be the leaf of T
4: **for all** $C_c \in ch_J(C)$ **do**
5: T_c :=`buildVariableTree`(C_c, J)
6: Attach T_c to T as a branch, rooted at X_l
7: **return** T

Lemma 4.5 states a key property of PDGs, and it is central to constructing a PDG model from a clique tree model.

Procedure `cliqueTreeToPDG` (Algorithm 4.13) implements the top-level transformation from a clique tree to an equivalent PDG. Invoking this procedure for each tree in a directed clique forest, a general clique forest is transformed to an equivalent PDG structure.

Procedure `buildVariableTree` (Algorithm 4.14) builds a variable tree from a clique tree J. The produced variable tree essentially has the same structure as J, but with each clique C exchanged for a linear order branch over certain new variables $new(C)$. $new(C)$ contains variables that appears in clique C and that have not appeared in any clique above C in the clique tree structure, that is:

$$new(C) = var(C) \setminus \{\cup_{C' \in pa^*_J(C)} var(C')\}, \tag{4.55}$$

where $var(C)$ is the set of variables associated with clique C. Any clique potential ϕ_C over clique node C is fully specified by $|R(var(C))| - 1$ parameters. The effective size of a fully expanded PDG w.r.t. variable forest F over $var(C)$ is $\sum_{X_i \in var(C)} |R(pa^*_F(X_i) \cup X_i)|$, which is bounded by $2|R(\mathbf{X})|$.

Example 4.2
Consider the clique tree of Figure 4.20(a). We have chosen the clique containing variables $\{X_1, X_2, X_3\}$ *as the root clique* C_r*, and invoke the* `buildVariableTree` *procedure on* C_r*. As*

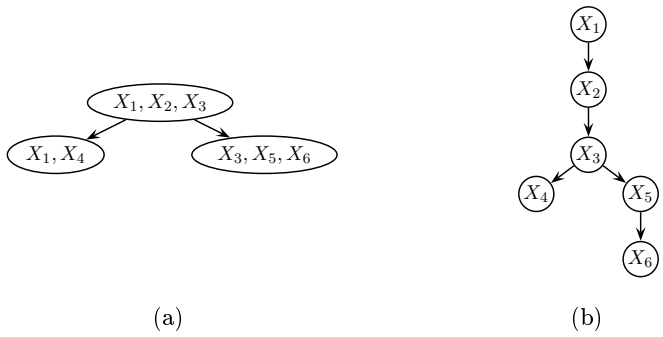


Figure 4.20. A clique tree (a) and the variable tree constructed by procedure `buildVariableTree` invoked on clique $\{X_1, X_2, X_3\}$.

Algorithm 4.15 Procedure for recursively building a PDG from a directed clique tree. Cliques are expanded into suitable sets of parameter-nodes by the `expandClique` procedure of Algorithm 4.16.

Input: C: root clique; J: clique tree; D: empty PDG structure build from clique tree J.

```
1: procedure buildPDGFromCliques(C, J, D)
2:     expandClique(C, J, D)
3:     for all C' ∈ ch_J(C) do
4:         buildPDGFromCliques(C', J, D)
```

$new(C_r) = var(C_r)$, we first build a linear tree $X_1 \rightarrow X_2 \rightarrow X_3$. For the two remaining cliques $\{X_1, X_4\}$ and $\{X_3, X_5, X_6\}$ the tree fragments X_4 and $X_5 \rightarrow X_6$ are constructed, and this finally yields the tree in Figure 4.20(b).

Procedure `buildPDGFromCliques` (Algorithm 4.15) recursively expands an empty PDG D by creating sets of parameter-nodes for all variables in the underlying variable forest. Nodes are connected such that PDG D can represent the distribution encoded by clique tree J. This task is accomplished by always matching a free parameter in the clique tree model by a corresponding free parameter in the PDG model.

The `expandClique` procedure (Algorithm 4.16) essentially ensures this, by expanding variables $new(C)$ of clique C into sets of parameter-nodes. First, variables $var(C) \setminus new(C)$ have already been included in the PDG, and we ensure that $new(C) \not\perp var(C) \setminus new(C)$ in PDG D.

The `createParameterNodes` procedure creates parameter nodes for variable X_i, and connects these nodes in PDG D such that any free parameter in the JT will be matched by a free parameter in D.

Example 4.3

Consider the variable-tree from Example 4.2 depicted in Figure 4.20(b), and assume all variables are binary. Invoking procedure `buildPDGFromCliques`(C, J, D) *(Alg. 4.15), where clique*

Algorithm 4.16 Expand a clique node C from clique tree J into sets of parameter nodes in a PDG D.

Input: C: clique node; J clique tree (containing C); D: PDG structure not containing parameter-nodes for variables $new(C)$.

```
1: procedure expandClique(C, J, D)
2:     Let F be the variable forest underlying D
3:     Y := var(C) \ new(C)
4:     for all X_i ∈ new(C) do
5:         createParameterNodes(X_i, Y, D)
6:         Y := Y ∪ {X_i}
```

Algorithm 4.17 Given a variable X_i in variable forest F and a subset of variables $\mathbf{Y} \subseteq pa_F^*(X_i)$ on which X_i depends, procedure `createParameterNodes` creates the necessary parameter nodes needed to represent this dependence in PDG D over variable forest F.

Input: X_i: random variable; $\mathbf{Y}$: set of dependent variables; D partially build PDG structure.

```
1: procedure createParameterNodes(X_i, Y, D)
2:     let F be the underlying variable forest of D
3:     let X_j = pa_F(X_i)
4:     V_i := ∅
5:     U := pa*_F(X_i)
6:     for all y ∈ R(Y) do
7:         add new parameter node ν_y to V_i
8:         for all ν ∈ V_j do
9:             for all u ∈ Path(ν, u) : u[Y] = y do
10:                set succ(ν, X_i, u[X_i]) to be ν_y
```

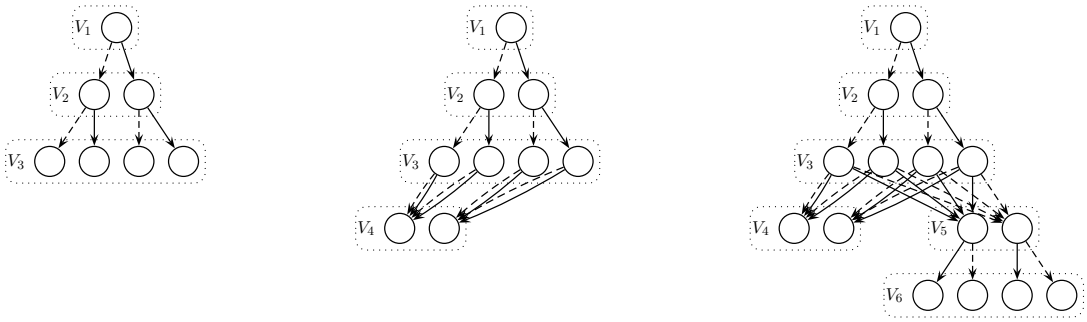


Figure 4.21. The result of applying the `buildPDGFromCliques` procedure to the clique-tree and variable forest from Example 4.2 (Figure 4.20(a) and (b)). The three steps corresponding to the three cliques of the clique-tree are depicted in sub-figures (a),(b) and (c).

C is the root of the clique tree J in Fig. 4.20(a) and D is the empty PDG structure of the variable tree in Figure 4.20(b). Figure 4.21 shows snapshots of the process of building a PDG by this procedure. First, Figure 4.21 shows the result of expanding the root clique by the `expandCligue` procedure. The clique contains variables $\{X_1, X_2, X_3\}$, and gives rise to a clique table with $2^3 = 8$ entries. To match every entry, the sub-tree over X_1, X_2 and X_3 is fully expanded. In Figure 4.21(b), the result of expanding the clique containing X_1 and X_4 can be seen. This clique gives rise to a table with $2^2 = 4$ entries over joint configurations of X_1 and X_4. Consequently, instead of expanding this subtree fully, we just create a new node $\nu \in V_4$ for each value of X_1. Figure 4.21(c) then shows the final result after expansion of the last clique.

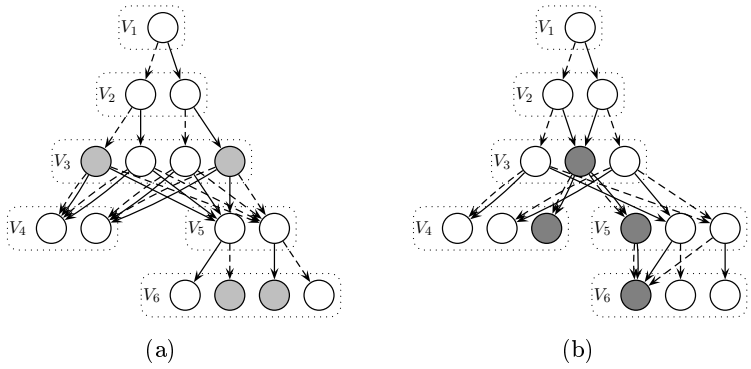


Figure 4.22. Example of collapsing non-reached nodes. Light-gray nodes in (a) are not reached by any data instance and are thus removed, creating the dark-gray garbage-nodes of (b).

Collapsing Non-reached Nodes We aim at refining the newly constructed PDG model using data. This means that we are ultimately less interested in actually capturing the distribution represented by the clique tree model, but rather we wish to construct a good approximation to the unknown generative distribution from which data was sampled. To this end, we perform an initial sweep through the newly constructed PDG model, removing nodes that are not reached by any data instances. A new "garbage"-node is created for each node-set, and any edge incoming to a node that is removed is directed into the garbage-node. For a newly created garbage-node, we can assign the garbage-node(s) of the succeeding variable(s) in the underlying variable forest as children. Such garbage-nodes ν are assigned a parameter vector $\mathbf{p}^{\nu}$ of uniform values.

Example 4.4
Consider the PDG model shown in Figure 4.22(a), and assume that the light-gray parameter-nodes are not reached by any instances $d \in \mathcal{D}_A$. Removal of non-reached nodes and creation of suitable garbage-nodes then results in the structure of Figure 4.22(b), where garbage-nodes are dark-gray.

In this toy example, the effective size of the PDG is reduced from 38 to 36, assuming all variables as binary.

Instead of keeping the garbage nodes that results from merging the non-reached nodes in the model, these garbage nodes could be removed completely. One would then need to redirect each edge incoming to a garbage-node to another existing parameter-node. This redirection could be to any other node without affecting the likelihood of training data, as no data-instances is associated with the edge. Rather, the removal would yield a sure score-improvement from the reduction in size. However, we choose to keep the garbage nodes in the model for two reasons:

1. In practise, the reduction in size resulting from completely removing garbage nodes, proved to be insignificant compared to the dramatic reduction from the initial merging of non-reached nodes.

Algorithm 4.18 The `hybridLearn` procedure learns a sequence of PDG models from an initial construction of a PDG model from a clique tree J. This initial PDG structure is then iteratively refined by a sequence of merge operations. The merge operations use increasing λ values, thus the merging of nodes will be more and more aggressive.

```
1: procedure hybridLearn(J, Λ)
2:     D := cliqueTreeToPDG(J)
3:     Collapse non-reached nodes in D
4:     for λ_min up to λ_max in Λ do
5:         mergeNodes(D, λ)
6:         output D
```

2. The garbage nodes may still be useful, even when no instance $d \in \mathcal{D}_A$ justify their existence. They provide uniform parameters for instances $d \in \mathcal{D}_B$ that still may reach them, and hence may improve the accuracy of the model.

The `hybridLearn` procedure of Algorithm 4.18 combines the approach to learning PDG models described in this section with a subsequent optimisation of the structure. We first translate a clique tree model into an equivalent PDG model. Then we perform a series of merges by the `mergeNodes` procedure (see Algorithm 4.9). The sequence of merges are increasingly aggressive, and in this way we expect to produce a series of models decreasing in size and accuracy.

4.6.1 Related Work

Darwiche (2002) propose to use Arithmetic Circuit (AC) representations for probabilistic inference. AC is a general representation framework for *multi-linear functions*, and are not dedicated to representing joint probability distributions. Unlike PDGs, no simple syntactic criterion characterise the set of ACs that do represent probability distributions. It would, therefore, seem difficult to learn ACs directly from data directly. Instead, Darwiche (2002) proposes a procedure for compiling a BN model into an equivalent AC representation, which easily capture and exploit CSI relations yielding a more computationally efficient representation. Compared to our hybrid learning of PDGs, Darwiche (2002) does not propose any optimisations of the AC after the compilation from a BN model. ACs do not naturally lend themselves to parameter re-estimation as is the case for PDGs, and re-estimation of parameters is especially important in such post-compilation optimisations to ensure that the loss in accuracy is minimised. However, the empirical results reported by Darwiche (2002) often shows a significant improvement in computational complexity of the compiled AC compared to the Clique Tree representation, even without such post-compilation optimisations.

Chapter 5

Comparative Analysis

In this chapter we perform a comparative analyses of the PGM languages presented in Chapter 3 and the methods for learning presented in Chapter 4. The overall goal of this chapter is to evaluate the ability of model languages to efficiently and accurately approximate a distribution, and to evaluate our learning methods ability to recover such efficient and accurate models. Major parts of the material presented in this chapter is based on ideas previously published in (Nielsen and Jaeger, 2006).

5.1 Methodology and Experimental Setting

We have applied our learning algorithms for BN, NB and PDG models to several datasets both real and synthetic, and produced SL-curves for each model language and each dataset. Each dataset was split up in two separate sets, one set for training (henceforth denoted $\mathcal{D}_A$) and one set of testing (henceforth denoted $\mathcal{D}_B$), and SL-curves over likelihood values obtained from both $\mathcal{D}_A$ and $\mathcal{D}_B$ was then produced. SL-curves were introduced in Section 4.1.2 as an analytical tool for cross-language comparisons.

As mentioned above, we will use both real and synthetic datasets in the comparative study. The use of synthetic data has the advantage that the generating distribution P is known. This approach is therefore popular for initial benchmarking of algorithms for the obvious reason that it avoids the difficulty of having to approximate the true generating distribution P by the empirical distribution $P^{\mathcal{D}}$ of a small sample $\mathcal{D}$ from P. Using data $\mathcal{D}$ sampled from known distributions P for the learning of model M will then enable us to evaluate the quality of the approximation provided by P^M directly by computation of $D_{KL}(P||P^M)$. However, in our analysis the obvious reasons for not only taking this approach are the following:

1. We wish to compare multiple PGM languages, and depending on the chosen distribution P we may give unfair treatment to some languages and favour others. It would be fair to assume that if data $\mathcal{D}$ has been sampled from a distribution P that is represented by a (non-trivial) model from PGM language $\mathcal{L}_1$, then P contains independence relations that are efficiently expressible in language $\mathcal{L}_1$ while these independence relations are less efficiently expressible in language $\mathcal{L}_2$, if expressible at all. Results reported in Section 5.2

support this assumption to some extent.

2. Successful learning from real data is typically the ultimate end goal of a learning algorithm. Any experiments on synthetic data is then only of interest in preliminary studies and benchmarking. In the final application of the learning algorithm, the data generating distribution will not be available, and all we have is a finite data-set of observations.

By optimising (4.8) we attempt to learn models that yield optimal effective-size/likelihood trade-offs (SL-optimal), i.e., models that are non-dominated in SL-space.[1] If the SL-curve for one model language $\mathcal{L}_1$ consistently dominates the SL-curve for another language $\mathcal{L}_2$, there can be (at least) two explanations for this:

1. for any SL-optimal $\mathcal{L}_2$ model M there exists a $\mathcal{L}_1$ model M' that dominates M (for this specific real-world distribution), or

2. we are unable to learn SL-optimal models for $\mathcal{L}_2$ by our learning procedures.

In our experiments we use real-world data, and are unable to guarantee that the SL-curve we construct consists of the SL-coordinates for SL-optimal models. We are therefore never able to conclude that explanation 1 above true. Again, as our learning procedures have no guarantees of learning SL-optimal models, explanation 2 can never be dismissed as false. Moreover, the existence of efficient and accurate SL-optimal models is of little practical value if we are unable to recover these models from data. The "practical" efficiency and accuracy of a model language will then be the efficiency and accuracy of the models we are able to learn, and these "practical" properties are then the basis for our comparative analysis.

As discussed previously (Section 4.1.2) when using SL-curves for selecting a single model, the model that attains maximal likelihood value over the testing data would typically be the canonical choice. For every experiment we will compare such models from each language. Instead of avoiding overfitting by using the test dataset $\mathcal{D}_B$ (or cross-validation when data is limited), a model optimising some fixed tradeoff between efficiency and accuracy (such as BIC or AIC scores) may be selected. We therefore also investigates the models optimising BIC and AIC scores for each dataset.

5.1.1 Empirical Accuracy and Efficiency

The analysis discussed thus far concerns the use of SL-curves that plots the tradeoffs between *effective size* and likelihood, offered by a model language. The effective size was previously introduced as a parameter of the model, such that general belief updating is computable in time linear in that parameter (see Section 4.1.2). The use of effective size allows conclusions about the differences in efficiency (of belief updating) only up to a linear factor. The linear factor depends on the specific implementation, and only then will it be measurable. We are

[1] A model M_1 is dominated by another model M_2, if M_2 has SL-coordinates that are to the left and above the other model's SL-coordinates, that is, M_2 has both smaller effective size and better likelihood score compared to M_1. Model M_1 is non-dominated if there does not exist a model M_2 from the same language that dominates M_1.

interested in this factor as conclusions may be sensitive to changing the efficiency measure from the theoretical measure of effective size to an empirically measured execution time.

We then measure the efficiency of exact inference empirically by the execution times for updating beliefs given random evidence. That is, we compute all marginal posteriors given a joint observation of a random set of evidence variables $\mathbf{E}$, and measure the average execution time of such random queries.

In addition to measuring the empirical efficiency, we also measure the empirical accuracy. Following the methodology of Lowd and Domingos (2005), a random query is generated as follows: draw an instance d at random from test data $\mathcal{D}_B$ and generate two random disjoint subsets of variables $\mathbf{Q}$ and $\mathbf{E}$ from $\mathbf{X}$. The random query is then $P(\mathbf{Q} = d[\mathbf{Q}]|\mathbf{E} = d[\mathbf{E}])$. The empirical accuracy of model M on this query, is then the log posterior probability: $\log P^M(\mathbf{Q} = d[\mathbf{Q}]|\mathbf{E} = d[\mathbf{E}])$. Compared to the global accuracy measure of log-likelihood of test data $L(\mathcal{D}_B|M)$, the empirical accuracy can be seen as a measure for "local" accuracy, i.e., restricted to specific marginal conditional distributions of P^M.

Setup of Experiments for Performing Empirical Measures

In practice, we generate n random queries, i.e., pairs of disjoint sets of variables $\langle \mathbf{Q}, \mathbf{E} \rangle$ and corresponding observations $\langle \mathbf{q} = d[\mathbf{Q}], \mathbf{e} = d[\mathbf{E}] \rangle$ extracted from randomly drawn instances d from a set of test-data (as explained above). Then, belief updating is performed in each model M both for evidence $\mathbf{E} = \mathbf{e}$ and evidence $(\mathbf{Q}, \mathbf{E}) = (\mathbf{q}, \mathbf{e})$. After a belief update, we store the joint probabilities ($P^M(\mathbf{E} = \mathbf{e})$ and $P^M(\mathbf{Q} = \mathbf{q}, \mathbf{E} = \mathbf{e})$ respectively) and the measured execution time. From the joint posteriors, we compute the empirical accuracy $\log P(\mathbf{Q} = \mathbf{q}|\mathbf{E} = \mathbf{e})$. In this way, we measure both the empirical efficiency of belief updating and the empirical accuracy of joint posteriors given random evidence.

5.1.2 General Experimental Setup

For learning BN models, the `KES` procedure (Algorithm 4.3) with the S^{BN}_{λ} score (see (4.24)) was used. BN models were optimised for a range of different λ values, and for each value of λ we used 11 different k values $k \in \{0.0, 0.1, \ldots, 1.0\}$. For each pair of k and λ, 100 restarts of `KES` was performed, and for each specific λ value the highest scoring BN model was selected.

For learning PDG models, we used the `LearnPDGs` procedure of Algorithm 4.6. The initial population size was manually tuned for each dataset, as was the specific significance levels used in the conditional independence tests in building the initial variable forests for each dataset.[2]

Finally, for learning NB models, the NB learning algorithm described in Section 4.4 was used. Recall that learning NB models with increasing effective size is especially simple as the

[2]The manual tuning of the initial population size and the significance levels was aimed at learning a range of different models. For some initial settings we experienced that the learning procedure was only able to recover a small set of different models. More specifically, we would typically start with a small population size and subsequently increase the size if the variance in learned models turned out to be too small. Also, the significance level used in the independence test would sometimes yield forest structures so simple that only a very small set of different PDG structures were possible. In such cases we would restart the procedure with less strict significance levels.

Name	$\|\mathbf{X}\|$	$\|\mathbf{E}\|$	R_{max}	R_{min}	R_{mean}	$size_{eff}$	$L(\mathcal{D}_A\|P^M)$	$L(\mathcal{D}_B\|P^M)$
Alarm	37	42	2	4	2.8	771	-13.720	-13.839
Hailfinder	56	66	2	11	4.0	9406	-70.812	-70.785

Table 5.1. Characteristics of the BN models used for sampling synthetic data. columns R_{max}, R_{min} and R_{mean} lists maximum, minimum and mean range of the random variables, $size_{eff}$ lists the effective size of the model, while $L(\mathcal{D}_A|P^M)$ and $L(\mathcal{D}_B|P^M)$ lists log-likelihood values of the models averaged over instances in the respective datasets.

structure is given and the only parameter that affects the efficiency is the number of latent components. The termination criterion for the EM algorithm (that is, maximum iterations and minimum change in parameters) was tuned manually for each dataset.[3]

Implementations The `KES` procedure (Alg. 4.3) was implemented in the C language, using efficient state-of-the-art Machine Learning libraries.[4] Both the `LearnPDGs` (Alg. 4.7) and the `LeandNB` (Alg. 4.5) procedures were implemented in the Java language using standard libraries of JDK v. 1.5 and the Weka package for basic data handling routines.[5, 6] All the learning experiments was performed on a Sun Fire X4100, 2.4 GHz CPU architecture with 4096 MB RAM running the RedHat-Enterprise Linux4 64bit operating system.

5.2 Learning from Synthetic Data

We will learn models from a collection of synthetic databases. Each database was generated by drawing random samples under a distribution represented by a known model. For producing SL-curves, we will use log-likelihood values averaged over the size of the data, and we will include in the plots a horizontal line representing the (negative) entropy of the data $-H(\mathcal{D}_A)$, as this is the maximal attainable log-likelihood value for any model.

5.2.1 Learning from BN Generated Data

In this section, we report on experiments using data sampled from manually constructed BN models. We use two widely studied models, the Alarm network (Beinlich et al., 1989) and the Hailfinder network (Abramson et al., 1996). We used data sampled from these models, in Section 4.3.4. Data sampled from these models was previously used in testing the `KES` procedure in Section 4.3.4. Characteristics of these networks can be seen in Table 5.1.

[3] The tuning was mainly necessary in order to ensure acceptable run times. For the larger datasets it was necessary to terminate EM after fewer iterations than for smaller datasets. The run time of EM is of cause directly dependent on the cardinality of the latent components.

[4] These libraries were initially developed at the AutonLab, Carnegie Mellon University, and most kindly made available to us by Dr. Andrew W. Moore.

[5] For information on the Java language, see `http://java.sun.com/`

[6] Weka is a library of tools and algorithms for Machine Learning and Data Mining tasks implemented in Java. The libraries can be obtained online at `http://www.cs.waikato.ac.nz/~ml/weka/`, and for futher detail on the Weka-toolbox, see (Witten and Frank, 2005).

Name	$\|\mathbf{X}\|$	$\|R(C)\|$	R_{min}	R_{max}	R_{mean}	$size_{eff}$	$L(\mathcal{D}_A\|P^M)$	$L(\mathcal{D}_B\|P^M)$
NB10	15	10	2	4	3.13	470	-20.468	-20.512
NB20	15	20	2	4	2.9	880	-19.641	-19.614

Table 5.2. Characteristics of NB models used for generating synthetic NB data. Columns $L(\mathcal{D}_A|P^M)$ and $L(\mathcal{D}_B|P^M)$ lists likelihood values (averaged over instances in the training data) for datasets generated from the respective models.

Results on BN Generated Data

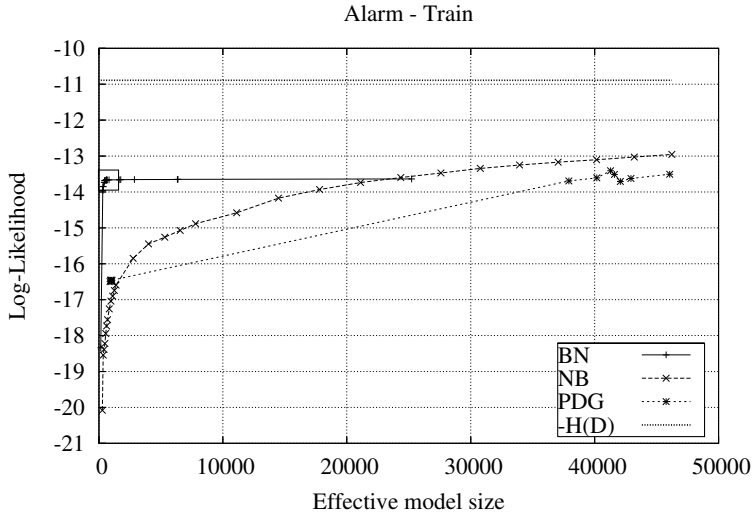
Figure 5.1 shows SL-curves generated from models learned from data sampled from the Alarm model. As expected, the BN models shows superior performance and consistently dominates NB and PDG models in Figure 5.1(b) where likelihoods are computed over $\mathcal{D}_B$. In Figure 5.1(a) where likelihoods are computed over $\mathcal{D}_A$ BN models dominates PDG and NB models only up to a certain effective size. The SL curves for BN models raises quickly to the level of accuracy of the generating model and then does not improve accuracy for models of increased complexity. NB models show a much more smooth increase in accuracy for increasingly complex models. For PDG models we have a large interval of effective size where no models where learned, which is probably due to poor tuning of the parameters in the learning procedure. When tuning the parameters, we were trying carefully to avoid such gabs in the SL-curves. The reason they still appear can have (at least) two explanations. Either there simply do not exist models in the range where we do not observe models, or we are unable to learn these models. Assuming there exists models, we might have chosen a set of significance levels that produce forest structures that do not support these models, and hence we are not able to learn them. Thus, poor tuning of the parameters could result in the observed SL-curves.

However, SL coordinates for the learned PDG models that are learned are close to NB models, and we therefore do not expect major differences in the performance of the PDG language compared to the NB language even for model sizes we have not observed.

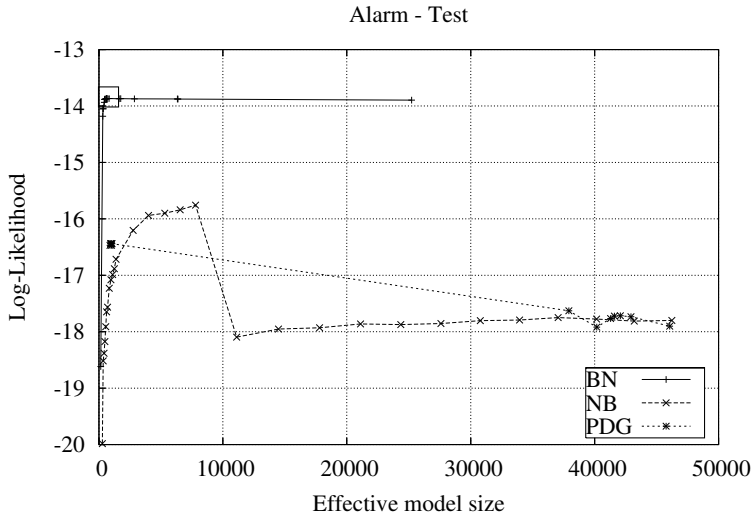
Observations similar to these were made from the experiment on data sampled from the Hailfinder model, the only difference being that NB models does not as clearly overfit the Hailfinder data as it is the case for the Alarm data. SL-curves for the experiments using data sampled from the Hailfinder model can be found in Appendix A (Figure A.1).

5.2.2 Learning from NB Generated Data

We have used 2 randomly generated NB models (NB10 and NB20) over 15 discrete random variables with ranging from binary valued to 5 state variables. The NB10 model has 10 latent components and NB20 has 20 latent components. Datasets $\mathcal{D}_A$ of size 10000 and $\mathcal{D}_B$ of size 5000 were sampled. Table 5.2 contains characteristics of the models.

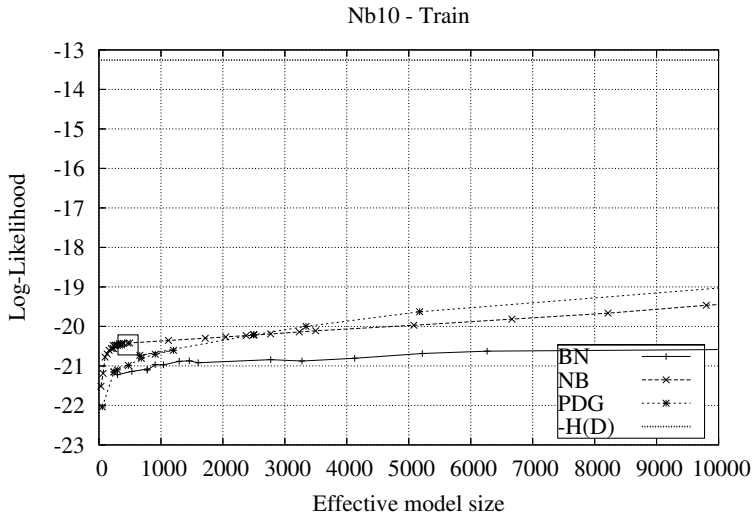


(a)

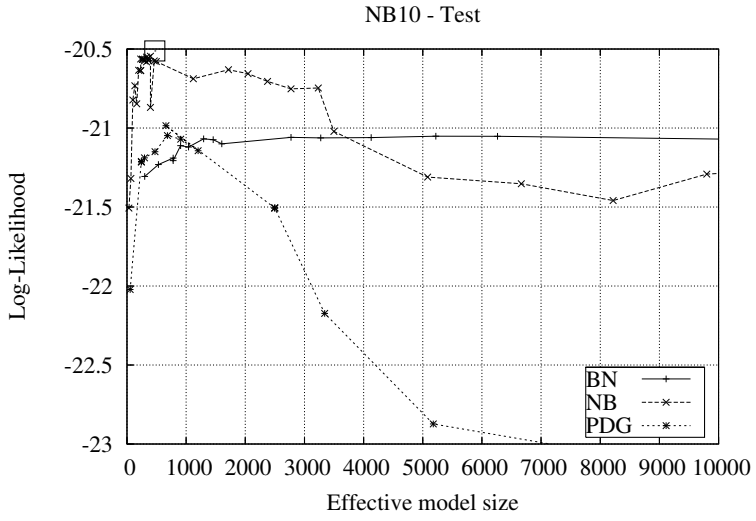


(b)

Figure 5.1. SL-curves for models learned from the Alarm data, for likelihood values over training data $\mathcal{D}_A$ (a) and test data $\mathcal{D}_B$ (b). The SL coordinates for the generative model is marked with a square. The Log-Likelihood is per data-instance, that is, divided by the data size ($|\mathcal{D}_A|$ and $|\mathcal{D}_B|$, respectively). Log likelihoods are per data instance.



(a)



(b)

Figure 5.2. SL-curves for models learned from the NB10 data, for likelihood values over training data $\mathcal{D}_A$ (a) and test data $\mathcal{D}_B$ (b). The SL coordinates for the generative model is marked with a square. Log likelihoods are per data instance.

Name	$\|\mathbf{X}\|$	R_{min}	R_{max}	R_{mean}	$size_{eff}$	$L(\mathcal{D}_A\|P^M)$	$L(\mathcal{D}_B\|P^M)$
Logic1	10	2	2	2	46	-4.000	-4.000
Logic2	5	2	2	2	18	-4.000	-4.000
Logic3	8	2	2	2	40	-7.000	-7.000
Rnd15	15	2	3	2.6	182	-14.852	-14.833
Rnd20	20	2	3	2.4	233	-18.082	-18.081

Table 5.3. Characteristics of PDG models used for generating synthetic PDG data. Columns $L(\mathcal{D}_A|P^M)$ and $L(\mathcal{D}_B|P^M)$ lists log-likelihood values (averaged over the number instances in training data) for datasets generated from the respective models.

Results on NB Generated Data

Figure 5.2 shows SL-curves from learning from NB10 sampled data. We first observe that for models of small effective size, NB models outperform both BN and PDG models as expected. However, no single language consistently dominates the other languages in neither Figures 5.2(a) nor 5.2(b). In Figure 5.2(a) BN models are consistently dominated, while in Figure 5.2(b) no language is consistently dominated. In Figure 5.2(b) we observe a remarkable stability in accuracy of the BN models that is not observed for neither PDG nor NB models. PDG models in particular seems to suffer from overfitting $\mathcal{D}_A$.

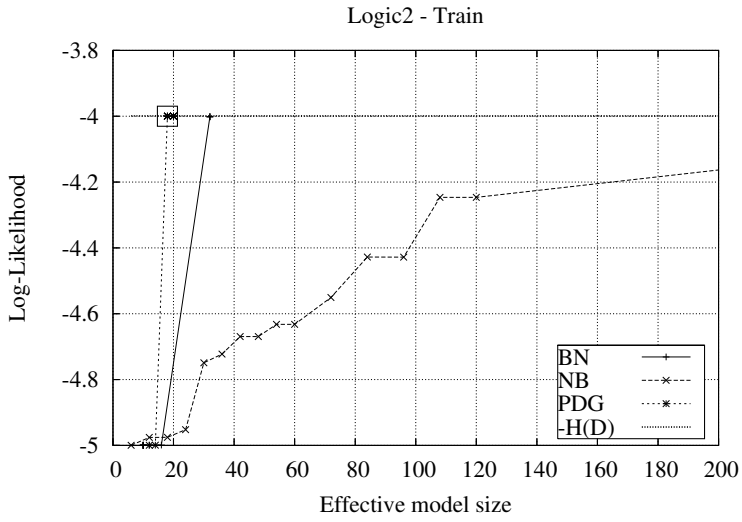
The results of experiments on NB20 sampled data leads to similar observations and does not lead to new conclusions. Figure A.2 in Appendix A contains SL-curves from learning from NB20 sampled data.

5.2.3 Learning from PDG Generated Data

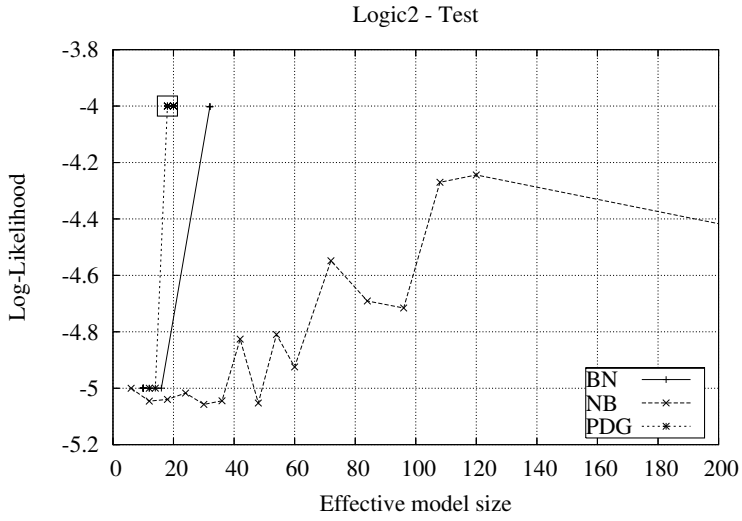
Three datasets were sampled from the manually constructed Logic1-3 PDG models (see Figure 4.15). The datasets are the same as the ones used for initial benchmarking of the learning procedure for PDG models, as discussed in Section 4.5.2.

Two datasets were sampled from randomly generated PDG models, the Rnd15 and Rnd20 models (see Figure 4.16). These datasets were also used in the initial benchmarking of the PDG learning algorithm, as discussed in Section 4.5.2.

Results on PDG Generated Data Figure 5.3 shows SL-curves from learning from Logic2 sampled data. Recall that the Logic2 model encodes the parity distribution over 5 binary variables. We observe the expected superiority of PDG models over both BN and NB models. BN models are capable of approximating the distribution as accurately as PDG models, however, BN models can only represent the parity distribution exactly with an effective size that is exponential in the number of parameters (as previously discussed, see Section 3.3.3). In the case of Logic2 with $n = 5$ we get $2^5 = 32$, which is exactly the effective size of the BN model that attains maximum likelihood value in Figure 5.3. The NB models fail to provide an efficient approximation for this dataset. As previously discussed (Section 3.3.3), the NB model will need an effective size of $5 * 2^5 + (2^5 - 1) = 191$ to represent the parity distribution over 5 binary variables exactly. In our experiments we were not able to recover this model.



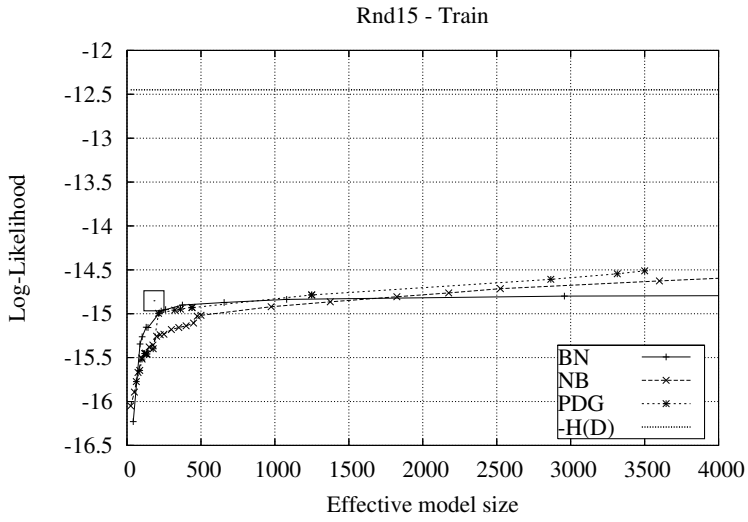
(a)



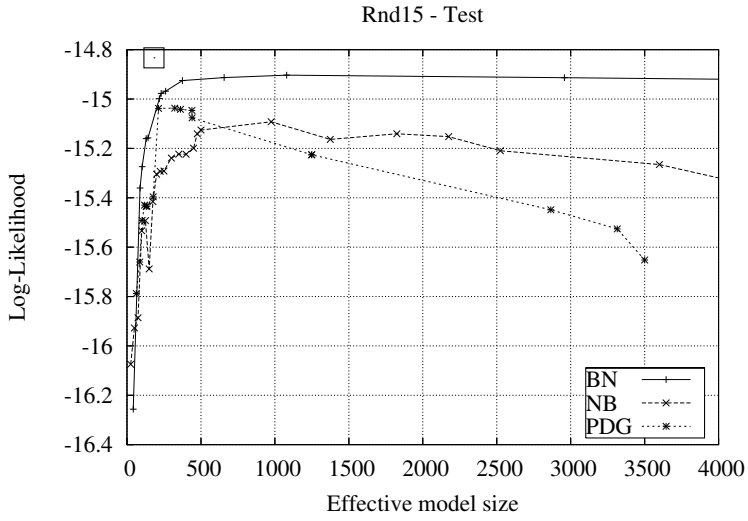
(b)

Figure 5.3. SL-curves for models learned from the Logic2 data, for likelihood values over training data $\mathcal{D}_A$ (a) and test data $\mathcal{D}_B$ (b). The SL coordinates for the generative model is marked with a square. Log likelihoods are per data instance.

The smallest effective size of a model with maximum likelihood was only learned when the cardinality of the latent variable was increased to 40, yielding effective size of 239. This is not particularly surprising as it is well known that the EM algorithm is prone to get trapped in local optima. In representing the parity distribution, the NB model needs to represent every joint configuration over the variables using a single component. More than one component of



(a)



(b)

Figure 5.4. SL-curves for models learned from the Rnd15 data, for likelihood values over training data $\mathcal{D}_A$ (a) and test data $\mathcal{D}_B$ (b). The SL coordinates for the generative model is marked with a square. Log likelihoods are per data instance.

the latent variable may represent the same configuration, that is, the component conditional $P(X_i|C = c_l) = P(X_i|C = c_k)$ for some pair of components $c_l \neq c_k$ and for all $X_i \in \mathbf{X}$. This is actually quite likely given that our NB learning algorithm uses instances drawn at random from $\mathcal{D}_A$ to instantiate new components, after the cardinality has been incremented (see Section 4.4). Therefore, we need more than the theoretical optimal 32 components to

represent the distribution exactly. This problem could be mitigated by merging equivalent components after termination of EM. The potential benefit from including an operator for merging of components in learning NB models is well studied, a detailed discussion is provided in (Karčiauskas, 2005). Our reason for not including such an operator was mainly to reduce the learning time. Also, we are aiming at producing a range of NB models of different size, and the merging operator is specifically aimed at finding the model with optimal latent cardinality.

The SL-curves for learning from Rnd15 sampled data can be seen in Figure 5.4. There are only small differences in the characteristics of the curves for likelihood values over $\mathcal{D}_A$ in Fig. 5.4(a). In Figure 5.4(b) however, BN models show very stable performance and consistently dominates BN and PDG models. For both PDGs and NBs, overfitting $\mathcal{D}_A$ is very clear, while BN models again are very stable in accuracy.

SL-curves for the experiments of learning from data sampled from the Hailfinder, Logic1, Logic3 and Rnd20 models can be found in Appendix A.1.

5.2.4 Discussion of Results

One general conclusion that can be drawn from learning from the synthetic datasets is that generally, the language of the model from which the data was sampled, is often the superior language for accurate and efficient approximations of the empirical distribution. Exceptions to this observations are the experiments of learning from the Logic1 and Rnd20 sampled data where BN models outperform the generative language of PDGs.

Table 5.4 contains SL coordinates for the models of maximal likelihood over $\mathcal{D}_B$, the BIC optimal models and the AIC optimal models. The SL coordinates of the generative models can be found in Tables 5.1, 5.2, and 5.3.

From the numbers in Table 5.4 we see that both BIC and AIC scores select models with an accuracy relatively close to the accuracy of the $M_{max(L(\mathcal{D}_B))}$ models, while (of-course) AIC punishes less for complexity when compared to BIC. The expected effect of reducing the punishment for increased size would be to overfit to $\mathcal{D}_A$, and indeed we observe this effect. When comparing the SL-coordinates of the learned models to the SL-coordinates of the generative models (see Tables 5.1, 5.2, and 5.3) we do not see any learned models dominating the generative model for any of the datasets.

The main conclusion we draw from the results of these experiments is first of all that no single PGM language proves to consistently outperform the others and no single language is consistently outperformed by the others. Also, when considering the $M_{max(L(\mathcal{D}_B))}$ selected models, NBs and BNs seem to have trouble approximating a distribution represented by the opposite model. That is, NB models perform poorly both concerning accuracy and efficiency on Alarm and Hailfinder sampled data while BN models have exhibits a blowup in effective size in order to approximate the NB10 and NB20 sampled data. The PDG models provide inaccurate approximations only for the Alarm and Hailfinder sampled data. For the Rnd15 and Rnd20 randomly generated PDG models we are somewhat surprised to observed BN models as providing the more accurate approximation than compared to PDG models at only a slightly larger effective size. Comparing the obtained BN models to the SL-coordinates of

		$M_{max(L(\mathcal{D}_B))}$			M_{BIC}			M_{AIC}			#models	Time
		$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	learned	(seconds)
Alarm	BN	624	-13.666	-13.868	496	-13.678	-13.879	6372	-13.649	-13.876	20	67845.31
	PDG	954	-16.462	-16.438	889	-16.496	-16.47	945	-16.459	-16.445	13	57635.79
	NB	7797	-14.886	-15.757	1380	-16.595	-16.716	14490	-14.171	-17.952	29	283008.18
Hailfinder	BN	8472	-70.804	-70.976	2884	-70.917	-71.03	38871	-70.8	-70.983	16	121282.84
	PDG	1486	-84.227	-84.456	1246	-84.345	-84.467	57780	-73.108	-101.16	10	658497.00
	NB	80808	-79.511	-89.222	672	-92.06	-92.126	73080	-80.056	-89.344	50	134632.92
NB10	BN	5220	-20.687	-21.051	780	-21.104	-21.207	6264	-20.627	-21.053	15	19014.18
	PDG	659	-20.731	-20.985	659	-20.731	-20.985	24247	-17.264	-24.098	16	46652.85
	NB	396	-20.431	-20.546	231	-20.482	-20.563	11352	-19.308	-21.257	42	71852.11
NB20	BN	4320	-19.819	-19.991	275	-20.065	-20.09	5952	-19.758	-20.004	15	19211.51
	PDG	422	-19.927	-20.070	312	-20.000	-20.078	12162	-17.252	-23.899	16	33386.83
	NB	450	-19.619	-19.670	360	-19.659	-19.716	8310	-18.776	-20.244	68	173017.08
Logic1	BN	60	-4.003	-4.003	44	-4.003	-4.003	84	-4.002	-4.003	11	7184.12
	PDG	78	-3.999	-4.000	76	-4.000	-4.000	76	-4.000	-4.000	22	5199.08
	NB	78	-3.999	-4.000	440	-3.999	-4.001	440	-3.999	-4.001	50	23343.29
Logic2	BN	32	-4.002	-4.002	32	-4.002	-4.002	32	-4.002	-4.002	13	4041.13
	PDG	20	-4.000	-4.000	18	-4.000	-4.000	18	-4.000	-4.000	10	1964.86
	NB	360	-3.999	-4.000	360	-3.999	-4.000	360	-3.999	-4.000	51	12840.00
Logic3	BN	256	-6.997	-7.015	144	-7.051	-7.057	256	-6.998	-7.016	18	10059.47
	PDG	68	-6.998	-7.001	68	-6.998	-7.001	86	-6.996	-7.004	21	2195.05
	NB	702	-6.991	-7.008	162	-7.030	-7.076	360	-6.999	-7.053	50	19359.38
Rnd15	BN	1080	-14.837	-14.903	375	-14.901	-14.925	1080	-14.837	-14.903	18	41633.18
	PDG	323	-14.959	-15.037	213	-14.996	-15.038	3500	-14.511	-15.652	17	17932.61
	NB	975	-14.92	-15.092	475	-15.028	-15.141	2525	-14.715	-15.21	31	38460.00
Rnd20	BN	674	-18.079	-18.129	674	-18.079	-18.129	4332	-18.047	-18.136	16	125469.22
	PDG	449	-18.684	-18.714	245	-18.735	-18.734	6811	-17.935	-19.789	22	43711.43
	NB	2117	-18.402	-18.765	580	-18.990	-19.052	5539	-17.933	-19.057	29	33900.00

Table 5.4. SL-coordinates for the model of maximal likelihood over $\mathcal{D}_B$ ($M_{max(L(\mathcal{D}_B))}$), the BIC optimal model (M_{BIC}) and AIC optimal model (M_{AIC}). Columns labelled $size_{eff}$ lists effective size, and columns labelled $L(\mathcal{D}_A)$ and $L(\mathcal{D}_B)$ lists log-likelihood values computed over $\mathcal{D}_A$ and $\mathcal{D}_B$ respectively and averaged over the number of instances in the dataset.

Name	$\lvert\mathbf{X}\rvert$	$\lvert\mathcal{D}_A\rvert$	$\lvert\mathcal{D}_B\rvert$	R_{min}	R_{max}	$\lvert R(\mathbf{X})\rvert$	$H(\mathcal{D})$	$-\log_2(\frac{1}{\lvert\mathcal{D}\rvert})$
Page-blocks	11	4482	574	5	5	10^7	10.669	12.304
Letter Recognition	17	18012	1988	4	26	10^{12}	13.828	14.288
Landsat	37	4435	2000	5	6	10^{25}	12.349	12.652
Adult	15	30162	15060	2	41	10^{11}	13.561	15.465
King,Rook vs. King	7	25188	2868	4	18	10^6	14.776	14.776
Abalone	8	3758	419	3	5	10^6	9.193	12.028
Poisonous Mushroom	23	7337	787	2	11	10^{14}	12.988	12.988

Table 5.5. Summaries of the real datasets used in the analysis. $\mathcal{D}$ refers to the full dataset, $\mathcal{D}_A$ is the part of $\mathcal{D}$ used for training, $\mathcal{D}_B$ is the part of $\mathcal{D}$ used for testing, R_{max} and R_{min} refers to the maximunm and minimum range of the variables $\mathbf{X}$ observed in the data. $H(\mathcal{D})$ is the entropy of the data.

the Rnd15 and Rnd20 models, we see that there indeed exists PDG models with the same level of accuracy. And recalling the successful results of learning from this data using the correct tree structure as a starting point (see Section 4.5.2) we see that one explanation of BNs being more accurate than PDGs on these datasets could be the induction of incorrect underlying variable forests.

5.3 Learning from Real Data

In this section we report on the results of learning PGMs from real datasets. The datasets we have used are available online at the UCI ML repository (Newman et al., 1998) in their original form. Table 5.5 contains a short summary of the datasets used. If a standard training/test partitioning of the original dataset were available we used it, otherwise instances where randomly assigned to either $\mathcal{D}_A$ (90%) or $\mathcal{D}_B$ (10%).

We include a short description of the datasets below.

Page-blocks This dataset contains instances of blocks of the page layout of a text document. A document has been pre-processed by a segmentation process, partitioning each page into disjoint blocks where each block has been labelled as either "text", "horizontal line", "picture", "vertical line" or "graphic". For each block, 10 different features (height, length, number of black pixels, etc.) of the block are recorded, and the label together with the value of these 10 features then makes up an instance in the dataset. This dataset has previously been used for evaluation of decision tree learning, e.g., Esposito et al. (1997).

The 10 features were originally numerical values, to avoid working modelling continuous random variables, we have discretised each of the 10 features into 5 equal frequency bins. We include the class label as a regular variable in our dataset.

Letter Recognition Each instance of this dataset contains label specifying one of the 26 capital letters from the English alphabet, plus 16 primitive measurements of a black-and-white rectangular pixel display when displaying this character. Each character was displayed in 20

different fonts, and each display were randomly distorted before the 16 measurements were recorded. We include the class label as a regular variable in our dataset.

Landsat This dataset contains information extracted from digital satellite images of land surfaces. Each case in the database is extracted from a 3×3 pixel image, with values for each pixel for 4 different spectral bands, thus totals 36 features. Each such feature is encoded as a 8 bit word, hence the range is 0 to 255. Each case is then augmented with a class label, labelling each case with one of 6 different types of surface. We have reduced the range of the 36 features to 5 approximately equal frequency bins.

Adult This dataset was extracted from a 1994 US Census database. Each instance contains values for 14 features (age, sex, marital status, race, work-class, education, etc.) and a class label indication whether the yearly income of the individual is above or below $USD 50.000. Past usage of this dataset has been aiming at developing classifiers for predicting the income-label of an individual given the values of the features. We have discretised numerical valued features into 5 equal frequency bins. We include the class-label into our dataset as a regular variable.

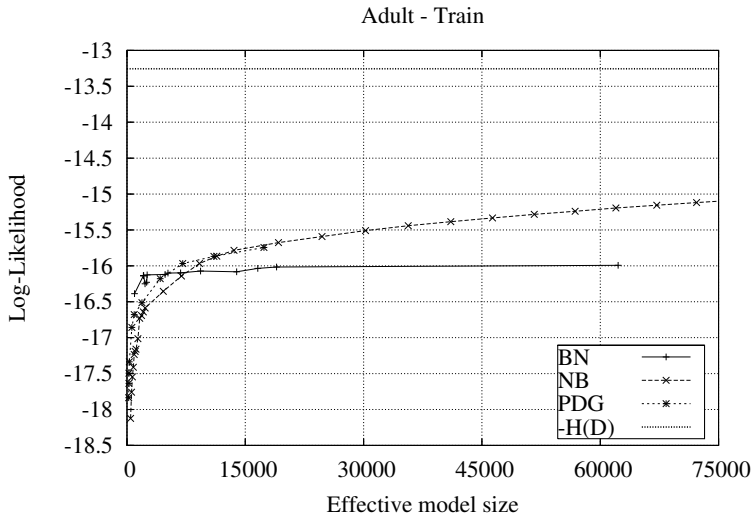
King, Rook vs. King This dataset is constructed from chess endgames in which only three pieces are left on the board, white king, white rook and black king. Each instance contains coordinates for each piece and a value for the optimal depth of win for white ranging from 0 to 16 moves. If white can not win within 16 moves a special "draw" state is recorded.

Abalone This dataset is made up of measurements of features of the abalone shellfish such as lenght, height, weight, age etc. In total 9 different features are recorded for a single abalone, 8 of them having numeric values. This dataset has previously been used for learning to classify the age of the abalone, based on the rest of the features. For our experiments, numeric valued features were discretised into 5 equal frequency bins.

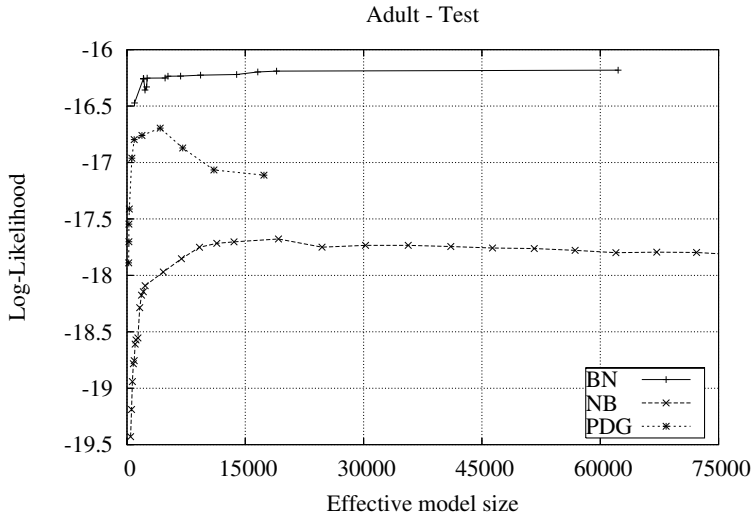
Poisonous Mushrooms This dataset is made up of features of different mushrooms such as colour, shape, size, etc. In total, 22 nominal features are recorded and each case is augmented with a label classifying the mushroom as either edible or poisonous.

5.3.1 Discussion of Results

Figure 5.5(a)-(b) shows SL-curves from learning from the Adult database. We observe very similar performance of all three languages on $\mathcal{D}_A$, although BN models gain less in likelihood when increasing complexity compared to both PDG and NB models. In Figure 5.5(b), we observe clear dissimilarities in performance. While BN models are still very stable and likelihood values are not affected in either direction by increasing complexity of the models, both PDG and NB models suffer from overfitting $\mathcal{D}_A$. In Figure 5.5(b), the BN language consistently dominates while the NB language is consistently dominated. The observations of dominance in Figure 5.5(b) is clearer than what we have observed for other datasets.



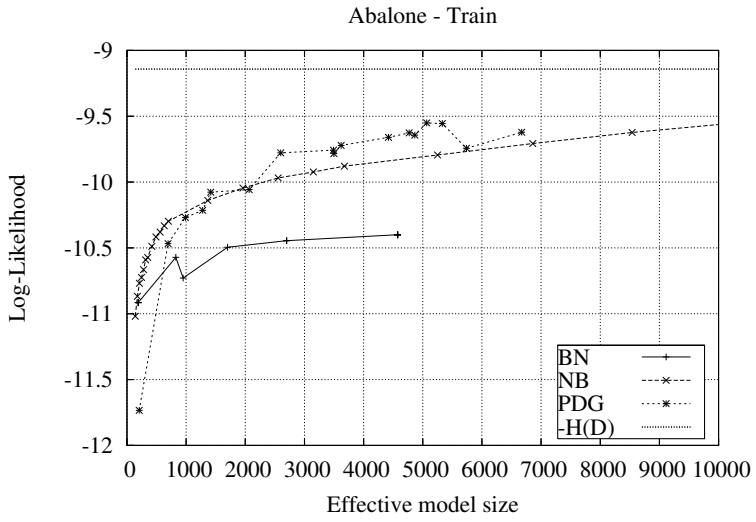
(a)



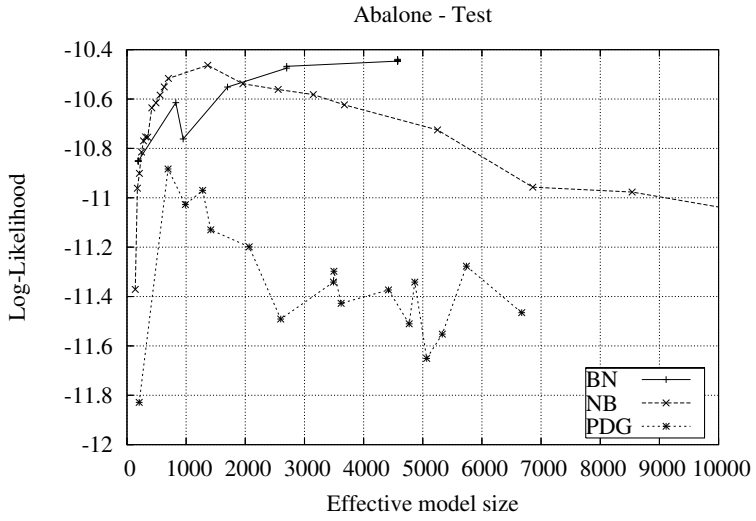
(b)

Figure 5.5. SL-curves from learning from the Adult dataset. Figure (a) displays plots using likelihoods over $\mathcal{D}_A$ and (b) displays plots of likelihoods over $\mathcal{D}_B$. Log likelihoods are per data instance.

Figure 5.6(a)-(b) shows SL-curves from learning from the Abalone data, and this is a more typical set of SL-curves where no language consistently dominates the others. For $\mathcal{D}_A$ (Fig. 5.6(a)) the PDG and NB models are again observed to benefit more in accuracy by the increase in complexity than does the BN models. For $\mathcal{D}_B$ (Fig. 5.6(b)), however, models from the PDG language are consistently dominated. NB and BN models offer an approximation of almost the same maximum accuracy over $\mathcal{D}_B$, while the more efficient approximation is offered by the NB model.



(a)



(b)

Figure 5.6. SL-curves from learning from the Abalone data. Figure (a) displays plots of likelihood values over $\mathcal{D}_A$ while plots in (b) uses likelihood values over $\mathcal{D}_B$. Log likelihoods are per data instance.

Tables 5.6 and 5.7 summarise the results from learning from real data.[7] First, Table 5.6 lists observed dominance. For each dataset we observe 1) if one language consistently dominates the others and 2) if one language is consistently dominated by the others. These observations are made for both $\mathcal{D}_A$ and $\mathcal{D}_B$, and in Table 5.6 we denote by $\mathcal{L}_1/\mathcal{L}_2$ the observation that language $\mathcal{L}_1$ consistently dominates the others and language $\mathcal{L}_2$ is consistently dominated by

[7] See Appendix A.2 for SL-curves for models learned from the Page-blocks, Letter Recognition, King Rook vs. King, Poisonous Mushroom and Landsat data.

	Page-blocks	Letter R.	Adult	K.R.v.K	Abalone	P. Mushroom	Landsat
$\mathcal{D}_A$	-/BN	-/BN	-/-	NB/BN	-/-	-/BN	NB/-
$\mathcal{D}_B$	PDG/-	NB/PDG	BN/NB	-/NB	-/PDG	PDG/NB	BN/PDG

Table 5.6. Summary of observations of consistent dominance. Row $\mathcal{D}_A$ lists consistent dominance observed for SL-curves of log likelihoods over $\mathcal{D}_A$, and $\mathcal{D}_B$ for SL-curves of log likelihoods over $\mathcal{D}_B$. For each dataset we list two observations in the format '$\mathcal{L}_1/\mathcal{L}_2$', which denotes that language $\mathcal{L}_1$ consistently dominates in this experiment, while $\mathcal{L}_2$ is consistently dominated. Either of the two or both might not be observed, indicated by -.

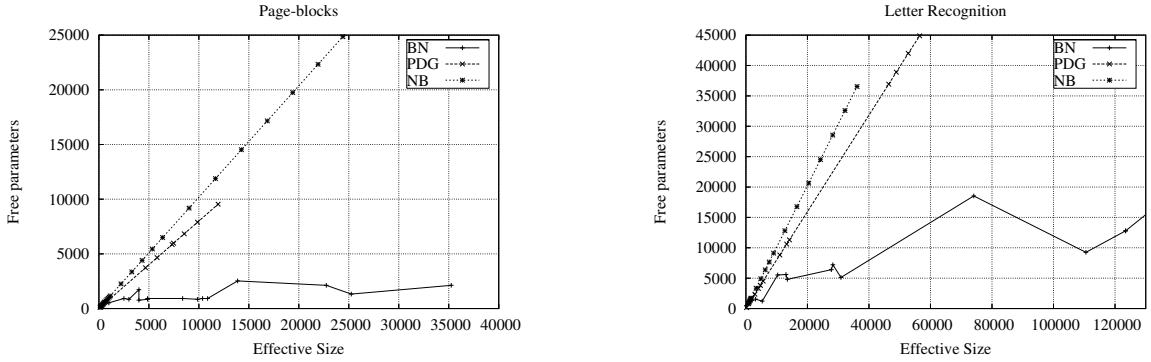
the others. If only one or none of these observations are made, this is indicated by a dash $-$. Table 5.7 lists SL-coordinates for three models from each language for each dataset: 1) the model attaining maximal likelihood over $\mathcal{D}_B$, 2) the model attaining maximal BIC score, and 3) the model attaining maximal AIC score. Also, Table 5.7 lists the number of models learned and execution times for learning procedures.

From Table 5.6 we see that the BN language is the language most frequently dominated on $\mathcal{D}_A$. As previously observed, BN models do not often gain much in accuracy by an increase in complexity. By complexity we here refer to the effective size, which for BN models is not in linear relation to the number of free independent parameters in the model. Therefore, the observation that BN models do not capitalise on increased complexity, is probably fully explained by the fact that for any two BN models of different effective size, the number of free parameters (and therefore the ability to represent the empirical distribution of the data) may be the almost the same. In Figure 5.7 we investigate the relationship between effective and representational size of models learned from real data. For models learned from Page-blocks data we plot representational size vs. effective size for BN and PDG models in Figure 5.7(a) and similar plots for models learned from the Letter Recognition data in Figure 5.7(b). We clearly see that increased effective size increases the representational ability of PDG models at a rate that is approximately linear. For BN models, the relationship is sub-linear or linear with at a very low rate. The important observation from Figure 5.7 is that increased complexity does not necessarily buy much representational power for the BN model.

The sub-linear relationship between effective size and representational size also explains the low propensity of BN models to overfit $\mathcal{D}_A$. This then also explains why we do not observe BN models being dominated consistently for likelihood values over $\mathcal{D}_B$ (see Table 5.6).

From the summaries given in Table 5.7 we observe concerning maximal likelihood over $\mathcal{D}_B$, a BN model is most frequently the model with highest value, which is not surprising given the above discussion on consistent dominance of BN models. However, the superior accuracy of the BN models over $\mathcal{D}_B$ when comparing to NB and PDG models, is often accompanied by a huge effective size. For none of the experiments do we observe a dominating model in terms of both effective size and likelihood.

Comparing the models selected by maximal likelihood value over $\mathcal{D}_B$ to the models selected by the BIC and AIC criterions (columns M_{BIC} and M_{AIC} in Table 5.7), we see that BIC consistently selects models of lower complexity than the $M_{\max(L(\mathcal{D}_B))}$ models. AIC often selects models that are more complex than the $M_{\max(L(\mathcal{D}_B))}$ models. The model selected by



(a) (b)

Figure 5.7. Plots showing the $size_{rep}$ vs. $size_{eff}$ of PDG models and BN models learned from the Pageblock data (a) and Letter Recognition data (b).

		$M_{max(L(\mathcal{D}_B))}$			M_{BIC}			M_{AIC}			#models	Time
		$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	learned	(seconds)
Page-blocks	BN	13875	-13.239	-13.362	4875	-13.788	-13.924	35250	-13.228	-13.373	15	9045.77
	PDG	11925	-11.806	-11.998	745	-14.686	-14.782	4660	-12.395	-12.582	9	2201.89
	NB	6490	-12.331	-12.916	825	-14.801	-14.726	3355	-12.944	-13.575	28	181248.10
Letter Recognition	BN	141675	-23.477	-23.872	30895	-25.971	-25.945	141675	-23.477	-23.872	16	38347.25
	PDG	14138	-24.607	-28.046	5596	-27.434	-28.643	48888	-20.899	-28.956	13	48360.62
	NB	20640	-23.856	-24.655	9116	-26.005	-26.260	36550	-22.04	-24.966	21	62537.11
Landsat	BN	5451000	-35.509	-36.000	678875	-36.210	-36.535	5451000	-35.509	-36.000	27	84707.71
	PDG	7418	-44.524	-50.465	5151	-47.504	-51.688	24098	-37.595	-52.936	12	15215.58
	NB	5850	-40.643	-42.417	3000	-42.585	-44.065	17550	-34.822	-48.708	30	280690.76
Adult	BN	62270	-15.993	-16.182	2286	-16.249	-16.357	16586	-16.036	-16.198	15	46983.06
	PDG	4208	-16.179	-16.695	909	-16.678	-16.796	7071	-15.967	-16.871	11	30331.27
	NB	19205	-15.676	-17.677	1610	-16.731	-18.286	13570	-15.784	-17.703	29	279967.07
King, Rook vs. King	BN	6912	-16.700	-16.880	5184	-16.982	-17.096	6912	-16.700	-16.881	18	20094.70
	PDG	18930	-15.691	-16.783	1450	-16.947	-17.112	17010	-15.687	-16.841	14	13524.50
	NB	32032	-15.370	-17.279	2080	-16.981	-19.061	15288	-15.794	-17.64	39	279925.69
Abalone	BN	4575	-10.400	-10.439	190	-10.915	-10.851	4575	-10.400	-10.446	15	5825.32
	PDG	694	-10.467	-10.883	694	-10.467	-10.883	1416	-10.077	-11.130	17	1421.85
	NB	1365	-10.139	-10.463	315	-10.591	-10.753	700	-10.297	-10.517	56	263656.95
Poisonous Mushroom	BN	208333	-13.924	-13.883	5531	-14.898	-14.771	85477	-13.937	-13.931	23	34847.75
	PDG	6560	-13.768	-13.95	2811	-13.956	-14.038	2811	-13.956	-14.038	14	2257.60
	NB	28518	-12.955	-14.421	1940	-14.655	-20.839	7760	-13.107	-19.747	43	280686.46

Table 5.7. SL-coordinates for a few important models from each language. SL-coordinates for the model attaining maximal likelihood value over $\mathcal{D}_B$ (column $M_{max(L(\mathcal{D}_B))}$), the model attaining maximal BIC score (column M_{BIC}) and maximal AIC score (column M_{AIC}) are listed for each language. The column "#models learned" lists the number of models learned for each language, and the last column lists execution times for each learning procedure.

AIC is often closer to the model attaining maximal likelihood over $\mathcal{D}_B$ than that selected by BIC. The fact that AIC penalises less than BIC for increased complexity is clear from the definitions of BIC and AIC (see Section 4.1).

We can sum up the observations discussed above by stating that a surprisingly similar performance of the three PGM languages is observed, while the BN language exhibits the most stable performance with less propensity for overfitting the training data.

5.4 Empirical Analyses

For the analyses of SL-curves reported in the previous sections, the effective size has been used as a theoretical measure of efficiency. The effective size for PGM M is a (model) specific parameter for which general belief updating is computable in linear time. This means that conclusions drawn from comparison of effective size are only valid up to a linear factor. In this section, we report on experiments measuring the absolute practical complexity including the linear factor.

The experimental setup is as follows:

- For belief updating in PDG models we use the `copmuteIflOfl` procedure of Algorithm 3.6 after inserting evidence. The probability of the evidence is then computed by Equation 3.51. The `computeIflOfl` procedure was implemented in the Java language using standard libraries of JDK v. 1.5[8] and the Weka package for basic data handling routines (Witten and Frank, 2005).

- For BN and NB models we used the Hugin[9] inference engine through the Hugin Java API. The Hugin inference engine is a C implementation of a clique tree propagation algorithm, see (Jensen et al., 1990b,a; Andersen et al., 1989). It is a highly optimised implementation and frequently recommended as one of the best tools for probabilistic inference (Cowell et al., 1999; Jensen, 2001; Castillo et al., 1997).

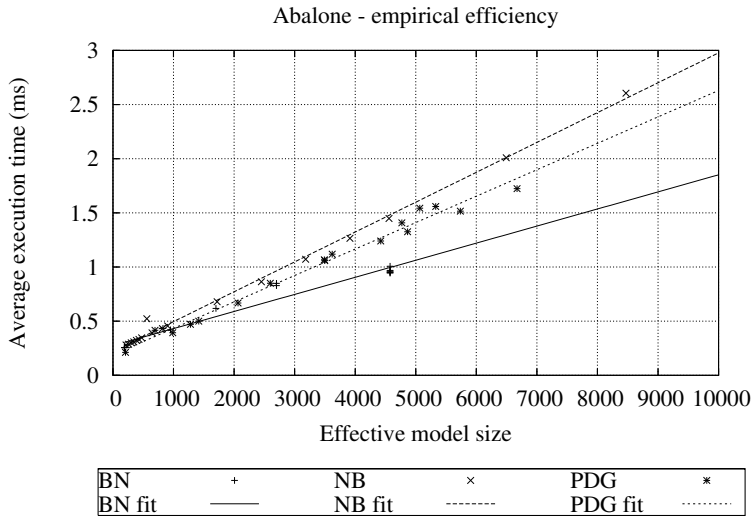
- For computing averages, we generated 1000 random queries from each set of test data. We used a fixed size for the random sets of variables: $|\mathbf{Q}| = 4$ and $|\mathbf{E}| = 3$. The procedure for generating queries were described in Section 5.1.1.

- The experiments where all performed on a Sun Fire280R, 900MHz SPARC CPU architecture with 4GB of main memory running Solaris 9.

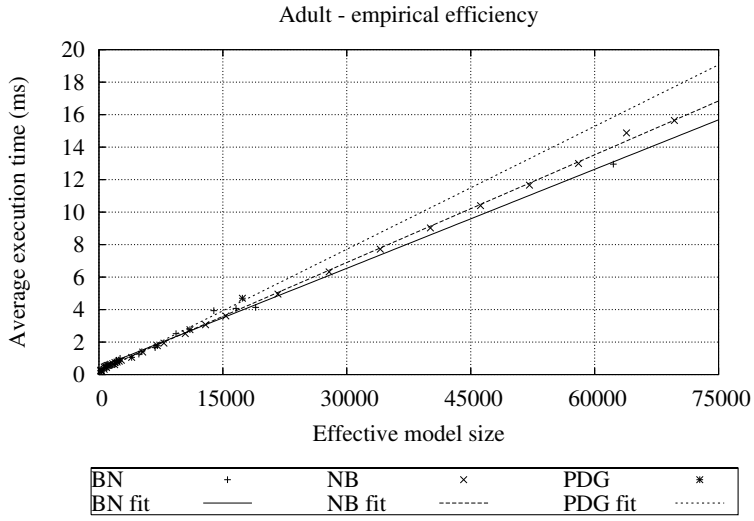
5.4.1 Discussion of Results

To extract the linear factor between effective size and the actual execution time of belief updating, we plot measured execution time against effective size. Examples can be seen in Figure 5.8(a) for models learned from Abalone data and Figure 5.8(b) for models learned from the Adult data. In addition, we plot the linear expression $y = \alpha \cdot size_{\mathit{eff}} + \beta$ where

[8] `http://java.sun.com/`
[9] `http://www.hugin.com/`



(a)



(b)

Figure 5.8. Average execution time of performing belief updating vs. effective size for models learned (a) the Abalone dataset and (b) the Adult dataset.

y is the average execution time and α and β are fitted through a standard least squares Marquardt-Levenberg fitting procedure.[10] This fitting procedure fits parameters α and β such that the sum of squared errors over data instances are minimised. The fitted α-values for all experiments are listed in Table 5.8 together with an asymptotic standard error. We observe that differences are quite limited, except for the Letter and Landsat data-bases. Here the linear factor for PDGs are much larger than for BNs and NBs. That BNs and NBs are

[10]We used the `fit` command in the gnuplot system (Williams and Kelley, 2004).

	BN		NB		PDG	
Page-blocks	0.22	$\pm 7.10 \cdot 10^{-6}$	0.30	$\pm 2.25 \cdot 10^{-6}$	0.27	$\pm 4.29 \cdot 10^{-6}$
Letter Recognition	0.26	$\pm 5.65 \cdot 10^{-6}$	0.26	$\pm 5.92 \cdot 10^{-6}$	0.98	$\pm 3.59 \cdot 10^{-5}$
Adult	0.20	$\pm 3.99 \cdot 10^{-6}$	0.22	$\pm 1.07 \cdot 10^{-6}$	0.25	$\pm 5.84 \cdot 10^{-6}$
King, Rook v. King	0.22	$\pm 3.09 \cdot 10^{-6}$	0.23	$\pm 6.91 \cdot 10^{-7}$	0.16	$\pm 3.63 \cdot 10^{-6}$
Abalone	0.16	$\pm 9.55 \cdot 10^{-6}$	0.28	$\pm 4.25 \cdot 10^{-6}$	0.24	$\pm 7.42 \cdot 10^{-6}$
Poisonous Mushroom	0.25	$\pm 5.36 \cdot 10^{-6}$	0.26	$\pm 6.89 \cdot 10^{-7}$	0.20	$\pm 1.04 \cdot 10^{-5}$
Landsat	0.24	$\pm 2.90 \cdot 10^{-6}$	0.30	$\pm 4.18 \cdot 10^{-7}$	0.64	$\pm 3.62 \cdot 10^{-5}$

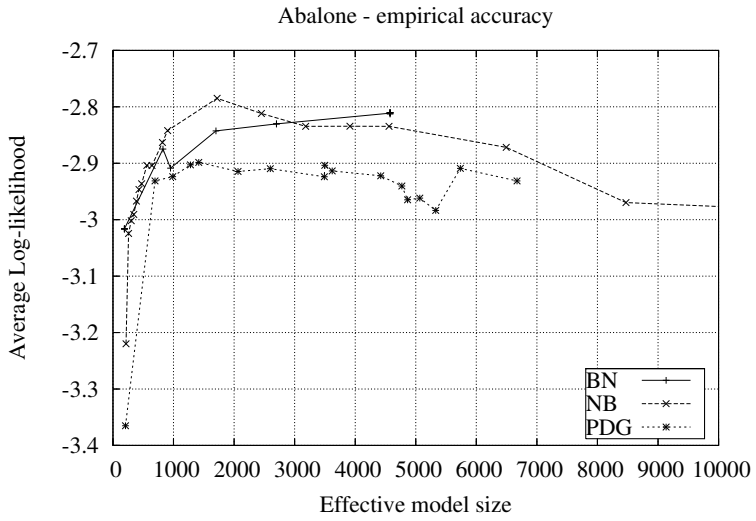
Table 5.8. The slope α (times 10^3) of the line $y = \alpha \cdot size_{\mathit{eff}} + \beta$, where y is the measured execution time and α, β are fitted by the standard least squares fitting procedure implemented by the `fit` command in the gnuplot system (see (Williams and Kelley, 2004)), $\pm$ asymptotic standard error of α.

always very close is of course not surprising considering that exactly the same belief updating procedure is used. We also observe that no single language is consistently better or worse than the others. Considering the standard errors, we observe that there exists some discrepancy between the different languages. In fact, for the datasets Landsat and Poisonous Mushroom the discrepancy is on the order a factor 100 (between NBs and PDGs) and a factor 10 (between BNs and PDGs). We believe that this is due irremovable measurement error stemming from different factors such as garbage collection in the Java Virtual Machine.

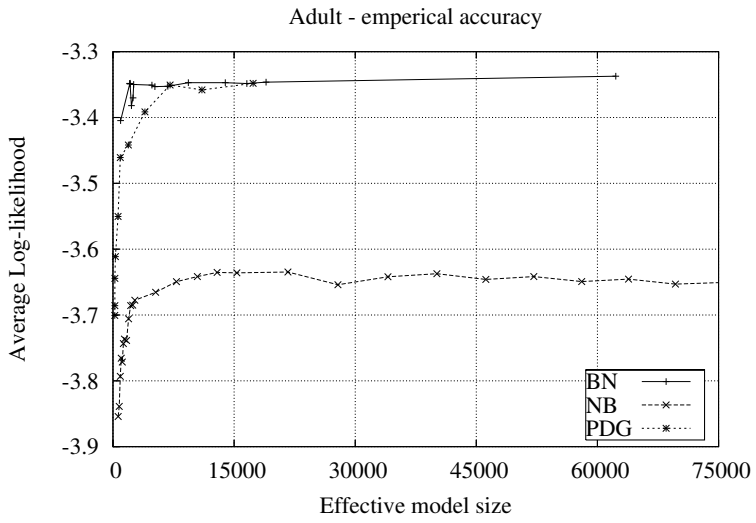
Figures 5.9(a) and 5.9(b) shows plots of the empirical accuracy measured as the averaged log-likelihood of random queries. Comparing the plot of empirical accuracy over queries generated from the Adult data in Figure 5.9(b) against effective size to the corresponding plot using the full log-likelihood of Adult test data in Figure 5.5(b), we see that PDG models are more competitive when measuring accuracy empirically. A similar observation is made for Abalone data from comparisons of empirical accuracy over the Abalone data (Fig. 5.9(a)) and the corresponding plot using the full log-likelihood over test data (Fig. 5.6(b)).

The values in Table 5.8 gives an estimate on the linear factor associated with the complexity of general belief updating. While the differences are relatively small, we are still interested in the stability of conclusions drawn from the effective size in the light of the actually measured execution times. That is, if changing the measure of efficiency from effective size to average execution time will have any impact on model selection. Therefore, we list in Table 5.9 for each dataset and each language, two models. First, the models attaining maximal empirical accuracy are listed under $M_{max(\log P(\mathbf{Q}|\mathbf{E}))}$. Second, the models attaining maximal log-likelihood value over $\mathcal{D}_B$ are listed under $M_{max(L(\mathcal{D}_B))}$. The second set of selected models where previously listed in Table 5.7, and we here augment the SL-coordinates for the models with the average execution time.

First, we consider the models $M_{max(L(\mathcal{D}_B))}$, and the language that would be preferred w.r.t. efficiency. For each dataset, the ordering of the models selected from the three language w.r.t. average execution time is the same as ordering w.r.t. effective size, except for Letter data. Here the selected PDG model has lowest effective size while the selected NB model has lowest average execution time. It should be noted that the NB and PDG models do not differ significantly in neither effective size nor average execution time, so even though the relative



(a)



(b)

Figure 5.9. Empirical average accuracy vs. effective size for (a) models learned from Abalone data and (b) models learned from Adult data. Log likelihoods are per query instance.

		$M_{max(\log P(\mathbf{Q}\|\mathbf{E}))}$			$M_{max(L(\mathcal{D}_B))}$		
		$size_{eff}$	$\log P(\mathbf{Q}\|\mathbf{E})$	time (ms)	$size_{eff}$	$L(\mathcal{D}_B)$	time (ms)
Page-blocks	BN	35250	-3.134	7.961	13875	-13.362	3.434
	PDG	11925	-2.774	3.392	11925	-11.998	3.392
	NB	6490	-2.994	2.074	6490	-12.916	2.073
Letter	BN	141675	-4.705	44.435	141675	-23.872	44.435
	PDG	56485	-4.645	54.567	14138	-28.046	5.954
	NB	24582	-4.668	6.866	20640	-24.655	5.666
Landsat	BN	5451000	-3.520	3162.610	5451000	-36.000	3162.610
	PDG	130247	-3.953	125.973	7418	-50.465	2.868
	NB	7440	-3.644	2.963	5850	-42.417	1.879
Adult	BN	62270	-3.337	12.955	62270	-16.182	12.955
	PDG	17359	-3.348	4.697	4208	-16.695	1.056
	NB	21672	-3.635	4.956	19205	-17.677	4.955
King, Rook v.	BN	6912	-6.461	1.745	6912	-16.880	1.745
King	PDG	18930	-6.441	3.336	18930	-16.783	3.336
	NB	35786	-6.616	8.382	32032	-17.279	7.712
Abalone	BN	4575	-2.811	0.963	4575	-10.439	1.223
	PDG	1416	-2.899	0.509	694	-10.883	0.386
	NB	1720	-2.785	0.698	1365	-10.463	0.428
Poisonous	BN	85477	-1.909	24.226	208333	-13.883	61.458
Mushroom	PDG	16854	-1.883	3.776	6560	-13.95	1.792
	NB	35105	-1.883	9.517	28518	-14.421	8.058

Table 5.9. Columns $M_{max(\log P(\mathbf{Q}|\mathbf{E}))}$ contains the models of maximum local accuracy, that is, average log probability of 1000 random queries. Columns $M_{max(L(\mathcal{D}_B))}$ contains models selected maximal likelihood values over $\mathcal{D}_B$ and associated effective size and average execution time.

ordering is changed, the models have close to similar average execution time.

Next, consider the models $M_{max(\log P(\mathbf{Q}|\mathbf{E}))}$ selected for maximal empirical accuracy. The ordering of the three languages w.r.t. maximal empirical accuracy is the same as the ordering of languages w.r.t. maximal log-likelihood over $\mathcal{D}_B$ for 4 of the 7 datasets. For the Letter, Abalone and Poisonous Mushroom datasets the BN language provides the most accurate model w.r.t. log-likelihood over $\mathcal{D}_B$, however when doing the comparison w.r.t. empirical accuracy the BN language no more provides the most accurate approximation.

One last observation, for the Landsat data, we observe that the huge relative differences in accuracy w.r.t. log-likelihood over $\mathcal{D}_B$ does not reemerge when we measure accuracy by log-likelihood over random queries.

One possible explanation for why orderings w.r.t. accuracy change when considering log-likelihood over random queries instead of log-likelihood over the full test data $\mathcal{D}_B$, could be the existence of a few unlikely data instances in $\mathcal{D}_B$. Then BN models will often provide a more smoothed model as they contain fewer free parameters than NB and PDG models, as shown for two examples by the plot in Figure 5.7. The likelihood of the more smoothed model will then not be as sensitive to a few rare instances in the data as the less smoothed models. However, when measuring accuracy by the log-likelihood over queries, this means only considering a marginal distribution over a subset of variables for every term in the sum of likelihoods. Therefore, the unlikely joint configurations may not be expressed in the subsets of variables used in the queries, and the less smoothed models prevails over the more smoothed models.

Lastly, the empirical measures of execution time for PDG models is encouraging as our prototype implementation performs competitively when compared to the Hugin inference engine.

5.4.2 Related Work

Lowd and Domingos (2005) performs an empirical comparison much like the one we have performed in this section, though only comparing BN and NB models. The measure of efficiency is based on computing only the joint conditional posterior of a subset of variables given some random evidence. This is particularly efficient in NB models as every variable not participating as a query variable or as evidence can immediately be removed from the computation. In this setting, Lowd and Domingos (2005) show that NB models exhibit superior efficiency to BN models, and that the accuracy of NB models is competitive with that offered by BN models. Our analysis shows that the computational efficiency of NB models does not extend to the inference task of belief updating, and concerning accuracy we are unable to proclaim any language the winner.

5.5 The Hybrid Learning Approach

The hybrid learning approach discussed in Section 4.6 combines BN learning and PDG learning. By using the clique tree (CT) of a BN model as the basis for a PDG structure,

we merge parameter nodes in the PDG structure with increasing aggressiveness (see Algorithm 4.18). In this way, we aim at constructing efficient PDG models without trading off accuracy, potentially improving on the efficiency of the original BN model. In this section, we evaluate the performance of PDG models learned using the hybrid approach. We use exactly the BN models learned from real data as discussed in Section 5.3.

Before going into a detailed analysis of the full experiment, we will analyse a single experiment in some detail.

Figures 5.10(a) and (b) show the result of the hybrid learning using a BN model learned from the Abalone data. By the rectangular point we mark the SL-coordinates of the BN model, and the points connected by the dashed line corresponds to SL-coordinates of the PDG models obtained by continued merging of nodes. That is, the rightmost point on the dashed line corresponds to the PDG model obtained without merging and collapsing of zero-inflow nodes. The rest of the points in the plot then each corresponds to the PDG model obtained by increasingly aggressive merge operations. The diamond marks the smallest PDG model that has higher or equal likelihood score over training data $\mathcal{D}_A$ compared to the original BN model. We denote this PDG model the "Best" PDG model.

From Figures 5.10(a) and (b), we see that the initial translation from CT to PDG results in an increase in effective size of the PDG model when comparing to the original effective size of the BN model indicated by the square. Also, an increase in likelihood is observed, which is explained by the fact that parameters are re-estimated for the PDG model after the structure has been constructed from the CT of the BN model. For the CT model, parameters come directly from the BN model and the CT therefore does not exploit the extended expressibility of more parameters. Parameters could have been re-estimated for the CT model, however, we use the more common approach of using parameters estimated in the BN model. Also, re-estimating parameters for the CT model obtained from a BN model would make our analysis less clear as learning has been performed only for the BN model and not the CT model.

The most interesting observation from Figures 5.10(a) and (b), is that accuracy does not deteriorate rapidly when the effective size is decreased by repeated merge operations. This shows us that the initial PDG model constructed from the CT model contains redundant parameter nodes that are not needed in the approximation offered by the model. This redundant complexity is then successfully identified and removed from the PDG by merge operations.

5.5.1 Discussion of Results

For each dataset and each BN model learned from the dataset, we summarise the important observations from three selected experiments. For each database, we have selected experiments using the following BN models:

1. the smallest effective size BN model,

2. the BN model that attains the highest likelihood value over $\mathcal{D}_B$ and

3. a BN model with an effective size in between the two other models.

For each of the 7 datasets, these 3 selected experiments are summarised in Table 5.10. Each experiment is summarised in form of the SL-coordinates of the original BN model, the SL-coordinates of the "Best" PDG model, and the relative improvement of the "Best" PDG model over the original BN model. Relative improvement for a value is calculated as:

$$\text{Relative Improvement} = \frac{\text{BN} \ - \ \text{Best PDG}}{\text{BN}}.$$

Please refer to Appendix A.4 for summaries of all experiments and all datasets.

The results summarised in Table 5.10 generally show that the hybrid approach (with few exceptions) successfully constructs PDG models that dominate the original BN models in SL-space, both when considering likelihood over $\mathcal{D}_A$ and $\mathcal{D}_B$.

The first experiment selected for each dataset is summarised in the first row within each block of three rows in Table 5.10. These are result of applying the hybrid approach to smallest effective size BN model that we learned for the given dataset. In these experiments we are not always successful in constructing PDG models that improve on the original BN model (Page-blocks and Letter Recognition being exceptions). This observation is not surprising, as the BN model that we try to improve is quite compact in the first place. Therefore, not many superfluous parameter nodes exists in the PDG representation of the CT model, and the merge operations are not able improve on the size by removal of nodes without reducing accuracy.

For the second row experiments we use a BN model of an effective size in between the optimal BN model (w.r.t. likelihood over $\mathcal{D}_B$) and the simplest (smallest effective size). Here we more consistently observe an improvement by the best PDG model over the initial BN model. For the Adult and Letter Recognition datasets we observe a small degradation in likelihood (2% and 0.6% when measure over $\mathcal{D}_B$). For the Landsat dataset, however, the degradation in likelihood over $\mathcal{D}_B$ is severe (40.2%). Figure 5.11 shows a detailed plot of this particular experiment, and we see that a few merges results in a major decrease in likelihood over $\mathcal{D}_B$ (Figure 5.11(b)). This particular dataset previously has proved difficult for direct learning of PDG models (Section 5.3) and we are therefore not surprised to find this particular dataset causing problems for the hybrid approach also. PDG models seem to fail in smoothing the representation sufficiently and instead captures the empirical distribution of $\mathcal{D}_A$ too closely, yielding the poor generalisation power to the instances of $\mathcal{D}_B$.

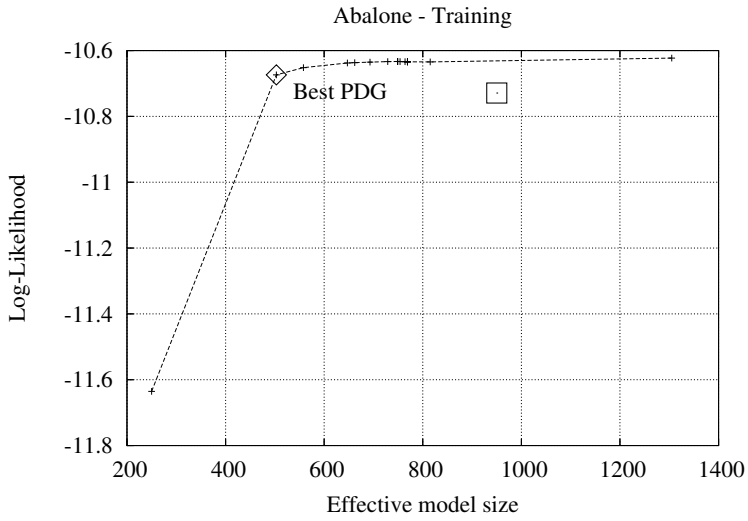
Figure 5.11 on page 139 shows plots of the SL coordinates of all the PDG models visited in this experiment. From this we see that the collapsing of zero inflow nodes reduces size dramatically, and the result is a PDG model that already scores worse on $\mathcal{D}_B$ compared to the original BN model (marked by the square). With the collapsing of zero inflow nodes, the PDG model keeps only the parameters necessary for capturing the distribution of $\mathcal{D}_A$. The huge joint state-space of the observable variables of the Landsat data ($\approx 10^{25}$) in combination with the small size of the dataset (6435), the empirical distribution is not expected to provide a good estimate of the generative distribution. However, by collapsing zero inflow nodes we decrease the models ability to smooth over $\mathcal{D}_B$ by removing (amongst others) parameters that are only reached by instances of $\mathcal{D}_B$.

If we investigate the measured execution times, we observe some quite unexpected times especially for Abalone, King Rook vs. King and Poisonous Mushroom. Here, the simplest models of smallest effective size also had the longest execution times. When profiling the implementations in detail we found that the extra time was used on tuning the smoothing factor by the `tuneSmooth` procedure (Alg. 4.2). Specifically, when the optimal (unknown) smoothing value was relatively large, the initial values used in the search was quite poorly chosen by an internal tuning procedure. This then yields a large number of cross validations, each of which includes expensive data access. This problem is implementation specific issue, and as we did not experience problems for examples of a more typical effective size, we will not spend more time on this.

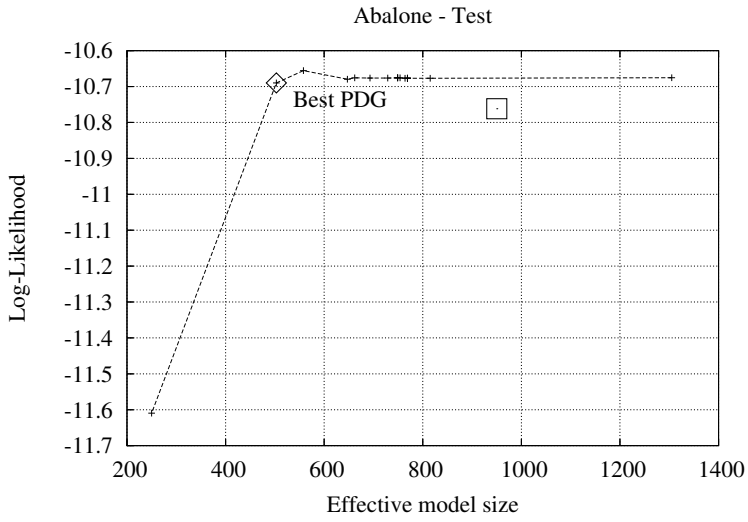
We also applied the hybrid learning approach to BN models learned from the synthetic datasets, the results are summarised in Table 5.11. The results are different from the results summarised in Table 5.10 in that the relative improvements in effective size are smaller. One obvious explanation is that the BN models that in this experiment set of experiments has smaller effective size (that is, smaller CT models) and therefore the potential size improvement is smaller. For the larger models (especially for Hailfinder and Rnd15), we still observe a significant improvement in size and at the prize of a fairly limited degradation in log-likelihood. This is especially pleasing to observe as these two datasets previously caused problems for direct learning of PDG models (see Section 5.2).

When inspecting the execution times, we again observe problems with the smaller models. This is similar to what we already observed in the summary in Table 5.10, and again originates from a bad choice of initial values for the `tuneSmooth` procedure.

The hybrid approach to learning PDG models has proven to be a feasible approach to obtaining good PDG models. We typically observe a significant reduction in effective size when comparing the best PDG model to the original BN model at a limited cost in accuracy and generalisation to new cases. The only experiments that fail in this respect are the ones using BN models learned from the Landsat data. This is not entirely unexpected, as the Landsat data proved to be one of the harder problems for the PDG learning algorithm as previously discussed in Section 5.3. The dataset on which the hybrid approach is most successful is the Poisonous Mushroom data. Again, this is not surprising when remembering that the PDG learner was also observed to be most successful on exactly this dataset (see Section 5.3).



(a)

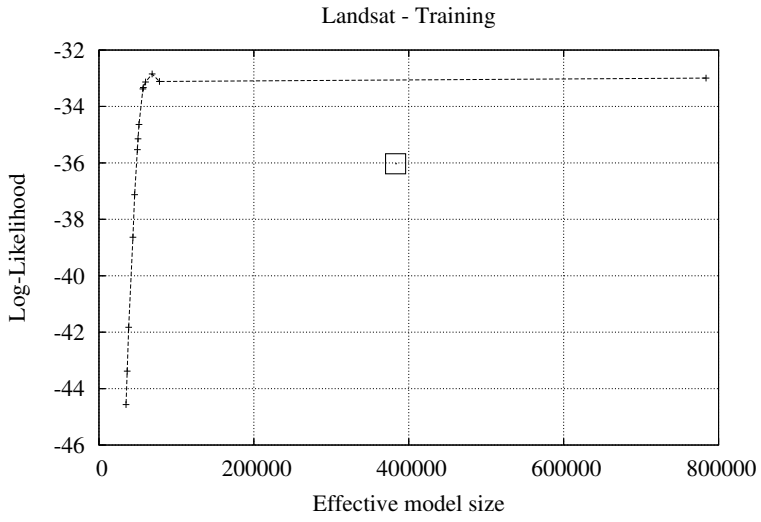


(b)

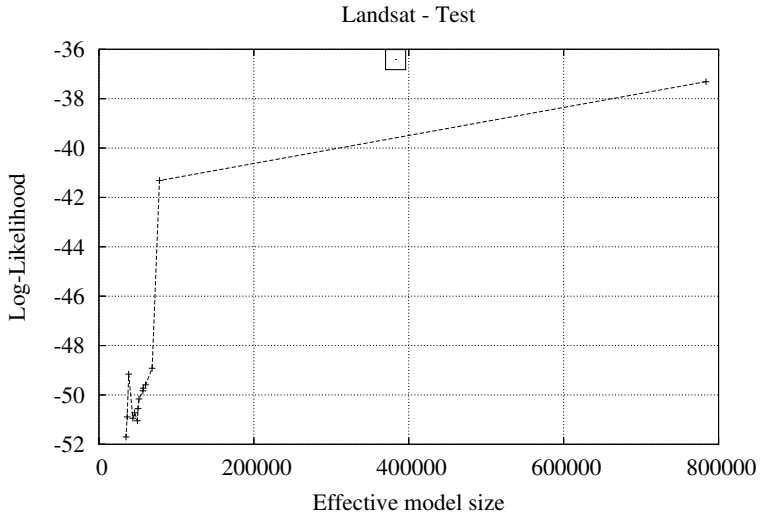
Figure 5.10. PDGs learned from JTs obtained from BNs learned from Abalone data. The square marks the SL coordinates of the BN model, and the rightmost point in the plot marks the PDG constructed from the JT of the BN. Points connected by the dashed line corresponds to SL coordinates obtained for the PDG model after increasingly aggressive merging of nodes. The square point marks the SL coordinates of the BN model and the diamond marks the best PDG model obtained in the experiment. Log likelihoods are per data instance.

	BN			Best PDG			Relative Difference			Time
	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
Page-blocks	1000	-14.512	-14.607	675	-14.363	-14.482	0.325	0.010	0.009	31.0
	8375	-13.734	-13.771	3935	-13.210	-13.286	0.53	0.038	0.035	74.9
	13875	-13.239	-13.362	8045	-12.502	-12.577	0.42	0.056	0.059	51.1
Abalone	190	-10.915	-10.851	270	-10.904	-10.837	-0.421	0.001	0.001	66.9
	2700	-10.445	-10.475	758	-10.362	-10.423	0.719	0.008	0.005	38.5
	4575	-10.401	-10.442	1003	-10.346	-10.487	0.781	0.005	-0.004	59.8
Adult	950	-16.388	-16.473	965	-16.380	-16.468	-0.016	0.000	0.000	1654.9
	18966	-16.016	-16.190	14316	-16.011	-16.290	0.245	0.000	-0.006	2038.6
	62270	-15.993	-16.182	31564	-15.860	-16.440	0.493	0.008	-0.016	2068.2
King, Rook vs. King	120	-18.413	-18.426	124	-18.412	-18.426	-0.033	0.000	0.000	1461.8
	3744	-16.957	-17.092	2906	-16.919	-17.065	0.224	0.002	0.002	1035.9
	6912	-16.700	-16.880	5782	-16.633	-16.835	0.163	0.004	0.003	1053.3
Landsat	910	-43.622	-43.705	1210	-43.605	-43.703	-0.330	0.000	0.000	266.9
	383000	-36.030	-36.419	49583	-35.532	-51.044	0.871	0.014	-0.402	463.4
	760375	-35.989	-36.304	64796	-35.836	-55.776	0.915	0.004	-0.536	1413.4
Letter Recognition	3119	-28.759	-28.609	2683	-28.372	-28.275	0.140	0.013	0.012	2049.2
	74075	-24.037	-24.311	19660	-23.886	-24.809	0.735	0.006	-0.02	2679.5
	141675	-23.477	-23.872	62494	-23.317	-25.254	0.559	0.007	-0.058	2714.3
Poisinous Mushroom	121	-32.332	-32.283	123	-32.332	-32.283	-0.017	0.000	0.000	1737.4
	114741	-14.281	-14.194	2212	-13.747	-13.772	0.981	0.037	0.030	113.5
	208333	-13.924	-13.883	2010	-13.564	-13.560	0.990	0.026	0.023	141.4

Table 5.10. Summary of the hybrid approach to learning PDG models. For each dataset three experiments have been selected (see Section 5.5.1 for details). Each row then corresponds to one experiment. The three columns with headline 'BN' shows SL coordinates for the initial BN model; the three columns with headline 'PDG' shows SL coordinates for the best PDG models; and the three columns with headline 'Relative Difference' shows the relative improvement over BN SL coordinates by the best PDG SL coordinates. The last column contains execution time in seconds, not including learning of the original BN model, but including the CT construction and translation from CT to equivalent PDG.



(a)



(b)

Figure 5.11. SL coordinates for all PDG models visited in experiment 3 of the Landsat data. The SL coordinates of the BN model is marked by the square. This experiment is summarised in the second row of the Landsat block in Table 5.10 on the preceding page. Log likelihoods are per data instance.

	BN			Best PDG			Relative Difference			Time
	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
Alarm	142	-18.342	-18.618	171	-18.500	-18.431	-0.204	-0.009	0.01	2916.6
	335	-13.847	-14.047	538	-13.937	-13.848	-0.606	-0.006	0.014	1881.8
	624	-13.666	-13.868	858	-13.741	-13.669	-0.375	-0.005	0.014	1688.9
Hailfinder	1628	-71.678	-71.750	1957	-71.618	-71.698	-0.202	0.001	0.001	232.5
	4820	-70.842	-70.989	4778	-70.787	-71.027	0.009	0.001	-0.001	307.1
	8472	-70.804	-70.976	5691	-70.761	-71.162	0.328	0.001	-0.003	335.5
NB10	300	-21.222	-21.306	463	-21.220	-21.305	-0.543	0.000	0.000	1022.2
	2772	-20.842	-21.059	3550	-20.758	-21.092	-0.281	0.004	-0.002	1018.0
	5220	-20.687	-21.051	9074	-20.564	-21.166	-0.738	0.006	-0.005	1111.5
NB20	179	-20.172	-20.170	224	-20.151	-20.155	-0.251	0.001	0.001	1018.0
	1296	-19.895	-20.013	1425	-19.889	-20.010	-0.100	0.000	0.000	1002.3
	4320	-19.819	-19.991	4193	-19.810	-20.083	0.029	0.000	-0.005	1050.5
Rnd15	42	-16.229	-16.256	46	-16.229	-16.256	-0.095	0.000	0.000	1671.8
	657	-14.872	-14.913	577	-14.865	-14.908	0.122	0.000	0.000	947.5
	1080	-14.837	-14.903	607	-14.833	-14.909	0.438	0.000	0.000	964.5
Rnd20	51	-20.977	-20.991	56	-20.977	-20.991	-0.098	0.000	0.000	2847.1
	224	-18.580	-18.570	233	-18.515	-18.505	-0.040	0.003	0.004	1786.3
	674	-18.079	-18.129	669	-18.077	-18.134	0.007	0.000	0.000	1683.1

Table 5.11. Summary of the hybrid approach to learning PDG models from synthetic. The models used in the experiments reported here where selected in the same way as those reported for real-data, see Table 5.10.

Chapter 6

Conclusion

In this dissertation, we have addressed aspects of unsupervised learning of PGMs. We have proposed algorithms for learning three different PGM languages, and performed a comparative analysis of the tradeoffs offered by different the different models we are able to learn.

The task of learning Bayesian Network models from data can be viewed as the task of recovering the true model representing the generative distribution, the strong assumption of data being samples of a DAG-faithful distribution has to be satisfied. Algorithms like the SGS algorithm (Spirtes et al., 2000) and the GES algorithm Meek (1997) exhibit asymptotic optimality when learning from such data sampled from DAG-faithful distributions. However, in practice the assumption of DAG faithfulness is unrealistic, and even when it is satisfied, the available sample may be too small, yielding suboptimal results for the asymptotically optimal procedures. In Section 4.3, we presented the `KES` procedure for learning Bayesian Network models. The `KES` procedure generalises the greedy search employed by the GES procedure (Meek, 1997) by offering a parameterised tradeoff between greediness and randomness in the search. `KES` maintains the asymptotic optimality of the GES procedure, while often avoiding low quality suboptimal models. In Section 4.3.4, we reported on initial experiments with the `KES` procedure. By multiple restarts of `KES` using a non-greedy setting ($k < 1.0$) we showed that the number of local inclusion optimal models that exists for a limited data-sample of a DAG-faithful distribution can be huge. While the greedy search of GES is inherently deterministic, the introduction of the stochastic search in `KES` broadens the field of vision of the search. The model recovered by the greedy search can be suboptimal when the sample is too small, and introducing a broader stochastic search can result in better models (e.g., see Figure 4.8(b)). The importance of investigating more local optima becomes very clear for distributions that are not DAG-faithful. Such data may misguide the greedy search to a suboptimal model, see Figure 4.6. In most realistic settings, data will be limited and, in addition, DAG-faithfulness will be violated. Therefore, the practical applicability of greedy heuristics are limited and stochastic searches are to be preferred.

Jaeger (2004) introduces the language of Probabilistic Decision Graphs (PDGs) as a general representation framework for joint probability distributions. PDGs can capitalise on the existence of CSI relations in providing a compact and computationally efficient representation. The computational structure used for general inference and belief updating is the PDG

representation itself, and no extra compilation step is needed. In Section 4.5, we present a heuristic procedure for learning PDG models from data. The procedure is composed of two conceptually disjoint phases. First, we induce a forest structure over the domain of variables. Second, we optimise a PDG structure over this variable forest. We use local split and merge operations, and for guiding the application of these operators we use both heuristic and exact measures of score improvement. In Section 4.5.2, we perform preliminary tests that demonstrates the ability of our proposed procedure to recover PDG models from data that offer accurate and efficient approximations of the generative distribution.

In Section 4.6 we proposed a procedure for learning PDG models from Clique Tree (CT) representations. By using CT representations obtained from a learned BN model, we combine BN learning and PDG learning in a hybrid approach. In this way, we provide a PDG representation that is equivalent to the CT representation in that it can represent the same set of joint probability distributions. In addition, by using data we optimise the efficiency of the PDG representation by estimating parameters and then removing redundant nodes by merging. In this way, we exploit CSI relations to achieve a more compact representation.

In Chapter 5 we performed a comparative analysis of the performance of BN, PDG and Naïve Bayes (NB) models, when learned from data. Our main goal was to evaluate the performance of the different model languages when models are learned from data. In this analysis, we both used synthetic data sampled from distributions represented by PGMs and real world data. In our comparison, we wanted to emphasise the computational efficiency as a main factor of comparison. We considered the task of probabilistic belief updating as the main computational task for PGMs and, therefore, identified for each modeling language its effective size as a model specific parameter in which belief updating is computable in linear time. This enabled us to perform a fair cross-language comparison of (theoretical) computational complexity. Concerning the accuracy of the approximation offered by models, we used the log-likelihood of a separate test-dataset $\mathcal{D}_B$. These two measures were combined in SL-curves, and we used such plots as the basis of one part of the analysis.

First, we analysed SL-curves of learning from synthetic data in Section 5.2. The analysis showed some expected and some unexpected outcomes. BN models and NB models both proved superior when exposed to learning from data sampled from the given models, respective languages, which was also what we expected to observe. For PDG models, we experienced some problems in learning from randomly generated PDG models, where instead BN models proved to offer more accurate approximations at a relatively low cost in effective size. This was somewhat unexpected, but it can be explained as another effect of the "Achilles heel" of our `LearnPDGs` procedure, namely the initial induction of a underlying variable forest. In the initial experiments of the procedure, we found that learning a good structure was not an easy task and suboptimal forests often had a significant impact on the PDG models that we actually learned (see Section 4.5.2).

Second, we analysed SL-curves of learning from real data in Section 5.3. One major result of the analysis was the observation that BN models are less prone to overfitting the training data than both BN and NBE models. We explained this observation by the fact that BN models typically have much fewer free parameters than both NB and PDG models of similar

effective size. For both PDG and NB models, there is a linear relationship between effective and representational size (given a fixed variable forest for the PDG). For BN models, no such trivial relationship between representational and effective size exists, but the observed relationship is typically sub-linear. Consequently, BN models do not gain representational power at the same rate as NB and PDG models when effective size is increased and we, therefore, observe overfitting at a lower rate for BN models. That being said, the analysis was unable to identify a clear winner among the three different languages, and results are very mixed over the different datasets.

Third, in Section 5.4 we performed an empirical analysis of computational efficiency by measuring execution times on randomly generated queries. We used our own prototype implementation of the *inflow/outflow* procedure for general belief updating in PDG models and for NB and BN models we used the Hugin[1] inference engine that implements a variation of the general CT algorithm for exact inference. In the results of these experiments, we were first of all pleased to observe that our prototype implementation of PDG inference was not completely incomparable to the state-of-the-art implementation of the Hugin inference engine. Next, we found that the conclusions drawn from using effective size as a measure of efficiency were mostly stable. That is, the ordering of language w.r.t. efficiency did not change by changing the measure of efficiency from effective size to average measured execution time. Next we considered the average log probability of randomly generated queries as an empirical measure of accuracy. Also here we found that the conclusions drawn from using the global measure of log-likelihood of data were mostly stable. However, for one example we found that a relatively large difference in log-likelihood of data between the three models was dramatically reduced when changing to the local measure of log probability of random queries. This observation can be explained by the existence of a few rare cases in the training data, that only contribute (negatively) to the computations of the global measure of accuracy. For the local measure using randomly generated queries, such extremely rare joint configurations are not sampled.

Finally, in Section 5.5 we analyse results of employing the hybrid approach to learning PDG models from CT representations compiled from learned BN models. Mostly, the experiments demonstrates the ability of the hybrid approach for learning PDG models that when compare to the original CT model offers a dramatic reduction in effective size without trading off accuracy. In this way PDG models may offer a more efficient computational structure for BN models than the more traditional CT algorithms. Compared to the related approach of compiling Arithmetic Circuits (ACs) from BN models by Darwiche (2002), our current proposal for hybrid learning necessitates an initial construction of a CT model from the BN model. Darwiche (2002) constructs ACs directly from the BN model and therefore avoids any potential problems with constructing the CT representation. On the other hand, the PDG language allows subsequent refinements in the form of merging of parameter-nodes and re-estimation of optimal parameters. The construction of AC representations from BN models by Darwiche (2002) exploits CSI relations that are identified in the parameterisation of the BN model. A key difference between that framework and our hybrid learning is then that we do not investigate the parameters of the CT model to exploit any CSI relations there may be.

[1] `http://www.hugin.com/`

Instead we turn to data and reestimate parameters and from here we exploit CSI relations indirectly by merging parameter-nodes. Our exploitation of CSI relations is therefore not very explicit, as we will consider any pair of nodes for merging, given the associated parameters are sufficiently close and without requiring that parameters match exactly. When working with real-world data we do not always expect CSI relations to manifest themselves clearly in data as noise may blur the image. Therefore, the merging of nodes seems a reasonable approach to optimising PDGs for size, and indeed in Section 5.5 we have shown good performance of PDG models learned by the hybrid approach when compared to the original CT model.

List of Symbols

$\mathscr{A}(\mathbf{Y})$	Partition generated by set if discrete random variables $\mathbf{Y}$, page 9.
$adj_G(X)$	Set of nodes adjacent to X in graph G, page 11.
$pa^*_G(\mathbf{A})$	Minimal ancestral set of nodes $\mathbf{A}$ in graph G, page 12.
$B^{\mathcal{D}}_G$	Parameterised BN model with DAG structure G and ML parameters θ estimated from data $\mathcal{D}$, page 58.
$ch_G(X)$	Set of children of node X in graph G, page 12.
$\mathbf{Y} \perp\!\!\!\perp \mathbf{U} \vert \mathbf{Z}[P]$	Conditional independence of $\mathbf{Y}$ and $\mathbf{U}$ given $\mathbf{Z}$ under distribution P, page 8.
$\mathcal{D}$	Sampled data, page 11.
$\mathcal{D}_A$	Part of data $\mathcal{D}$ used for training, page 52.
$\mathcal{D}_B$	Part of data $\mathcal{D}$ exclusively used for evaluation purposes, page 52.
$de_G(X)$	Set of descendants of node X in graph G, page 12.
$de^*_G(X)$	$de_G(X) \cup X$, page 12.
$size_{\mathit{eff}}(M)$	Effective size of model M, see (3.20) for BN models, (3.48) for PDG models and (3.59) for NB models, page 27.
$\mathscr{I}(\mathscr{B}, \mathscr{C})$	Partition generated by intersecting partitions $\mathscr{B}$ and $\mathscr{C}$, page 9.
$IB(M(G))$	Inclusion boundary of BN model $M(G)$, page 22.
$inc(\nu)$	Edges in a PDG structure incoming to node ν., page 85.
$\mathit{ifl}(\nu)$	Inflow of parameter-node ν, page 37.
G^m	Moral graph of graph G, page 12.
$nd_G(X)$	Set of non-descendants of node X in graph G, page 12.
ν	Parameter-node in PDG structure, page 29.
$\mathit{ofl}(\nu)$	Outflow of parameter-node ν, page 38.
$P(X)$	Probability distribution of random variable X, page 6.

p_l^ν	Element l in $\mathbf{p}^\nu$, page 29.
$\mathbf{p}^\nu$	Parameter vector for parameter-node ν, page 29.
$pa_G(X)$	Set of parents of node X in graph G, page 12.
$Path(\nu, \mathbf{Y})$	Set of joint instantiations of $\mathbf{Y}$ reaching ν, page 33.
$pa_G^*(X)$	Set of predecessors of node X in graph G, page 12.
$P(\mathbf{X})$	Joint probability distribution of random variables $\mathbf{X}$, page 7.
$reach(i, \mathbf{X})$	Parameter-node reached by $\mathbf{x}$ in V_i, page 32.
$R(\mathbf{X})$	The set of mutually exclusive joint states of discrete random variables $\mathbf{X}$, page 7.
$R(X)$	The set of mutually exclusive states of discrete random variable X, page 6.
G_A	Subgraph of graph G induced by subset of nodes A, page 11.
$succ(\nu_i, X_j, x_{i,h})$	The successor of parameter-node $\nu_i \in V_i$ for child variable X_j and for outgoing edge label-led $x_{i,h} \in R(X_i)$, page 29.
V_i	Set of parameter-nodes in PDG structure label-led with variable X_i, page 29.
$\mathbf{x}[\mathbf{Y}]$	The projection of joint state $\mathbf{x} \in R(\mathbf{X})$ onto a subset $\mathbf{Y} \subseteq \mathbf{X}$, page 7.
$\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \ldots$	Sets of random variables, page 7.
$\mathbf{x}, \mathbf{y}, \mathbf{z}, \ldots$	Joint states or of sets of random variables, page 7.
$X, Y, Z, \ldots$	Random variables, page 6.
$x, y, z, \ldots$	States of random variables, page 6.

Bibliography

Abellán, J., Gómez-Olmedo, M., and Moral, S. (2006). Some Variations on the PC Algorithm. In *Proceedings of the Third European Workshop on Probabilistic Graphical Models*, pages 1–8.

Abramson, B., Brown, J., Edwards, W., Murphy, A., and Winkler, R. (1996). Hailfinder: A Bayesian System for Forecasting Severe Weather. *International Journal of Forecasting*, 12:57–71.

Agresti, A. (1990). *Categorical Data Analysis*. John Wiley & Sons, Inc.

Akaike, H. (1974). A New Look at the Statistical Model Identification. *IEEE Transactions on Automatic Control*, 19(6):716–723.

Andersen, S. K., Olesen, K. G., Jensen, F. V., and Jensen, F. (1989). HUGIN - a Shell for Building Bayesian Belief Universes for Expert Systems. In *Proceedings of the 11th joint Conference on Artificial Intelligence*, pages 1080–1085.

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

Apostol, T. (1974). *Mathematical Analysis*. Addison-Wesley, second edition.

Arnborg, S., Corneil, D. G., and Proskurowski, A. (1987). Complexity of Finding Embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284.

Beinlich, I., Suermondt, G., Chavez, R., and Cooper, G. F. (1989). The ALARM Monitoring System. In *Proceedings of the Second European Conference on Artificial Intelligence and Medicine*, pages 247–256. Springer.

Bernardo, J. M. and Smith, A. F. M. (1994). *Bayesian Theory*. Wiley & Sons.

Beygelzimer, A. and Rish, I. (2003). Approximability of Probability Distributions. In *Advances in Neural Information Processing Systems 16*. The MIT Press.

Billingsley, P. (1986). *Probability and Measure*. Wiley & Sons.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Boutilier, C., Friedman, N., Goldszmidt, M., and Koller, D. (1996). Context-Specific Independence in Bayesian Networks. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers.

Bozga, M. and Maler, O. (1999). On the Representation of Probabilities over Structured Domains. In *Proceedings of the 11th International Conference on Computer Aided Verification*, pages 261–273. Springer.

Cano, A., Moral, S., and Salmerón, A. (2000). Penniless Propagation in Join Trees. *International Journal of Intelligent Systems*, 15(11):1027–1059.

Caprile, B. (2001). Uniformly Generating Distribution Functions for Discrete Random Variables. Technical report, ITC-irst - Centro per la Ricerca Scientifica e Tecnologica, Italy.

Castelo, R. (2002). *The Discrete Acyclic Digraph Markov Model in Data Mining*. PhD thesis, University of Utrecht, The Netherlands.

Castelo, R. and Kočka, T. (2003). On Inclusion-driven Learning of Bayesian Networks. *Journal of Machine Learning Research*, 4:527–574.

Castillo, E., Gutiérrez, J. M., and Hadi, A. S. (1997). *Expert Systems and Probabilistic Network Models*. Springer-Verlag.

Cheeseman, P. and Stutz, J. (1996). Bayesian Classification (AutoClass): Theory and Results. In *Advances in Knowledge Discovery and Data Mining*, pages 153–180. AAAI Press.

Chickering, D. M. (1995). A Transformational Characterization of Equivalent Bayesian Network Structures. In *Proceedings of Eleventh Conference on Uncertainty in Artificial Intelligence,* Montreal, QU, pages 87–98. Morgan Kaufmann.

Chickering, D. M. (1996). Learning Bayesian Networks is NP-Complete. In Fisher, D. and Lenz, H., editors, *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer-Verlag.

Chickering, D. M. (2002). Optimal Structure Identification with Greedy Search. *Journal of Machine Learning Research*, 3:507–554.

Chickering, D. M., Heckerman, D., and Meek, C. (1997). A Bayesian Approach to Learning Bayesian Networks with Local Structure. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 80–89. Morgan Kaufmann Publishers.

Chickering, D. M., Heckerman, D., and Meek, C. (2004). Large-Sample Learning of Bayesian Networks is NP-Hard. *The Journal of Machine Learning Research*, 5:1287–1330.

Chickering, D. M. and Meek, C. (2002). Finding Optimal Bayesian Networks. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 94–102. Morgan Kaufmann Publishers.

Chow, C. K. and Liu, C. N. (1968). Approximating Discrete Probability Distributions with Dependence Trees. *IEE Transactions on Information Theory*, 14(3):462–467.

Cooper, G. F. (1987). Probabilistic Inference Using Belief Networks is NP-Hard. Technical report, Knowledge Systems Laboratory, Stanford University.

Cooper, G. F. and Herskovits, E. (1992). A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning*, 9(4):309–347.

Cover, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*. John Wiley & Sons, Inc.

Cowell, R. G. (2001). Conditions Under Which Conditional Independence and Scoring Methods Lead to Identical Selection of Bayesian Network Models. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 91–97. Morgan Kaufmann Publishers.

Cowell, R. G., Dawid, A. P., Lauritzen, S. L., and Spiegelhalter, D. J. (1999). *Probabilistic Networks and Expert Systems*. Springer.

Dagum, P. and Luby, M. (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60:141–153.

Darwiche, A. (2002). A logical approach to factoring belief networks. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning*.

Dawid, A. P. (1979). Conditional independence in statistical theory. *Journal of the Royal Statistical Society, Series B*, 41(1):1–31.

DeGroot, M. H. (1986). *Probability and Statistics*. Addison-Wesley, 2nd edition.

Dempster, A. P., Laird, N. M., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38.

Domingos, P. and Pazzani, M. J. (1997). On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29(2-3):103–130.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. Wiley & Sons.

Elidan, G. (2004). *Learning Hidden Variables in Probabilistic Graphical Models*. PhD thesis, Hebrew University, Jerusalem, Israel.

Elidan, G. and Friedman, N. (2001). Learning the Dimensionality of Hidden Variables. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 144–151.

Elidan, G. and Friedman, N. (2005). Learning hidden variable networks: The information bottleneck approach. *Journal of Machine Learning Research*, 6:81–127.

Esposito, F., Malerba, D., and Semeraro, G. (1997). A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491.

Fisz, M. (1980). *Probability Theory and Mathematical Statistics*. John Wiley & Sons, Inc, 3rd edition.

Flores, M. J., Gámes, J. A., and Moral, S. (2006). The *independency tree* model: a new approach for clustering and factorisation. In *Proceedings of the Third European Workshop on Probabilistic Graphical Models*, pages 83–90.

Flores, M. J., Gámez, J. A., and Olesen, K. G. (2003). Incremental compilation of Bayesian networks. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 233–240. Morgan Kaufmann Publishers.

Geiger, D. and Heckerman, D. (1996). Knowledge representation and inference in similarity networks and Bayesian multinets. *Artificial Intelligence*, 82:45–74.

Geiger, D. and Pearl, J. (1988). On the logic of influence diagrams. In *Proceedings of the 4th workshop on Uncertainty in Artificial Intelligence*, pages 136–147.

Geiger, D., Verma, T., and Pearl, J. (1990). Identifying Independence in Bayesian Networks. *Networks*, 20(5):507–534.

Gomes, C. and Selman, B. (1997). Algorithm Portfolio Design: Theory vs. Practice. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 190–198. Morgan Kaufmann Publishers.

Hájek, A. (Summer 2003). Interpretations of probability. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer.

Heckerman, D. (1995). A tutorial on learning with bayesian networks. Technical report, Microsoft Research.

Jaeger, M. (2004). Probabilistic Decision Graphs - Combining Verification and AI Techniques for Probabilistic Inference. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 12:19–42.

Jaeger, M., Nielsen, J. D., and Silander, T. (2004). Learning Probabilistic Decision Graphs. In *Proceedings of 2nd European Workshop on Probabilistic Graphical Models*, pages 113–120.

Jaeger, M., Nielsen, J. D., and Silander, T. (2006). Learning Probabilistic Decision Graphs. *International Journal of Approximate Reasoning*, 42(1-2):84–100.

Jensen, F. (2006). *HUGIN API Reference Manual.* Hugin Expert A/S, `http://www.hugin.com/`. Version 6.5.

Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs.* Springer.

Jensen, F. V., Lauritzen, S. L., and Olesen, K. G. (1990a). Bayesian Updating in Causal Probabilistic Netwroks by Local Computation. *Computational Statistics Quarterly*, 4:269–282.

Jensen, F. V., Olesen, K. G., and Andersen, S. K. (1990b). An Algebra of Bayesian Belief Universes for Knowledge-Based Systems. *Networks*, 20(5):637–659.

Karciauskas, G., Kočka, T., Jensen, F. V., Larrañaga, P., and Lozano, J. A. (2004). Learning of Latent Class Models by Splitting and Merging Components. In *Proceedings of the Second European Workshop on Probabilistic Graphical Models*, pages 137–144.

Karčiauskas, G. (2005). *Learning with Hidden Variables: A Parameter Reusing Approach for Tree-Structured Bayesian Networks.* PhD thesis, Facutly of Engineering and Science, Aalborg University, Aalborg, Denmark.

Kjærulff, U. (1990). Triangulation of graphs - algorithms giving small total state space. Technical report, Institute for Electronic Systems, Department of Mathematics and Computer Science, Aalborg University.

Kolmogorov, A. N. (1950). *Foundations of the Theory of Probability.* Chelsea Publishing Company, New York.

Kočka, T. (2001). *Graphical Models: learning and applications.* PhD thesis, Faculty of Informatics and Statistics, Prague University of Economis, Prague, Czech Republic.

Kočka, T., Bouckaert, R., and Studený, M. (2001). On characterizing inclusion of Bayesian networks. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 261–268. Morgan Kaufmann Publishers.

Kullbach, S. and Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

Langley, P., Iba, W., and Thompson, K. (1992). An Analysis of Bayesian Classifiers. In *National Conference on Artificial Intelligence*, pages 223–228.

Lauritzen, S. L. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201.

Lauritzen, S. L. (1996). *Graphical Models.* Oxford University Press.

Lauritzen, S. L., Dawid, A. P., Larsen, B. N., and Leimer, H. G. (1990). Independence properties of directed markov fields. *Networks*, 20:491–505.

Lauritzen, S. L. and Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50(2):157–224.

Liang, H., Zhang, H., and Yan, Y. (2006). Decision Trees for Probability Estimation: An Empirical Study. In *Proceedings of the 18th International Conference on Tools with Artificial Intelligence*.

Lowd, D. and Domingos, P. (2005). Naïve Bayes Models for Probability Estimation. In *Proceedings of the Twentysecond International Conference on Machine Learning*, pages 529–536.

Madsen, A. L. (1999). *All Good Things Come to Those Who Are Lazy - Efficient Inference in Bayesian Networks and Influence Diagrams Based on Lazy Evaluation.* PhD thesis, Department of Computer Science, Faculty of Engineering and Science, Aalborg University, Denmark.

Madsen, A. L. and Jensen, F. V. (1998). Lazy Propagation in Junction Trees. Technical report, Aalborg University, Institute for Electronic Systems, Department of Computer Science.

McAllester, D., Collins, M., and Pereira, F. (2004). Case-factor diagrams for structured probabilistic modeling. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, pages 382–391. AUAI Press.

McLachlan, G. J. and Krishnan, T. (1997). *The EM Algorithm and Extensions*. Wiley & Sons.

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

Meek, C. (1997). *Graphical models: selecting causal and statistical models.* PhD thesis, Carnegie Mellon University, Pittsburgh (PA), U.S.A.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Neal, R. M. (1993). Probabilistic Inference Using Markov Chain Monte Carlo Methods. Technical report, Department of Computer Science, University of Toronto.

Neapolitan, R. E. (2003). *Learning Bayesian Networks*. Prentice Hall.

Newman, D., Hettich, S., Blake, C., and Merz, C. (1998). UCI repository of machine learning databases: `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Nielsen, J. D. and Jaeger, M. (2006). An Empirical Study of Efficiency and Accuracy of Probabilistic Graphical Models. In *Proceedings of the Third European Workshop on Probabilistic Graphical Models*, pages 215–222.

Nielsen, J. D., Kočka, T., and Peña, J. M. (2003). On Local Optima in Learning Bayesian Networks. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 435–442. Morgan Kaufmann Publishers.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers.

Pearl, J. and Verma, T. (1987). The logic of representing dependencies by directed graphs. In *Proceedings of the Conference of the American Association of Artificial Intelligence*, pages 374–379.

Peña, J. M., Kočka, T., and Nielsen, J. D. (2004). Featuring Multiple Local Optima to Assist the User in the Interpretation of Induced Bayesian Network Models. In *Proceedings of the Fifteenth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*.

Peña, J. M., Lozano, J. A., and Larrañaga, P. (2002). Learning recursive Bayesian multinets for data clustering by means of constructive induction. *Machine Learning*, 47(1):63–89.

Provost, F. and Domingos, P. (2003). Tree Induction for Probability-Based Ranking. *Machine Learning*, 52:199–215.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

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

Shachter, R. D. (1988). Probabilistic Inference And Influence Diagrams. *Operational Research*, 36(4).

Shafer, G. and Shenoy, P. P. (1990). Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2:327–352.

Skyrms, B. (1984). *Pragmatics and empiricism*. Yale University Press, New Haven and London.

Spirtes, P., Glymour, C., and Scheines, R. (2000). *Causation, Prediction, and Search*. The MIT Press, 2nd edition.

Studený, M. (1989). Multiinformation and the Problem of Characterization of Conditional Independence Relations. *Problems of Control and Information Theory*, 18(1):3–16.

Thiesson, B., Meek, C., Chickering, D. M., and Heckerman, D. (1997). Learning Mixtures of DAG Models. Technical report, Microsoft Research.

Verma, T. and Pearl, J. (1991). Equivalence and synthesis of causal models. In *UAI '90: Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, pages 255–270. Elsevier Science Inc.

Vessereau, A. (1958). Sur les conditions d'application de criterion χ^2 de Pearson. *Rev. Stat. Appl.*, 6(2).

Vilalta, R. and Rish, I. (2003). A decomposition of classes via clustering to explain and improve naive bayes. In *Proceedings of 14th European Conference on Machine Learning*.

Williams, T. and Kelley, C. (2004). *gnuplot : An Interactive Plotting Program*. `http://www.gnuplot.info/`. Manual, version 4.0.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition.

Zhang, N. L. (1998). Computational properties of two exact algorithms for Bayesian networks. *Applied Intelligence*, 9:173–183.

Zhang, N. L. (2004). Hierarchical Latent Class Models for Cluster Analysis. *Journal of Machine Learning Research*, pages 697–723.

Zhang, N. L. and Poole, D. (1994). A simple approach to Bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*.

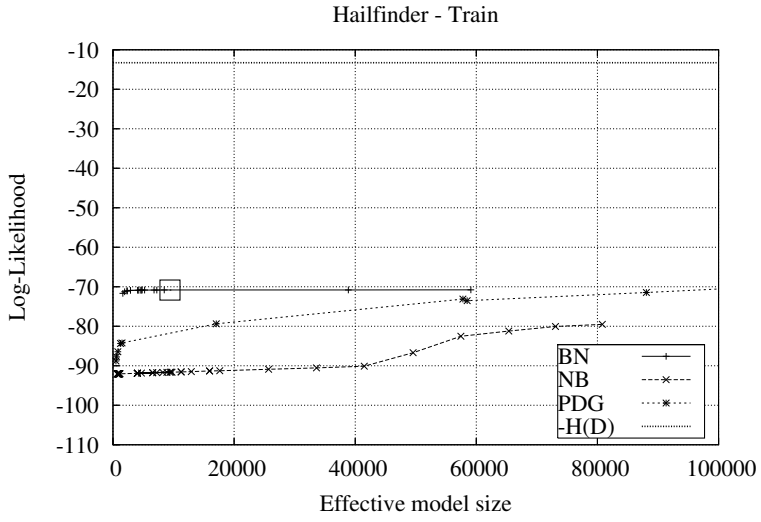
Appendix A

Extended Test Results

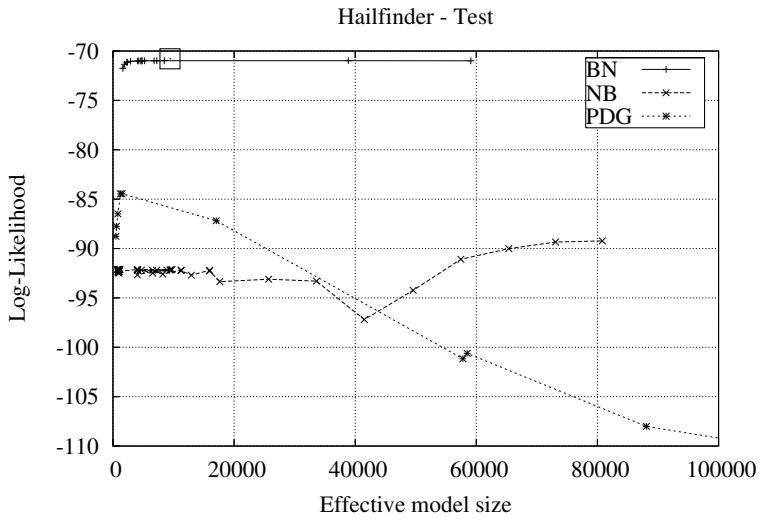
In this appendix we complete the results for learning from synthetic data (see Section 5.2), learning from real data (see Section 5.3), empirically measurements of efficiency and accuracy (see Section 5.4) and for the hybrid learning approach (see Section 5.5). Section A.1 contains results from experiments on synthetic data in the form of SL-curves. Section A.2 contains SL-curves from experiments with real data. Section A.3 contains plots of empirical measurements of computational efficiency and accuracy.

A.1 SL-Curves for Learning from Synthetic Data

Below we bring SL-curves to complete the results of the experiments on data generated from synthetic data. We include the SL-curves for experiments that was previously not explicitly reported in the analysis in Section 5.2, but only in the summary in table 5.4 on page 120. That is, for data sampled from the Hailfinder BN model (figure A.1), from the NB20 NB model (figure A.2) and from the Logic1, Logic3 and Rnd20 PDG models (figures A.3, A.4 and A.5 respectively).

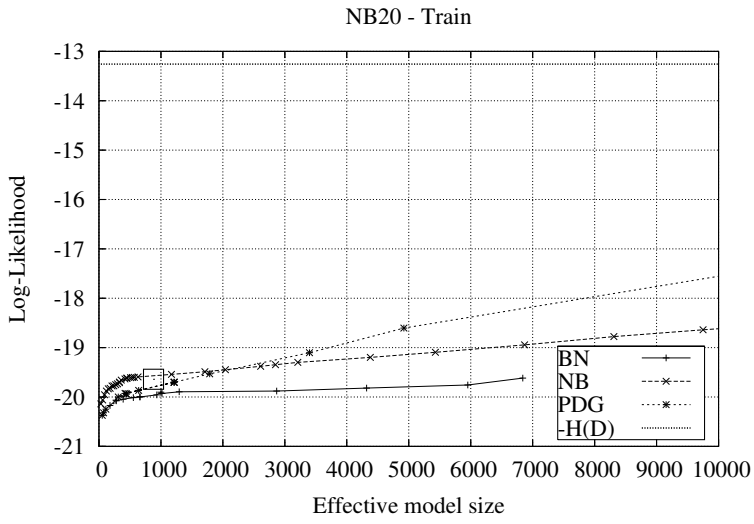


(a)

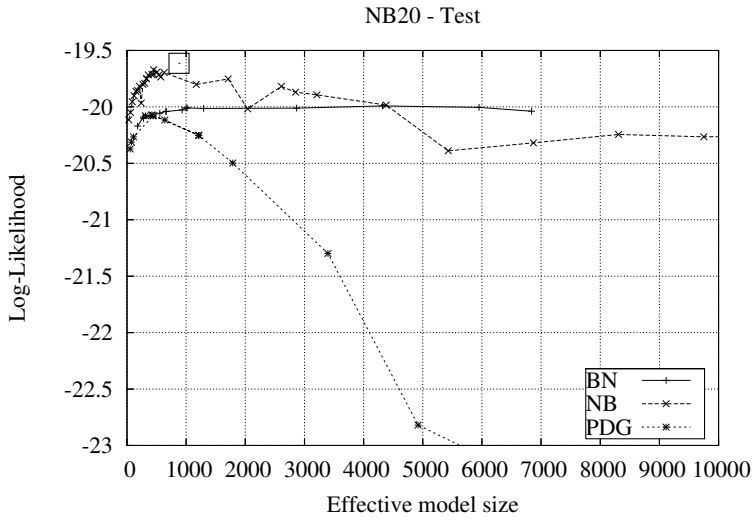


(b)

Figure A.1. SL-curves for models learned from the Hailfinder data, for likelihood values over training data $\mathcal{D}_A$ (a) and test data $\mathcal{D}_B$ (b). The SL coordinates for the generative model is marked with a square.

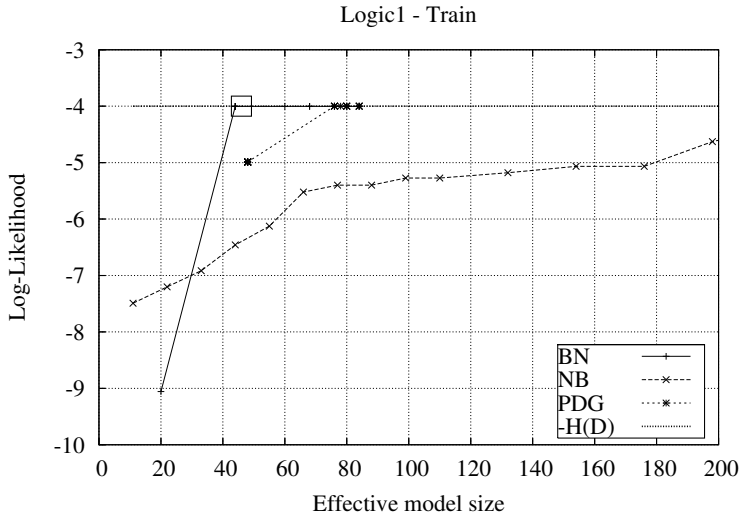


(a)

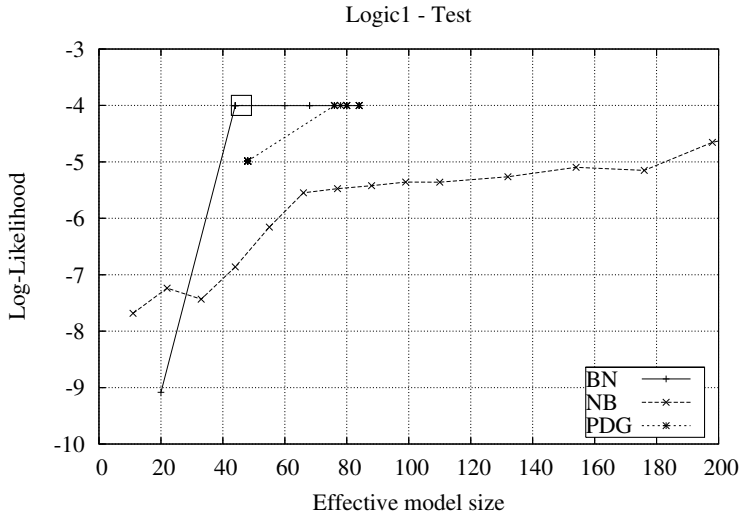


(b)

Figure A.2. SL-curves for models learned from the NB20 data, for likelihood values over training data $\mathcal{D}_A$ (a) and test data $\mathcal{D}_B$ (b). The SL coordinates for the generative model is marked with a square.

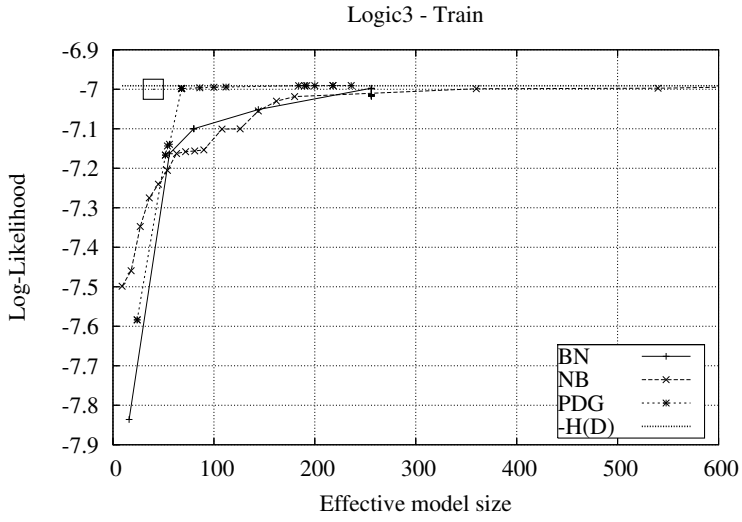


(a)

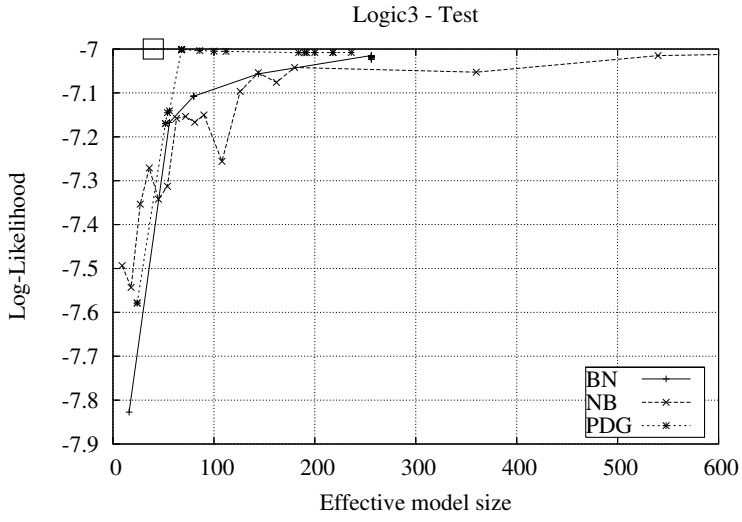


(b)

Figure A.3. SL-curves for models learned from the Logic1 data, for likelihood values over training data $\mathcal{D}_A$ (a) and test data $\mathcal{D}_B$ (b). The SL coordinates for the generative model is marked with a square.

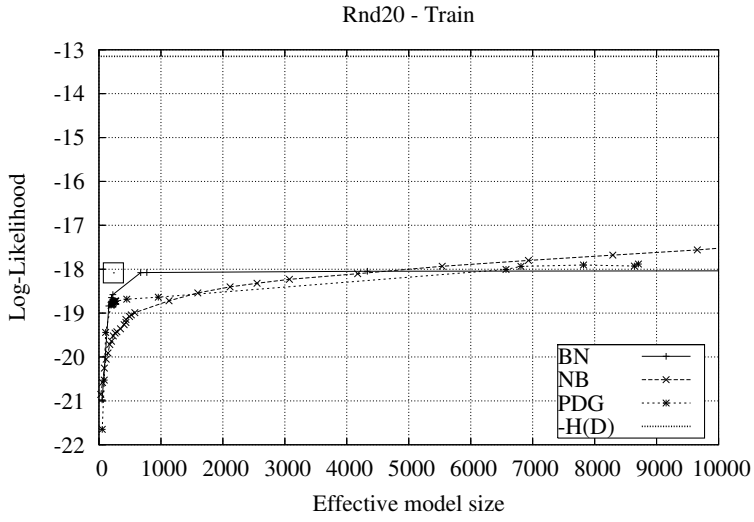


(a)

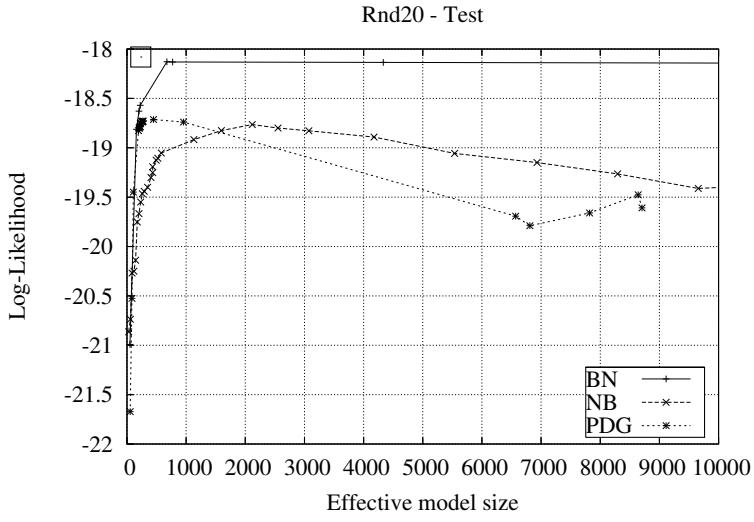


(b)

Figure A.4. SL-curves for models learned from the Logic3 data, for likelihood values over training data $\mathcal{D}_A$ (a) and test data $\mathcal{D}_B$ (b). The SL coordinates for the generative model is marked with a square.



(a)

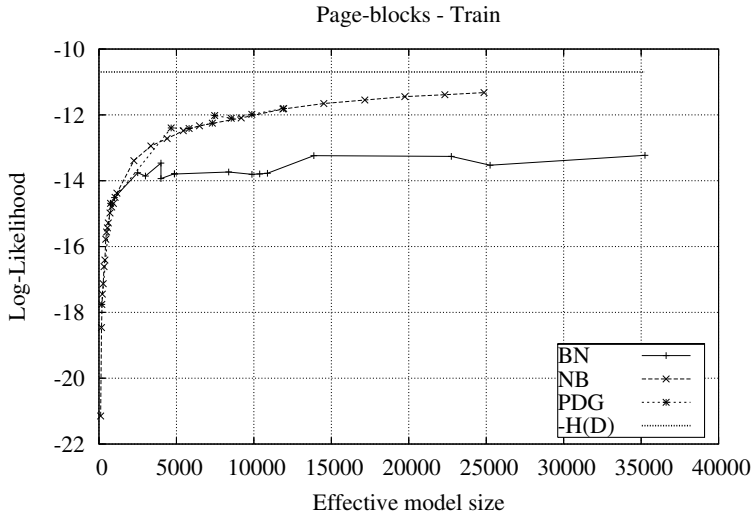


(b)

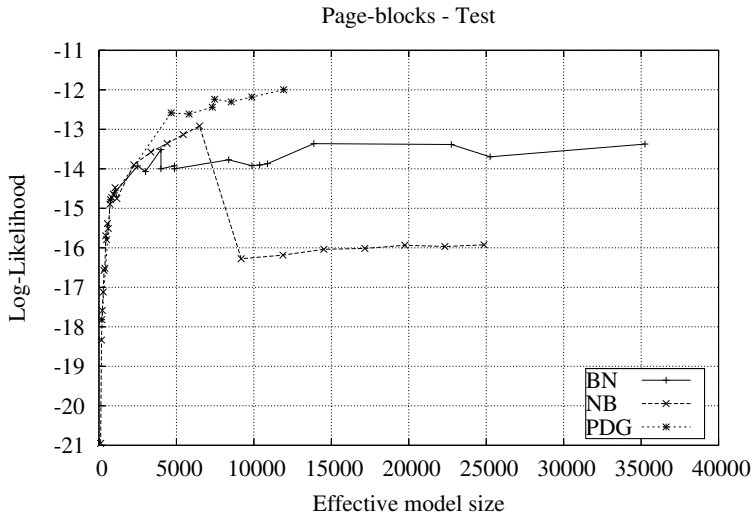
Figure A.5. SL-curves for models learned from the Rnd20 data, for likelihood values over training data $\mathcal{D}_A$ (a) and test data $\mathcal{D}_B$ (b). The SL coordinates for the generative model is marked with a square.

A.2 SL-Curves for Learning from Real Data

In this section we bring SL-curves to complete the results of learning PGMs from real data. We include SL-curves for the experiments that was previously not explicitly included in the analysis of this set of experiments in Section 5.3, but only represented in the summaries of tables 5.6 on page 125 and 5.7 on page 127.

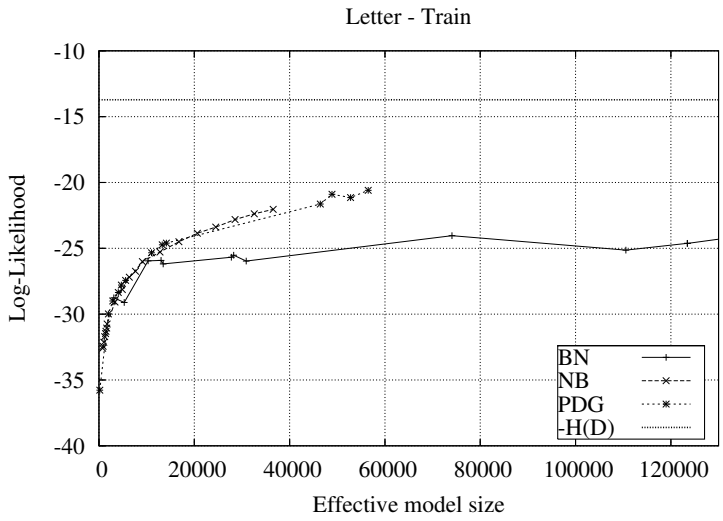


(a)

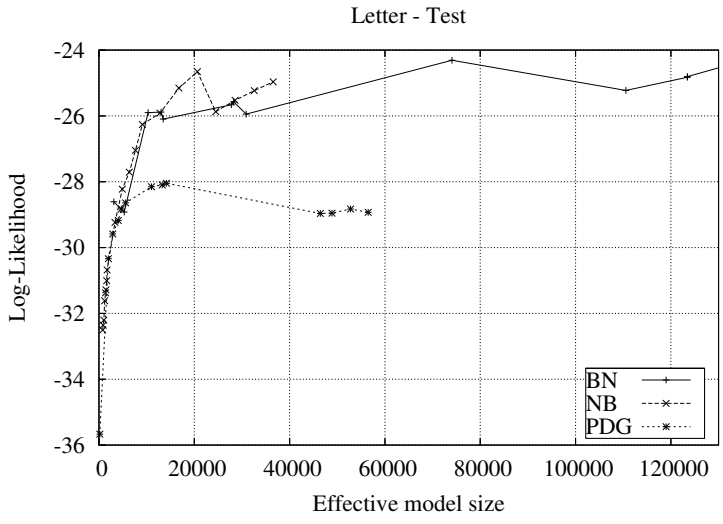


(b)

Figure A.6. SL-curves for models learned from the Page-blocks data.

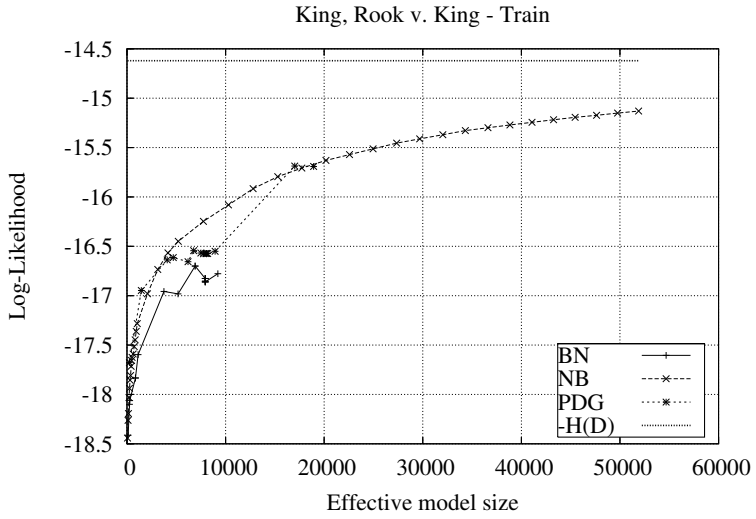


(a)

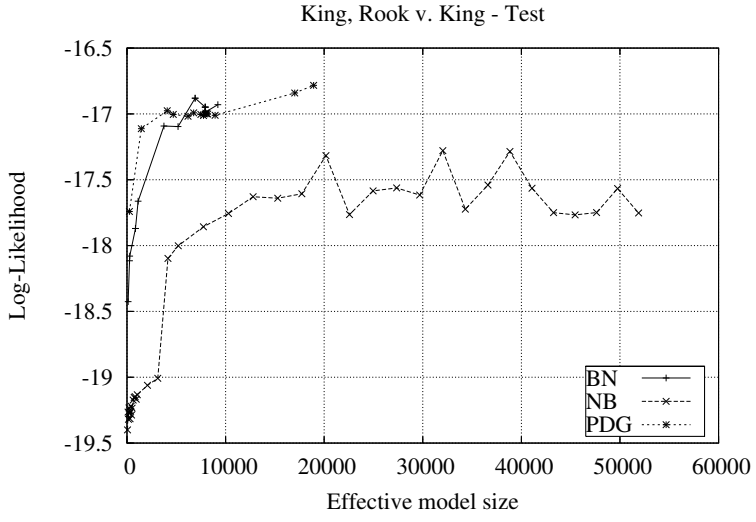


(b)

Figure A.7. SL-curves for models learned from the Letter Recognition data.

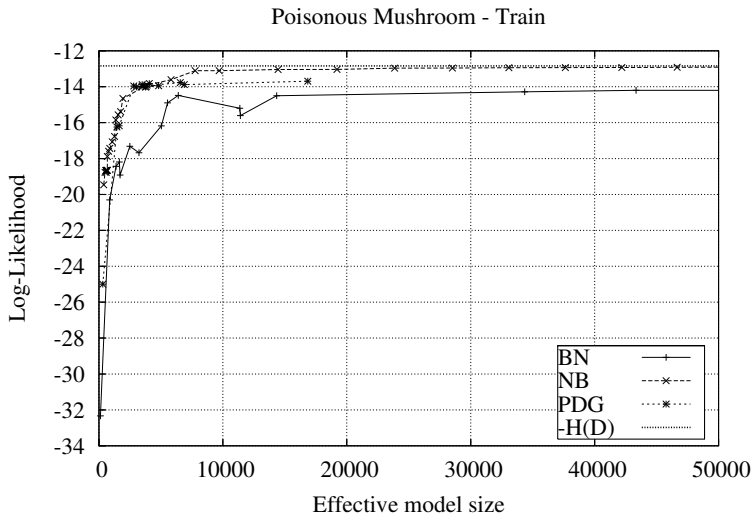


(a)

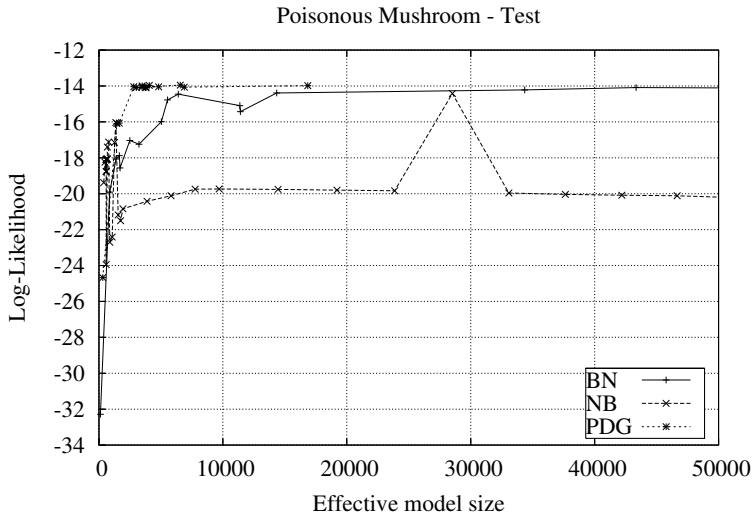


(b)

Figure A.8. SL-curves for models learned from the King, Rook vs. King data.

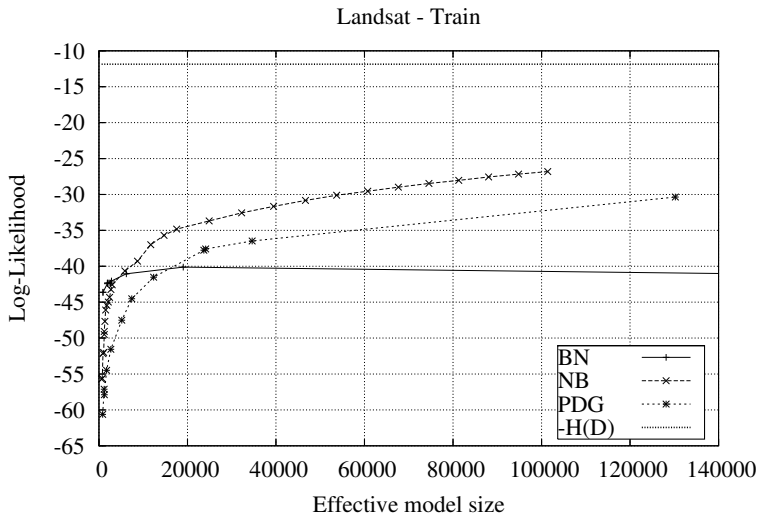


(a)

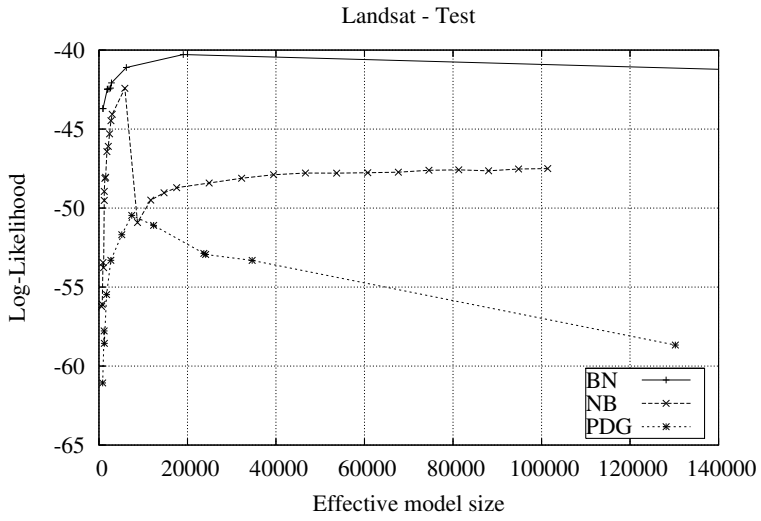


(b)

Figure A.9. SL-curves for models learned from the Poisonous Mushroom data.



(a)

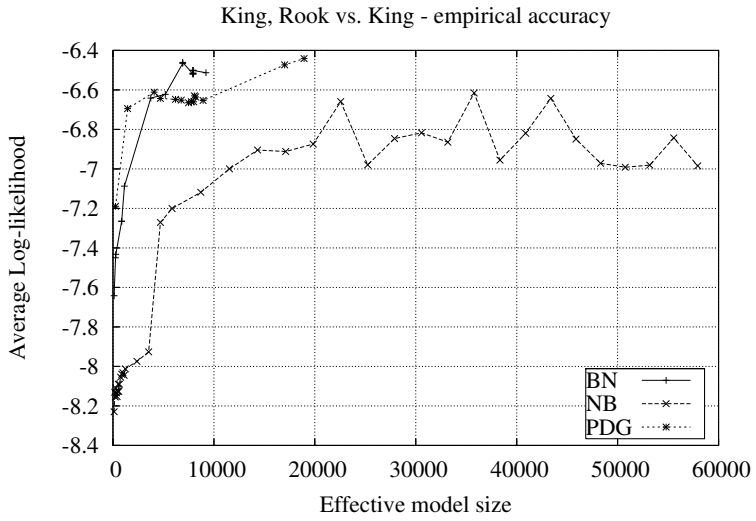


(b)

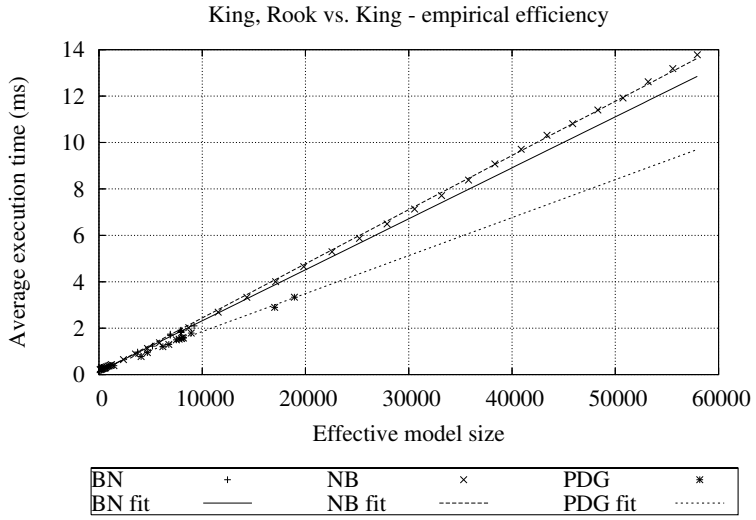
Figure A.10. SL-curves for models learned from the Landsat data.

A.3 Analyses of Empirical Efficiency and Accuracy

In this section we complete the results of the empirical analysis of computational efficiency and accuracy reported in Section 5.4. We bring plots of empirical measures of execution time vs. effective size and the least squares fit to a line, which was previously only summarised by the slope of the fit in table 5.8 on page 130. We also bring SL-curves produced by using empirically measured accuracy that was previously summarised in table 5.9 on page 132.

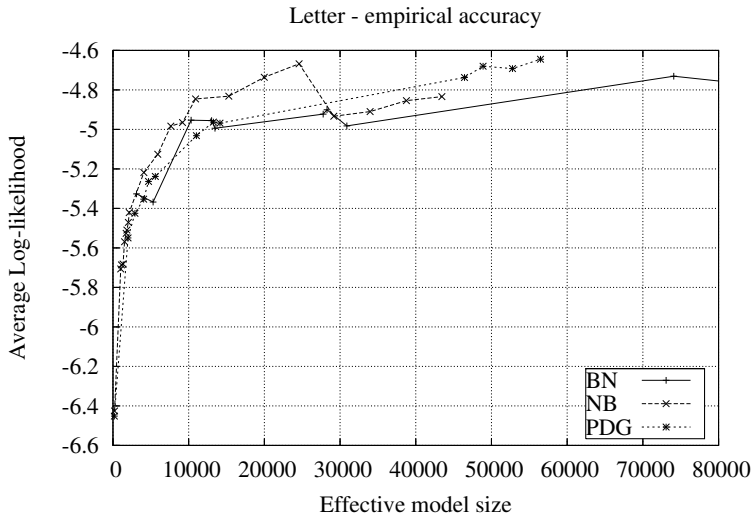


(a)

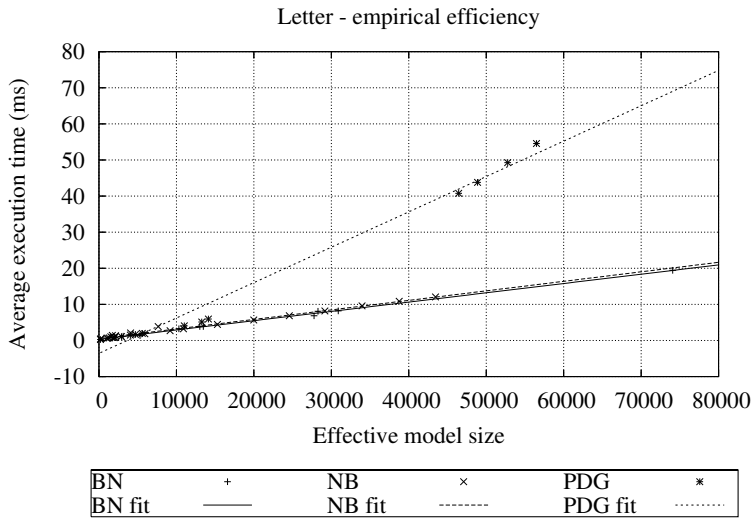


(b)

Figure A.11. Empirical average accuracy (a) and average execution time (b) vs. effective size for the King, rook vs. King dataset.

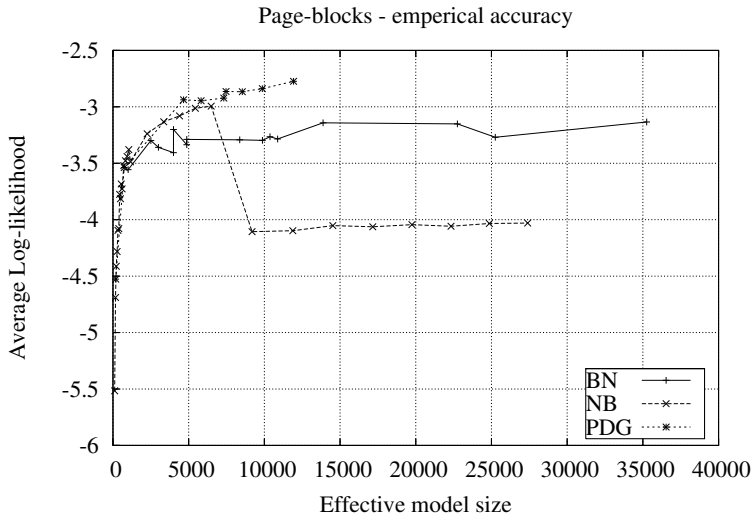


(a)

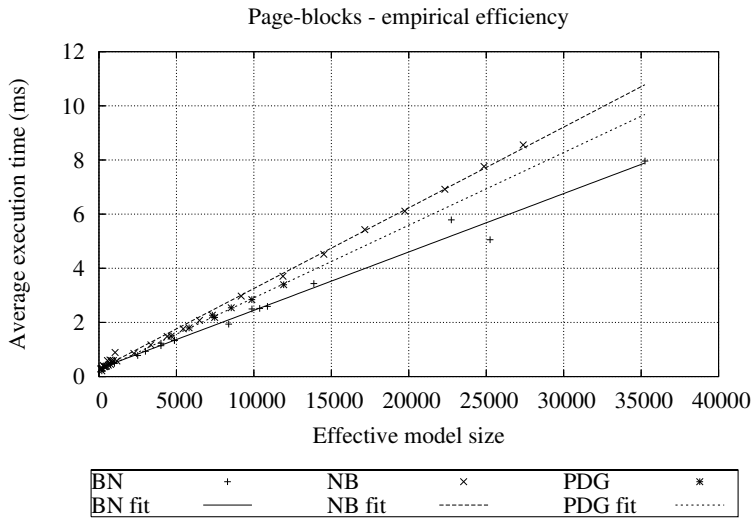


(b)

Figure A.12. Empirical average accuracy (a) and average execution time (b) vs. effective size for the Letter Recognition dataset.

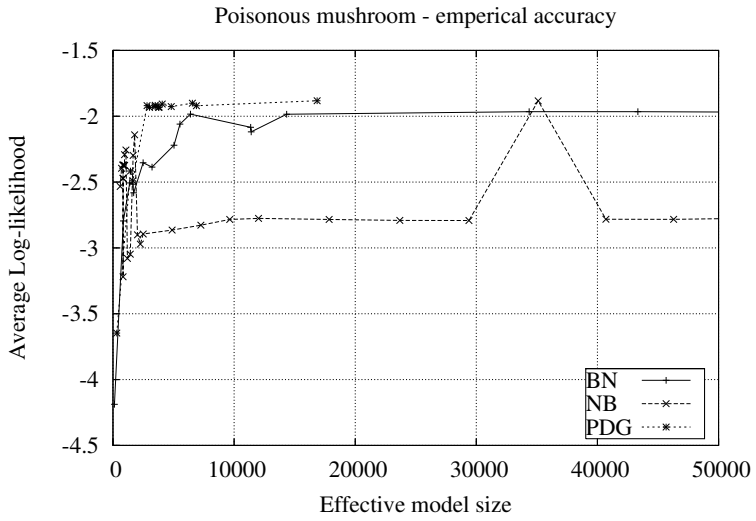


(a)

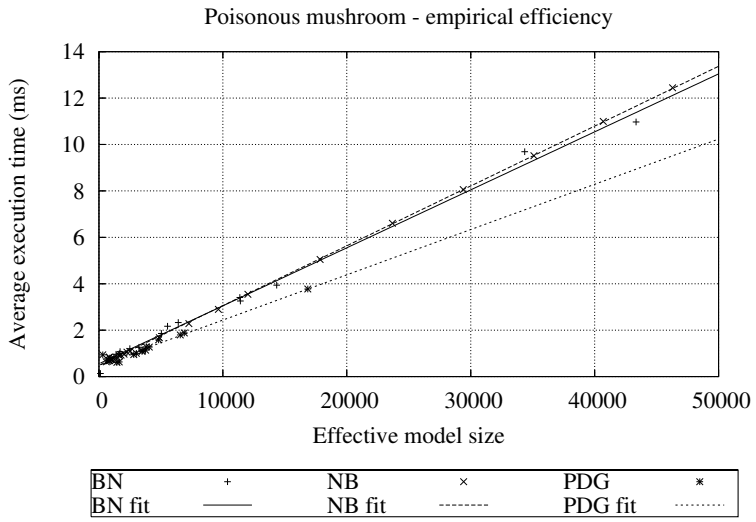


(b)

Figure A.13. Empirical average accuracy (a) and average execution time (b) vs. effective size for the Page-blocks dataset.

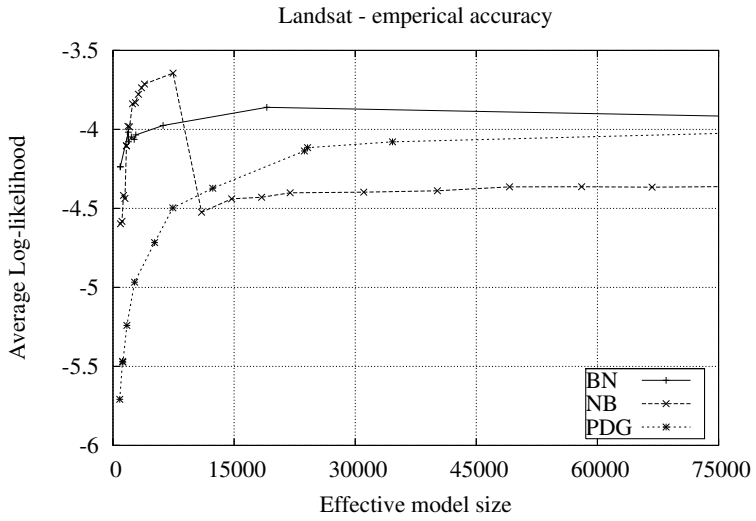


(a)

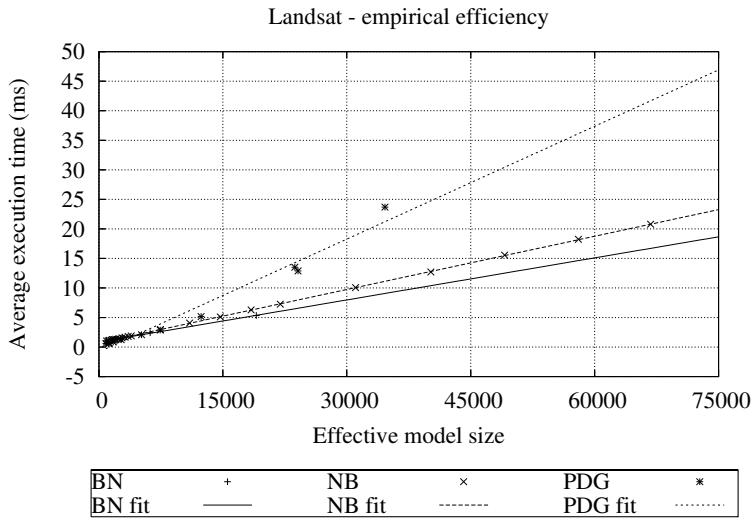


(b)

Figure A.14. Empirical average accuracy (a) and average execution time (b) vs. effective size for the Poisonous mushroom dataset.



(a)



(b)

Figure A.15. Empirical average accuracy (a) and average execution time (b) vs. effective size for the Landsat dataset.

A.4 Detailed Results from Hybrid Learning

In this section, we complete the results of applying hybrid learning on both real and synthetic datasets. Summaries were previously given in tables 5.10 on page 138 and 5.11. Here we present results of all BN models learned from each dataset. In the table below, each row corresponds to one experiment of exposing a specific BN model for the hybrid learning approach described in Section 5.5.

BN			Best PDG			Relative Difference			Time
$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	ES	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
1000	-14.512	-14.607	675	-14.363	-14.482	0.325	0.01	0.009	31
2500	-13.758	-13.927	2600	-13.525	-13.637	-0.04	0.017	0.021	49.9
3000	-13.859	-14.069	2990	-13.388	-13.635	0.003	0.034	0.031	39.1
4000	-13.465	-13.513	3520	-13.066	-13.137	0.12	0.03	0.028	43.5
4000	-13.937	-14	3145	-13.389	-13.463	0.214	0.039	0.038	38.2
4875	-13.788	-13.924	3700	-13.265	-13.417	0.241	0.038	0.036	47
4875	-13.796	-14	5510	-13.365	-13.604	-0.13	0.031	0.028	37.2
8375	-13.734	-13.771	3935	-13.21	-13.286	0.53	0.038	0.035	74.9
9875	-13.808	-13.922	8885	-13.028	-13.179	0.1	0.056	0.053	43.8
10375	-13.794	-13.902	4780	-13.157	-13.25	0.539	0.046	0.047	61.2
10875	-13.774	-13.87	7575	-12.966	-13.104	0.303	0.059	0.055	52.6
13875	-13.239	-13.362	8045	-12.502	-12.577	0.42	0.056	0.059	51.1
22750	-13.26	-13.384	7850	-12.435	-12.472	0.655	0.062	0.068	48.7
25250	-13.529	-13.696	9045	-13.252	-13.338	0.642	0.02	0.026	69.5
35250	-13.228	-13.373	8390	-12.491	-12.569	0.762	0.056	0.06	56.2

Table A.1. Summary of hybrid learning on Page-blocks data.

BN			Best PDG			Relative Difference			Time
$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	ES	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
190	-10.915	-10.851	270	-10.904	-10.837	-0.421	0.001	0.001	66.9
825	-10.572	-10.614	603	-10.499	-10.59	0.269	0.007	0.002	42.9
950	-10.729	-10.762	503	-10.674	-10.69	0.471	0.005	0.007	34.4
1700	-10.495	-10.552	933	-10.404	-10.503	0.451	0.009	0.005	45.4
2700	-10.445	-10.475	758	-10.362	-10.423	0.719	0.008	0.005	38.5
4575	-10.401	-10.442	1003	-10.346	-10.487	0.781	0.005	-0.004	59.8

Table A.2. Summary of hybrid learning on Abalone data.

BN			Best PDG			Relative Difference			Time
$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	ES	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
950	-16.388	-16.473	965	-16.38	-16.468	-0.016	0	0	1654.9
2087	-16.137	-16.256	2224	-16.12	-16.252	-0.066	0.001	0	1833.3
2087	-16.138	-16.258	2224	-16.12	-16.252	-0.066	0.001	0	1835
2119	-16.136	-16.259	2287	-16.117	-16.251	-0.079	0.001	0	1802.2
2286	-16.249	-16.357	1384	-16.235	-16.346	0.395	0.001	0.001	1508.2
2491	-16.229	-16.331	1652	-16.229	-16.336	0.337	0	0	1488.6
2559	-16.125	-16.252	2563	-16.099	-16.241	-0.002	0.002	0.001	1667.8
4826	-16.12	-16.251	2881	-16.1	-16.249	0.403	0.001	0	1583.5
5191	-16.101	-16.235	3859	-16.065	-16.242	0.257	0.002	0	1880.4
6795	-16.097	-16.234	3477	-16.072	-16.248	0.488	0.002	-0.001	1849.8
9331	-16.072	-16.226	8300	-16.027	-16.273	0.11	0.003	-0.003	1855.9
13910	-16.082	-16.221	11694	-16.032	-16.34	0.159	0.003	-0.007	1923.4
16586	-16.036	-16.198	15152	-15.959	-16.258	0.086	0.005	-0.004	1945.5
18966	-16.016	-16.19	14316	-16.011	-16.29	0.245	0	-0.006	2038.6
62270	-15.993	-16.182	31564	-15.86	-16.44	0.493	0.008	-0.016	2068.2

Table A.3. Summary of hybrid learning on Adult data.

BN			Best PDG			Relative Difference			Time
$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	ES	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
120	-18.413	-18.426	124	-18.412	-18.426	-0.033	0	0	1461.8
256	-18.1	-18.115	290	-18.1	-18.114	-0.133	0	0	1428.1
280	-18.062	-18.08	314	-18.062	-18.078	-0.121	0	0	1253.7
864	-17.834	-17.87	986	-17.696	-17.739	-0.141	0.008	0.007	1255.4
864	-17.834	-17.87	986	-17.696	-17.739	-0.141	0.008	0.007	1256.9
1136	-17.597	-17.663	1314	-17.587	-17.654	-0.157	0.001	0.001	993.8
3744	-16.957	-17.092	2906	-16.919	-17.065	0.224	0.002	0.002	1035.9
5184	-16.982	-17.096	1958	-16.92	-17.013	0.622	0.004	0.005	1043.4
6912	-16.7	-16.881	5782	-16.633	-16.835	0.163	0.004	0.003	1045.1
6912	-16.7	-16.88	5782	-16.633	-16.835	0.163	0.004	0.003	1053.3
7936	-16.826	-16.948	6232	-16.713	-16.884	0.215	0.007	0.004	1051.4
7936	-16.826	-16.948	6232	-16.713	-16.884	0.215	0.007	0.004	1059
7936	-16.826	-16.948	6232	-16.713	-16.884	0.215	0.007	0.004	1060.5
7936	-16.852	-16.982	6232	-16.713	-16.884	0.215	0.008	0.006	1061.7
7936	-16.861	-16.987	6232	-16.713	-16.884	0.215	0.009	0.006	1049.7
7936	-16.861	-16.987	6232	-16.713	-16.884	0.215	0.009	0.006	1060.1
7936	-16.861	-16.987	6232	-16.713	-16.884	0.215	0.009	0.006	1047.3
9216	-16.777	-16.93	5206	-16.709	-16.888	0.435	0.004	0.002	1041.5

Table A.4. Summary of hybrid learning on KRvK data.

BN			Best PDG			Relative Difference			Time
$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	ES	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
910	-43.622	-43.705	1210	-43.605	-43.703	-0.33	0	0	266.9
910	-43.622	-43.705	1210	-43.605	-43.703	-0.33	0	0	265.2
1900	-42.416	-42.464	1887	-42.144	-42.252	0.007	0.006	0.005	170.9
1925	-42.326	-42.485	1555	-42.168	-42.407	0.192	0.004	0.002	153.4
2275	-42.322	-42.463	1742	-42.082	-42.354	0.234	0.006	0.003	165
2625	-42.356	-42.405	1925	-42.069	-42.348	0.267	0.007	0.001	138.4
2825	-42.053	-42.073	2061	-41.909	-41.989	0.27	0.003	0.002	180.8
6200	-41.055	-41.103	3167	-40.655	-40.951	0.489	0.01	0.004	171.1
19050	-40.122	-40.283	6170	-39.979	-40.881	0.676	0.004	-0.015	147.5
157650	-41.124	-41.354	38565	-40.87	-51.047	0.755	0.006	-0.234	268.7
323400	-36.55	-36.876	53070	-35.924	-52.307	0.836	0.017	-0.418	382.1
335800	-38.679	-38.865	48932	-37.11	-50.91	0.854	0.041	-0.31	421
383000	-36.03	-36.419	49583	-35.532	-51.044	0.871	0.014	-0.402	463.4
445000	-36.035	-36.465	53415	-35.12	-53.028	0.88	0.025	-0.454	463.9
678875	-36.21	-36.535	60392	-35.73	-54.43	0.911	0.013	-0.49	1351.1
685500	-36.264	-36.707	58798	-35.509	-56.045	0.914	0.021	-0.527	505.8
741375	-36.015	-36.425	58963	-34.995	-55.784	0.92	0.028	-0.531	1897.6
743375	-36.1	-36.362	67371	-35.972	-55.812	0.909	0.004	-0.535	1363.5
760375	-35.989	-36.304	64796	-35.836	-55.776	0.915	0.004	-0.536	1413.4
853875	-36.261	-36.679	68605	-35.299	-56.923	0.92	0.027	-0.552	1351.9
966750	-36.228	-36.589	82237	-35.318	-63.261	0.915	0.025	-0.729	2870.2
988250	-36.007	-36.425	62803	-35.487	-59.231	0.936	0.014	-0.626	2467.8
1082375	-36.177	-36.57	62990	-35.984	-58.839	0.942	0.005	-0.609	2028.3
1097875	-36.078	-36.553	73346	-35.221	-60.162	0.933	0.024	-0.646	2290.4
1861250	-35.75	-36.321	71141	-34.78	-58.26	0.962	0.027	-0.604	3229.9

Table A.5. Summary of hybrid learning on Landsat data.

BN			Best PDG			Relative Difference			Time
$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	ES	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
3119	-28.759	-28.609	2683	-28.372	-28.275	0.14	0.013	0.012	2049.2
5303	-29.112	-28.919	2069	-28.631	-28.493	0.61	0.017	0.015	1486.2
10321	-25.95	-25.9	11633	-25.615	-25.625	-0.127	0.013	0.011	1763.5
13025	-25.923	-25.883	10022	-25.812	-25.763	0.231	0.004	0.005	1708.4
13479	-26.184	-26.096	6512	-26.071	-26.052	0.517	0.004	0.002	2212.4
27775	-25.678	-25.66	19875	-25.586	-25.748	0.284	0.004	-0.003	1673.1
28275	-25.523	-25.574	25282	-25.365	-25.518	0.106	0.006	0.002	1878.2
30895	-25.971	-25.945	21826	-25.383	-25.525	0.294	0.023	0.016	2255.1
74075	-24.037	-24.311	19660	-23.886	-24.809	0.735	0.006	-0.02	2679.5
110559	-25.141	-25.227	24862	-24.896	-26.144	0.775	0.01	-0.036	2752.4
123455	-24.621	-24.827	34627	-24.464	-25.464	0.72	0.006	-0.026	2975.6
123455	-24.627	-24.805	34627	-24.464	-25.464	0.72	0.007	-0.027	2958.9
131125	-24.255	-24.495	38396	-24.178	-25.311	0.707	0.003	-0.033	3157.4
141675	-23.477	-23.872	62494	-23.317	-25.254	0.559	0.007	-0.058	2714.3
188845	-23.842	-24.165	54991	-23.772	-25.289	0.709	0.003	-0.047	2998.4
447075	-23.875	-24.2	45407	-23.869	-25.544	0.898	0	-0.056	3897.1

Table A.6. Summary of hybrid learning on Letter data.

BN			Best PDG			Relative Difference			Time
$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	ES	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
121	-32.332	-32.283	123	-32.332	-32.283	-0.017	0	0	1737.4
862	-20.301	-19.911	856	-19.627	-19.265	0.007	0.033	0.032	68.1
1382	-18.432	-18.067	1041	-17.831	-17.527	0.247	0.033	0.03	69.1
1646	-18.195	-17.878	1024	-17.248	-16.981	0.378	0.052	0.05	69.5
1707	-18.918	-18.564	977	-18.012	-17.771	0.428	0.048	0.043	70.3
2493	-17.317	-17.03	1289	-16.528	-16.329	0.483	0.046	0.041	75.4
3227	-17.678	-17.245	1231	-16.682	-16.369	0.619	0.056	0.051	71.8
5037	-16.182	-15.986	1674	-15.534	-15.423	0.668	0.04	0.035	71.9
5531	-14.898	-14.771	1860	-14.446	-14.373	0.664	0.03	0.027	72.5
6401	-14.487	-14.452	1945	-14.187	-14.165	0.696	0.021	0.02	72.2
11369	-15.197	-15.096	1573	-14.598	-14.556	0.862	0.039	0.036	71.8
11411	-15.607	-15.43	1500	-14.718	-14.616	0.869	0.057	0.053	72.3
14339	-14.505	-14.392	2272	-13.988	-13.96	0.842	0.036	0.03	75.6
34359	-14.284	-14.218	2171	-13.719	-13.712	0.937	0.04	0.036	79.4
43341	-14.197	-14.09	2167	-13.654	-13.623	0.95	0.038	0.033	82.8
68377	-14.207	-14.14	2288	-13.669	-13.676	0.967	0.038	0.033	87.9
70731	-14.268	-14.154	2139	-13.874	-13.789	0.97	0.028	0.026	90
81463	-14.451	-14.39	2363	-13.949	-13.951	0.971	0.035	0.031	87.7
85477	-13.937	-13.931	1878	-13.535	-13.55	0.978	0.029	0.027	98.5
93183	-14.503	-14.429	1771	-13.997	-13.966	0.981	0.035	0.032	120.6
114741	-14.281	-14.194	2212	-13.747	-13.772	0.981	0.037	0.03	113.5
208333	-13.924	-13.883	2010	-13.564	-13.56	0.99	0.026	0.023	242.3
215351	-14.426	-14.378	1902	-13.771	-13.803	0.991	0.045	0.04	141.4

Table A.7. Summary of hybrid learning on Mushroom data.

BN			Best PDG			Relative Difference			Time
$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	ES	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
142	-18.342	-18.618	171	-18.5	-18.431	-0.204	-0.009	0.01	2924.5
312	-13.963	-14.181	480	-14.066	-13.96	-0.538	-0.007	0.016	1994.8
333	-13.853	-14.051	525	-13.943	-13.856	-0.577	-0.006	0.014	1870.1
335	-13.847	-14.047	538	-13.937	-13.848	-0.606	-0.006	0.014	1882.4
432	-13.735	-13.936	691	-13.821	-13.741	-0.6	-0.006	0.014	1888.2
448	-13.686	-13.885	762	-13.765	-13.68	-0.701	-0.006	0.015	1749.9
496	-13.678	-13.879	849	-13.756	-13.68	-0.712	-0.006	0.014	1762.1
504	-13.675	-13.874	857	-13.752	-13.677	-0.7	-0.006	0.014	1755.3
598	-13.671	-13.875	898	-13.744	-13.678	-0.502	-0.005	0.014	1726.4
624	-13.666	-13.868	858	-13.741	-13.669	-0.375	-0.005	0.014	1694.2
636	-13.666	-13.869	876	-13.741	-13.669	-0.377	-0.005	0.014	1702.8
636	-13.666	-13.869	876	-13.741	-13.669	-0.377	-0.005	0.014	1696.7
646	-13.665	-13.869	904	-13.74	-13.669	-0.399	-0.005	0.014	1815.3
811	-13.665	-13.868	1246	-13.735	-13.672	-0.536	-0.005	0.014	1729.5
1632	-13.661	-13.871	2760	-13.716	-13.697	-0.691	-0.004	0.013	1710.2
1760	-13.657	-13.87	3215	-13.712	-13.69	-0.827	-0.004	0.013	1862.2
2862	-13.654	-13.872	5391	-13.688	-13.717	-0.884	-0.002	0.011	1910.2
6330	-13.652	-13.876	13736	-13.672	-13.755	-1.17	-0.001	0.009	1815.6
6372	-13.649	-13.876	12482	-13.659	-13.775	-0.959	-0.001	0.007	1708.1
25233	-13.638	-13.896	85277	-13.6	-13.873	-2.38	0.003	0.002	1751.8

Table A.8. Summary of hybrid learning on Alarm data.

BN			Best PDG			Relative Difference			Time
$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	ES	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
1628	-71.678	-71.75	1957	-71.618	-71.698	-0.202	0.001	0.001	236.4
1933	-71.266	-71.349	2573	-71.244	-71.348	-0.331	0	0	227.7
2289	-71.04	-71.131	2752	-71.031	-71.151	-0.202	0	0	248.1
2361	-71.053	-71.144	2867	-71.044	-71.183	-0.214	0	-0.001	265.2
2884	-70.917	-71.03	3508	-70.855	-71.033	-0.216	0.001	0	337.6
4070	-70.881	-71.008	4373	-70.824	-71.028	-0.074	0.001	0	259.8
4253	-70.876	-71.005	4646	-70.81	-71.029	-0.092	0.001	0	290.2
4622	-70.862	-70.997	4060	-70.81	-71.01	0.122	0.001	0	349.7
4639	-70.857	-71.002	4835	-70.793	-71.023	-0.042	0.001	0	274.5
4820	-70.842	-70.989	4778	-70.787	-71.027	0.009	0.001	-0.001	310.1
5226	-70.843	-70.993	5584	-70.822	-71.073	-0.069	0	-0.001	269.7
6776	-70.822	-70.991	5543	-70.766	-71.105	0.182	0.001	-0.002	440.4
7234	-70.832	-70.988	5144	-70.786	-71.071	0.289	0.001	-0.001	354.1
8472	-70.804	-70.976	5691	-70.761	-71.162	0.328	0.001	-0.003	339.8
38871	-70.8	-70.983	18351	-70.714	-72.133	0.528	0.001	-0.016	573.5
59098	-70.793	-70.988	32579	-70.75	-75.619	0.449	0.001	-0.065	4618.6

Table A.9. Summary of hybrid learning on Hailfinder data.

BN			Best PDG			Relative Difference			Time
$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	ES	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
300	-21.222	-21.306	463	-21.22	-21.305	-0.543	0	0	1032.9
531	-21.138	-21.231	670	-21.096	-21.196	-0.262	0.002	0.002	951.1
780	-21.081	-21.19	858	-21.032	-21.161	-0.1	0.002	0.001	983.2
780	-21.104	-21.207	487	-21.097	-21.21	0.376	0	0	959.9
906	-20.967	-21.112	1301	-20.964	-21.111	-0.436	0	0	1018.8
1041	-20.973	-21.12	1163	-20.95	-21.106	-0.117	0.001	0.001	961.5
1296	-20.886	-21.068	1638	-20.879	-21.075	-0.264	0	0	1010.8
1458	-20.87	-21.074	1559	-20.856	-21.077	-0.069	0.001	0	1026.2
1602	-20.92	-21.1	1697	-20.873	-21.09	-0.059	0.002	0	1005.7
2772	-20.842	-21.059	3550	-20.758	-21.092	-0.281	0.004	-0.002	1028.5
3276	-20.876	-21.062	3048	-20.732	-21.077	0.07	0.007	-0.001	1040.7
4128	-20.806	-21.061	7473	-20.676	-21.14	-0.81	0.006	-0.004	1035.3
5220	-20.687	-21.051	9074	-20.564	-21.166	-0.738	0.006	-0.005	1121.1
6264	-20.627	-21.053	9714	-20.554	-21.12	-0.551	0.004	-0.003	1080.8
19116	-20.486	-21.111	14794	-20.447	-21.336	0.226	0.002	-0.011	1345.6

Table A.10. Summary of hybrid learning on NB10 data.

BN			Best PDG			Relative Difference			Time
$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	ES	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
179	-20.172	-20.17	224	-20.151	-20.155	-0.251	0.001	0.001	1028.3
275	-20.065	-20.09	403	-20.064	-20.088	-0.465	0	0	981.8
280	-20.078	-20.102	388	-20.062	-20.097	-0.386	0.001	0	944.3
391	-20.045	-20.077	580	-20.033	-20.073	-0.483	0.001	0	998.3
555	-20.013	-20.056	1184	-19.987	-20.048	-1.133	0.001	0	998.4
661	-19.995	-20.039	1387	-19.972	-20.031	-1.098	0.001	0	1009.1
661	-20.001	-20.041	1373	-19.967	-20.032	-1.077	0.002	0	990.4
934	-19.958	-20.027	1045	-19.938	-20.025	-0.119	0.001	0	1016.1
1000	-19.903	-20.01	1402	-19.903	-20.012	-0.402	0	0	990.1
1018	-19.93	-20.009	1236	-19.925	-20.012	-0.214	0	0	1017.7
1296	-19.895	-20.013	1425	-19.889	-20.01	-0.1	0	0	1012.1
2868	-19.88	-20.01	4771	-19.779	-20.044	-0.664	0.005	-0.002	1071.8
4320	-19.819	-19.991	4193	-19.81	-20.083	0.029	0	-0.005	1062.5
5952	-19.758	-20.004	7655	-19.605	-20.137	-0.286	0.008	-0.007	1078.7
6840	-19.618	-20.038	6662	-19.597	-20.093	0.026	0.001	-0.003	1065.9

Table A.11. Summary of hybrid learning on NB20 data.

BN			Best PDG			Relative Difference			Time
$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	ES	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
42	-16.229	-16.256	46	-16.229	-16.256	-0.095	0	0	1677.1
88	-15.345	-15.361	110	-15.33	-15.343	-0.25	0.001	0.001	681.8
103	-15.263	-15.274	114	-15.256	-15.276	-0.107	0	0	1131.9
130	-15.154	-15.16	123	-15.153	-15.167	0.054	0	0	1091
142	-15.157	-15.158	148	-15.129	-15.135	-0.042	0.002	0.002	1059.9
219	-14.991	-14.998	211	-14.991	-14.991	0.037	0	0	1056.2
231	-14.97	-14.977	263	-14.969	-14.968	-0.139	0	0.001	1001.8
261	-14.953	-14.968	258	-14.952	-14.961	0.011	0	0	1018.3
375	-14.901	-14.925	418	-14.9	-14.921	-0.115	0	0	955.3
657	-14.872	-14.913	577	-14.865	-14.908	0.122	0	0	956.5
1080	-14.837	-14.903	607	-14.833	-14.909	0.438	0	0	973.9
2958	-14.8	-14.913	1188	-14.789	-14.958	0.598	0.001	-0.003	993.9
16065	-14.735	-14.989	10741	-14.701	-15.348	0.331	0.002	-0.024	1169.8
99387	-14.518	-15.351	19509	-14.325	-16.329	0.804	0.013	-0.064	1937.9
135108	-14.617	-15.252	11169	-14.272	-17.672	0.917	0.024	-0.159	2148.3
205578	-14.433	-15.626	14358	-14.17	-16.664	0.93	0.018	-0.066	2127.4
271188	-14.636	-16.092	9885	-14.169	-16.718	0.964	0.032	-0.039	2240.7
944784	-14.485	-16.309	11956	-13.35	-20.409	0.987	0.078	-0.251	5996.7

Table A.12. Summary of hybrid learning on Rnd15 data.

BN			Best PDG			Relative Difference			Time
$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	ES	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	$size_{eff}$	$L(\mathcal{D}_A)$	$L(\mathcal{D}_B)$	(seconds)
51	-20.977	-20.991	56	-20.977	-20.991	-0.098	0	0	2861.4
159	-18.837	-18.813	186	-18.791	-18.771	-0.17	0.002	0.002	1878.1
203	-18.646	-18.629	238	-18.565	-18.543	-0.172	0.004	0.005	1736
224	-18.58	-18.57	233	-18.515	-18.505	-0.04	0.003	0.004	1799.9
674	-18.079	-18.129	669	-18.077	-18.134	0.007	0	0	1691.8
772	-18.076	-18.131	627	-18.076	-18.136	0.188	0	0	1579.6
4332	-18.047	-18.136	1968	-18.014	-18.221	0.546	0.002	-0.005	1694.9
25488	-18.009	-18.156	11956	-17.969	-18.622	0.531	0.002	-0.026	1935.2
48564	-17.94	-18.241	12302	-17.912	-18.834	0.747	0.002	-0.033	2410
131328	-17.76	-18.505	20418	-17.709	-20.074	0.845	0.003	-0.085	3272.9
289008	-17.607	-18.772	31620	-17.164	-22.148	0.891	0.025	-0.18	4444
727056	-17.529	-19.678	26614	-16.649	-26.175	0.963	0.05	-0.33	9434.2

Table A.13. Summary of hybrid learning on Rnd20 data.

Appendix B

On Expectation when Sampling with Replacement

Let $\mathcal{S}$ be a set of N distinct elements. Consider the experiment of sampling from $\mathcal{S}$ with replacement, and let R be the size of a sample with replacement. Define the random variable X on the sample space of the N^R different possible sequences of such samples:

$$X^{(R)} : \text{Number of } \textit{distinct} \text{ elements in a sample of size } R. \tag{B.1}$$

Let $new(K)$ be true if the K'th element drawn is an object not drawn before, and false otherwise. We can define $X^{(R)}$ recursively as:

$$X^{(R)} = X^{(R-1)} + \mathbf{1}(new(R)), \tag{B.2}$$

where $\mathbf{1}(\cdot)$ is the indicator function, here assuming value 1 when the R'th draw results in sampling an element we have not sampled before, and 0 otherwise.

Theorem B.1
The expected value of $X^{(R)}$ is:

$$E[X^{(R)}] = \sum_{i=1}^{R} \left(\frac{N-1}{N}\right)^{(i-1)} \tag{B.3}$$

Proof: We will prove the theorem by induction in R.

For $R = 1$ we only draw a single which will trivially always be distinct from every other element drawn, and $E[X^{(R)}] = 1$ from (B.3).

Assume (B.3) holds for $R-1$. For R we can then write the expectation as:

$$E[X^{(R)}] = E[X^{(R-1)}] + E[\mathbf{1}(new(R))]. \tag{B.4}$$

By the induction hypothesis $E[X^{(R-1)}]$ is given by eq. (B.3), so we need to show that:

$$E[\mathbf{1}(new(R))] = \left(\frac{N-1}{N}\right)^{R-1}. \tag{B.5}$$

Let $distinct(k, l)$ be true if k distinct elements have been sampled in the first l draws and false otherwise. We can then derive an expression for $E[\mathbf{1}(new(n))]$.

$$\begin{aligned}
E[\mathbf{1}(new(R))] &= P(new(R)) \\
&= \sum_{k=1}^{R-1} P(new(R)|distinct(k, R-1)) \cdot P(distinct(k, R-1)) \\
&= \sum_{k=1}^{R-1} \frac{N-k}{N} \cdot P(distinct(k, R-1)) \\
&= \sum_{k=1}^{R-1} P(distinct(k, R-1)) - \sum_{k=1}^{R-1} \frac{k}{N} P(distinct(k, R-1)) \\
&= 1 - \frac{1}{N} E[X^{(R-1)}] \\
&= 1 - \frac{1}{N} \sum_{i=1}^{R-1} \left(\frac{N-1}{N}\right)^{i-1} \\
&= 1 - \frac{1}{N} \sum_{i=1}^{R-1} \frac{(N-1)^{i-1}}{N^{R-2}} N^{R-i-1} \\
&= 1 - \frac{1}{N^{R-1}} \sum_{i=1}^{R-1} (N-1)^{i-1} N^{R-i-1} \\
&= \frac{1}{N^{R-1}} \left(N^{R-1} - \sum_{i=1}^{R-1} (N-1)^{i-1} N^{R-i-1} \right)
\end{aligned}$$

From this it is clear that to show relation (B.5), it is sufficient to show that:

$$N^{R-1} - \sum_{i=1}^{R-1} (N-1)^{i-1} N^{R-i-1} = (N-1)^{R-1} \tag{B.6}$$

We show (B.6) by induction in R. For $R = 1$ (B.6) is satisfied. Assume (B.6) is satisfied for $R-1$, for R we then get:

$$\begin{aligned}
N^{R-1} - \sum_{i=1}^{R-1} (N-1)^{i-1} N^{R-i-1} &= N^{R-1} - N^{R-2} - \sum_{i=2}^{R-1} (N-1)^{i-1} N^{R-i-1} \\
&= N^{R-1} - N^{R-2} - \sum_{i=1}^{R-2} (N-1)^{i} N^{R-i-2} \\
&= (N-1) \left(\frac{N^{R-1} - N^{R-2}}{N-1} - \sum_{i=1}^{R-2} (N-1)^{i-1} N^{R-i-2} \right) \\
&= (N-1) \left(\frac{N^{R-2}(N-1)}{(N-1)} - \sum_{i=1}^{R-2} (N-1)^{i-1} N^{R-i-2} \right) \\
&= (N-1)^{R-1}
\end{aligned}$$

The last equation is valid by the induction hypothesis and (B.6) is therefore true for all R, which then concludes our proof for theorem B.1. $\square$

Appendix C

Dansk Resumé

Dette resumé er en direkte oversættelse af nærværende afhandling med den danske titel *Om Lærring Uden Opsyn af Probabilistiske Grafiske Modeller*.

Probabilistiske grafiske modeller (PGM'er) er et matematisk begrebsramme til repræsentation af fælles sandsynligheds fordelinger over en mængde tilfældige variable (Cowell et al., 1999; Jensen, 2001; Lauritzen, 1996; Pearl, 1988). PGM'er er blevet en standard tilgang til repræsentation og håndtering af usikkerhed i Kunstig Intelligens. Også i de relaterede områder som Mønster Genkendelse og Maskine Læring har PGM'er modtaget megen opmærksomhed og er blevet andvendt succesfuldt i talrige domæner (Bishop, 2006; Mitchell, 1997; Duda et al., 2001).

Når PGM'er bliver indlært fra data (til forskel fra manuelt konstrueret), bruges der en scoringsfunktion til at vurdere kvaliteten af modeller og derved diskriminere mellem alternativer. Indlærings proceduren vælger så fra de alternative modeller den model der er optimal mht. scoringsfunktionen. Typiske scoringsfunktioner kombinerer en gevinst for præcision med en straf for kompleksitet i en vægtet sum. Generelt kalder vi sådanne scoreringsfunktioner for straffede sandsynligheds scoringsfunktioner, og de antager følgende simple form:

$$S(M, \mathcal{D}) = \lambda \cdot L(\mathcal{D}|M) - (1 - \lambda) \cdot size(M), \tag{C.1}$$

for PGM M, data $\mathcal{D}$, sandsyndlighed L, og en afvejnings koefficient $0 < \lambda < 1$. Typisk vil antallet af forskellige alternative modeller være alt for stort til at kunne foretage en udtømmende søgning, og studier har vist at mange instancer af lærings opgaver for PGM'er er NP-svære (Chickering et al., 2004; Chickering, 1996). Følgelig er det passende og ofte nødvendigt i praksis at anvende heuristiske procedure.

Studiet som rapporteres i denne afhandling har været fokusret på aspekter af indlæring af PGM'er fra data. I det følgende vil vi kort diskutere de problemer vi behandler samt de løsninger vi foreslår.

En af de mest populære typer PGM'er er den Bayesianske Netværks (BN) model (Pearl, 1988; Jensen, 2001). Indlæring af BN modeller har været genstand for megen opmærksomhed og både mere og mindre opløftende resultater er fundet. Mens det er blevet bevist at problemet

at lære BN modeller som optimere (C.1) er et NP-svært problem (Chickering et al., 2004), er det samtidig blevet vist at procedure der genskaber den optimale BN model ofte er brugbare for mange relevante domæner (for eksempel SGS algoritmen (Spirtes et al., 2000) og GES algoritmen (Chickering and Meek, 2002; Meek, 1997)). Disse lærringsprocedure støtter sig imidlertid til den stærke antagelse omkring en data genererende proces der udviser relationer af uafhængighed mellem de observerede variable som kan indkodes i den orienterede ikke cykliske grafiske (eng. directed acyclic graph eller DAG) struktur i en BN model, mao. en proces der er DAG-troværdig. Denne antagelse er ofte urealistisk i anvendelser i den virkelige verden (dvs. ikke syntetisk konstruerede eksempler), og kvaliteten af de lærte modeller kan være meget afhængig af hvorvidt denne antagelse er tilfredsstillet. Derfor kan den praktiske anvendelighed af sådanne procedure være begrænset.

I denne afhandling foreslår vi en simpel generalisering af en grådig søge procedure. Ved generaliseringed introduceres en parameter der muliggør en afvejning af grådighed for tilfældighed i beslutnings strategien der guider søgningen. Ved at anvende flere genstarter sammen med en stokastisk beslutnings strategi opretholder algoritmen den teoretiske optimalitet fra den grådige søgning, og tilmed muliggør dette en bredere afsøgning af søgerummet. Dette vil blive vigtigt når den stærke antagelse af DAG-troværdighed bliver brudt. I sådanne tilfælde kan deterministisk søgning som grådig søgning vise sig at lede til en sub-optimal model mens en multipel genstartet stokastisk søgning vil identificere flere lokalt optimale modeller.

I de fleste anvendelseområder er en af hovedopgaverne for PGM'er at være en repræsentation som tillader effektive opdatering af marginale betingede sandsynligheder (eng. belief updating). Ved opdatering af marginale betingede sandsynligheder forstås processen at beregne alle marginale sandsynligheder for alle variable betinget af observationer af en delmængde af variable. For BN modeller er dette problem NP-svært (Cooper, 1987).Ofte er det dog muligt at finde BN modeller som både udviser en håndterbare beregnelighed og stadig har en tilstrækkelig præcis repræsentation. På den anden side kan der nemt konstrueres eksempler hvor enhver model som er mindre kompleks en den maksimalt komplekse model ikke vil være i stand til at repræsentere fordelingen præcist (Jaeger, 2004; Beygelzimer and Rish, 2003). Sådanne udfordrene eksempler konstrueres typisk ved at definere fordelinger som indeholder *kontekst-specifikke* (u)afhængigheds (eng. *context-specific* (in)dependence eller CSI) relationer. Eksistensen af CSI relationer som ikke er repræsenterbare af en BN model har motiveret udviklingen af udvidelser til den originale BN model som effektivt kan repræsentere sådanne fordelinger. Eksempler herpå er Baysian Multinets (BM) af Geiger and Heckerman (1996), Mixtures of Bayesian Networks (MBN) af Thiesson et al. (1997) og Recursive Bayesian Multinets (RBM) af Peña et al. (2002). Disse er alle variationer af den fælgende fælles arkitektur: en kontekst er defineret af en distingiveret variabel eller mængde af distingiverede variable, og betinget på konteksten eksistere der så en BN model over de resterende variable. I MBN'er er konteksten defineret ved en ikke observeret latent variabel, og i RBM'er er konteksten defineret ved en mængde observerede variable. Algoritmer for generelt at udføre probabilistiske slutninger (som opdatering af tro) i disse modeller kan drage fordel af CSI relationer indkodet

af modellen , men i sidste ende eksistere det generelle problem mht. beregning i BN modeller stadig.

I denne afhandling foreslår vi en procedure til læring af Probabilistike Beslutnings Graf (eng. Probabilistic Decision Graph eller PDG) modeller. PDG sproget er en tilføjelse til den voksende mængde af PGM repræsentations sprog til diskrete fælles sandsynligheds fordelinger (Jaeger, 2004). PDG'er tilbyder både en naturlig tilgang til indkodning af en vis klasse af CSI relationer mellem de observerede variable og tilbyder også effektiv beregning. En særdeles indbydende egenskab ved PDG sproget er at repræsentations strukturen også udgør en primær stuktut til effektiv beregning af generel opdatering af marginale betingede sandsyndligheder. Dette er vigtigt for lærringsprocedure når de lærte modeller senere skal bruges til sådanne opdateringer. I et sådant scenario kan vi umiddelbart adskille modeller mht. beregneligheds kompleksitet ud fra den givne repræsentation. For mange andre relevante PGM sprog er det ikke trivielt at udlede et meningsfuldt mål for kompleksitet af beregnelighed - i særdeleshed er dette tilfældet for BN modeller hvor bestemmelse af beregningskompleksitet involvere et NP-komplet optimerings problem (Arnborg et al., 1987).

Det er ofte nødvendigt at antage at data er komplet i den forstand at der ikke forekommer latente (ikke observerede) variable der påvirker de observerede variable gennem ikke trivielle interaktioner. Dette er dog ofte en meget stærk antagelse og kan være inkonsistent med den generelle forståelse af domænet som domæne eksperter måtte have. Eksistensen af sådanne latente variable kan give en data genererende proces som udviser en mængde relationer af uafhængighed som ikke kan repræsenteres i DAG strukturen af BN modeller. Den eksplicitte genskabelse af sådanne latente variable er en ambitiøs opgave. Ikke desto mindre har mange studier i den seneste tid efterfulgt en løsning til problemer forbundet med læring af latente variable både i en generel DAG struktureret BN model (Elidan, 2004) og også med focus på hierarkiske (træ) strukturer (Karčiauskas, 2005). En træ-struktureret BN model som modellere alle observerede variable som betinget uafhængige givet tilstanden af en enkelt latent variabel (normalt benævnt en Naiv Bayes (NB) model), er blevet studeret omfattende til probabilistisk *blød* klynge-inddeling (eng. clustering) af datapunkter (Duda et al., 2001). Sådanne modeller kan dog også nemt og naturligt anvendes til generel beregning af probabilistiske slutninger. Et forholdsvist nyt studie sammelignede BN of NB modeller og resultaterne faldt ud til NB sprogets fordel mht. beregnings kompleksitet og præcisionen af repræsentationen (Lowd and Domingos, 2005).

I denne afhandling udfører vi en komparativ analyse af forskellige PGM sprog, deres evne til effektivt og præcist at repræsentere en tilnærmelse af en given sandsynlighedsfordeling og vores evne til at lære sådanne modeller fra en endelig database. Sådanne analyser er ikke nye, og we tilføjer derfor blot vores resultater til resultater fra tilsvarende analyser foretaget i tidligere studier så som den komparative analyse af empiriske målinger af effektivitet og præcision af BN og NB modeller af Lowd and Domingos (2005) og det mere teoretiske studie af udvalget af forskellige tilnærmelser som er mulige i BN modeller af Beygelzimer and Rish (2003). I vores analyse anvender vi først som analytisk værktøj SL-kurver. SL-kurver viser en karakteristik af et sprog ved at plotte effektivitet og præcision af modeller fra sproget. Som

et mål for effektivitet bruger vi beregningskompleksitet hvilket er en teoretisk kvantitet, mens vi for præcision bruger sandsyndligheden for den observerede database under antagelse af at den givne model genererede databasen. Dernæst udfører vi en empirisk analyse af effektivitet ved brug af en af de bedste implementationer til beregning af probabilistiske slutninger. Vi inkluderer her også en sammenligning af empiriske målinger af præsision ved et gennemsnit over tilfældigt genererede forespørgsler. Vi inkludere PDG'er i vores komparative analyse, der som sagt er et forholdsvist nyt PGM sprog.

Endelig, som et i nogen grad separat spor, foreslår vi en algoritme til konstruktion af en PDG model fra et klike træ (eng. Clique Tree eller CT) repræsentation af fordelingen. Vi kombinerer BN lærring og PDG lærring ved at konstruere en CT repræsentation of fordelingen repræsenteret af BN modellen, og oversætter så denne CT repræsentation til en ækvivalent PDG model som så bliver udsat for optimerings operationer der kan give en repræsentation som er konkurrencedygtig med den originale BN model. Vi benævner denne tilgang "hybrid lærring" af PDG modeller da den kombinere en lært BN model og dennes CT repræsentation med lærring af en optimeret PDG repræsentation.

C.1 Oversigt over Afhandlingen

I Kapitel 2 giver vi en introduktion til relevant baggrunds materiale og notationelle konventioner der bliver brugt i den resterende del af afhandlingen. Kapitel 3 introducere formelt de PGM repræsentations sprog som vi vil undersøge i den senere analyse. Vi inkludere en diskussion af beregnings kompleksitet af generelle probabilistiske forespørgsler ved for hvert sprog at præsentere procedure til udførsel af eksakt opdatering af alle marginale sandsynligheder betinget på observationer. I Kapitel 4, foreslår vi procedure til lærring af modeller fra data for hvert af de tidligere præsenterede PGM sprog. Kapitel 5 indeholder en beskrivelse af eksperimenter omkring lærring af PGM'er fra data, og vi udfører båden en teoretisk og empirisk komparativ analyse af PGM sprogene mht. de foreslåede lærrings procedure. Yderligere indeholder Kapitel 5 en analyse af hybrid lærring af PDG modeller. Endelig, i Kapitel 6 opsumere vi de vigtige observationer som blev gjort gennem den komparative analyse, og vi diskutere hvilke konklusioner der kan drages på baggrund af det studie som rapporteres i denne afhandling.