VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

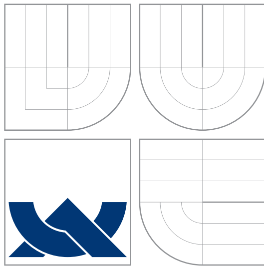
BAYESIAN NETWORKS APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

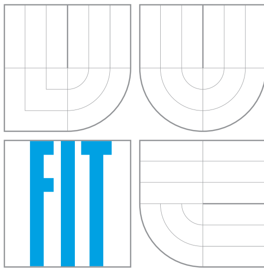
AUTOR PRÁCE
AUTHOR

Bc. DAVID CHALOUPKA

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

APLIKACE BAYESOVSKÝCH SÍTÍ

BAYESIAN NETWORKS APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. DAVID CHALOUPKA
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. FRANTIŠEK V. ZBOŘIL, CSc.
SUPERVISOR

BRNO 2013

// Abstrakt
Tato diplomová práce se zabývá možnými aplikacemi Bayesovských sítí. Nejprve se zaměřuje na obecnou teorii pravděpodobnosti a později na úrovni matematiky vysvětluje samotnou teorii Bayesovských sítí, přístupy k inferenci a k učení včetně ozřejmění silných a slabých stránek popisovaných technik. Součástí výkladu jsou v mnoha případech ilustrativní příklady a podrobně komentovaná matematická odvození prezentovaných vzorců. V praktické části práce je kladen důraz na aplikace vyžadující učení Bayesovské sítě, jednak ve smyslu učení parametrů a jednak ve smyslu struktury. První aplikací jsou obecné benchmarkové úlohy, které zkoumají chování prezentovaných technik a zaměřují se na způsob optimální volby parametrů učení Bayesovské sítě. Druhou aplikací je užití Bayesovských sítí pro účely dolování znalostí o příčinách zločinnosti prostřednictvím vizualizace závislostí mezi náhodnými proměnnými popisujícími zkoumanou doménu. Třetí aplikace zkoumá možnosti nasazení Bayesovské sítě jakožto spam filtru a dosažené výsledky porovnává prostřednictvím všeobecně užívané datové sady s výsledky naivního Bayesovského filtru, který rovněž vychází z teorie pravděpodobnosti.

Abstract
This master's thesis deals with possible applications of Bayesian networks. The theoretical part is mainly of mathematical nature. At first, we focus on general probability theory and later we move on to the theory of Bayesian networks and discuss approaches to inference and to model learning while providing explanations of pros and cons of these techniques. The practical part focuses on applications that demand learning a Bayesian network, both in terms of network parameters as well as structure. These applications include general benchmarks, usage of Bayesian networks for knowledge discovery regarding the causes of criminality and exploration of the possibility of using a Bayesian network as a spam filter.

Klíčová slova
Bayesovská síť, pravděpodobnost, stochastická inference, učení struktury, strojové učení, spam.

Keywords
Bayesian network, probability, stochastic, inference, structure learning, machine learning, spam.

Citace
David Chaloupka: Bayesian Networks Applications, diplomová práce, Brno, FIT VUT v Brně, 2013

Bayesian Networks Applications

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. Františka V. Zbořila, CSc.

....
David Chaloupka
16. května 2013

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu, docentu Františku V. Zbořilovi, za cenné rady, odborné vedení a vstřícný přístup v nesnázích při řešení této práce.

© David Chaloupka, 2013.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Contents

1 Introduction 3

2 Preliminaries 5
- 2.1 Philosophical views . . . 5
- 2.2 Probability distributions . . . 6
- 2.3 Factors . . . 6

3 Theory of Bayesian networks 8
- 3.1 Flow of probabilistic influence . . . 10
- 3.2 Inference in Bayesian networks . . . 12
  - 3.2.1 Sum-product . . . 12
  - 3.2.2 Variable elimination . . . 13
  - 3.2.3 Belief propagation . . . 14
  - 3.2.4 Sampling methods . . . 14
  - 3.2.5 Markov Chain Monte Carlo . . . 17
- 3.3 Model learning . . . 18
- 3.4 Parameter estimation . . . 19
  - 3.4.1 Maximum likelihood estimation . . . 19
  - 3.4.2 Bayesian estimation . . . 20
- 3.5 Structure learning . . . 22
  - 3.5.1 Optimization algorithm . . . 22
  - 3.5.2 Likelihood score . . . 23
  - 3.5.3 BIC score . . . 24
  - 3.5.4 Bayesian score . . . 25
  - 3.5.5 Learning specific structures . . . 28
  - 3.5.6 Method for finding all possible alterations of a network . . . 29
- 3.6 I-equivalent structures . . . 30

4 Implementation 32
- 4.1 Package structure . . . 32
- 4.2 Data structures . . . 32
- 4.3 Exceptions . . . 33
- 4.4 Input, output . . . 33
- 4.5 Sampling . . . 34
  - 4.5.1 Producing a single sample . . . 34
  - 4.5.2 Friendly query specification . . . 36
  - 4.5.3 Processing a larger number of samples . . . 37
  - 4.5.4 Implementation notes . . . 38

4.6 Structure learning . . . 40
4.6.1 Learning algorithm . . . 40
4.6.2 Scoring functions . . . 40
4.6.3 Implementation notes . . . 41
4.6.4 Dataset . . . 41
4.7 User interface . . . 43
4.7.1 Network layout . . . 43
4.7.2 Third party packages . . . 44

5 Selected applications **45**
5.1 Learning of a known model . . . 45
5.1.1 Learning parameters . . . 45
5.1.2 Learning structure . . . 48
5.2 Crime, its causes and countermeasures . . . 55
5.2.1 Data preprocessing . . . 55
5.2.2 Experiments . . . 56
5.2.3 Conclusion . . . 59
5.3 Spam filtering . . . 60
5.3.1 Overview of document classification . . . 60
5.3.2 Dataset and data preprocessing . . . 61
5.3.3 Experiments . . . 61
5.3.4 Conclusion . . . 63

6 Conclusion **64**

A Notation overview **67**

B CD Content **68**

C Crime analysis – final networks **69**

Chapter 1

Introduction

Bayesian networks are a subclass of probabilistic graphical models and serve as a tool for modeling joint probability distributions of random variables. Bayesian networks were originally used to evaluate medical data of a patient in order to determine the probability of him having a certain disease which is a situation with many unknowns, eg. missing medical history or unknown results of medical tests that haven't been performed. Since then there have been many other applications such as inspection of pedigree trees in genetics or discovery of protein signaling mechanisms based purely on statistical data.

The main advantages of Bayesian networks lie in the fact that they are easy to interpret, even for a non-specialist, and in their inherent property that they can represent joint probability distributions in a very compact way in terms of space complexity. Construction of a Bayesian network for some concrete domain is a complicated task demanding complex algorithms, enough input data and, in many cases, expert knowledge. Another problem is providing accurate answers to queries regarding the probability distribution induced by a Bayesian network, so called inference.

The aim of this thesis is to present techniques of learning Bayesian networks, both in terms of conditional probability tables, ie. model parameters, as well as in terms of structure, and to demonstrate possible applications of these networks. Inference techniques are narrowed down to stochastic approaches because other ways are complex even in the most basic form. Still, all approaches to inference will be mentioned at least on the conceptual level since it is a fundamental part of the theory of Bayesian networks.

Chapter 2 serves as an introduction to the fundamentals of the probability theory and provides necessary foundations for understanding the rest of this thesis, including the used mathematical notation.

Chapter 3 discusses the theory of Bayesian networks. First, the motivation for using Bayesian networks is explained, followed by an overview of inference methods and by a detailed analysis of model learning techniques in terms of parameters as well as structure.

In chapter 4 I will introduce the realization output of this thesis, focusing on the overall design and philosophy. This chapter should be partly seen as program documentation.

Chapter 5 discusses selected applications of parameter and structure learning of Bayesian networks. The first application is a study of optimal parameter settings of the model learning procedures for networks and datasets of various sizes. The second application uses model learning as a tool to discover the main causes of criminality. The third application explores the option of using a generalized Naïve Bayes model for spam filtering.

Appendix A provides an overview of the mathematical notation used in this thesis.

Appendix B describes content of the CD enclosed with this thesis.

Appendix C contains folded A3 figures of Bayesian networks from Section 5.2.

During the winter semester I focused on studying the theory of Bayesian networks and a major part of Chapters 2 and 3 of this thesis has been the core of the term project. In the summer semester I implemented the presented techniques, first as a stand-alone library and then as a GUI application based on this library. With this application were conducted the experiments presented in Chapter 5. Also, during the summer semester some of the theoretical parts had to be revisited, extended and, according to comments of my supervisor, the more difficult ones were supplemented by illustrative examples to make them easier to grasp.

Chapter 2

Preliminaries

This chapter will present theoretical foundations for understanding Bayesian networks and for performing computation over them. Studying Bayesian networks (further abbreviated as BNs) is challenging both because of necessary mathematical rigor as well as for the need of efficient non-trivial algorithms to construct them and to provide answers for probability queries. In case of some formulas their mathematical derivation will be presented because, I believe, it provides useful insight into the underlaying theory. In other cases we will jump straight to the practical conclusion needed in order to algorithmically solve the problem at hand because the derivation is either very difficult or the process itself is uninteresting for a non-mathematician and would unnecessarily prolong the thesis or even obfuscate the topic.

First will be established notation used in this thesis and necessary overview of probability theory. Then we will move on to the theory of Bayesian networks, explain their advantages and proceed to inference and model learning techniques.

2.1 Philosophical views

Probability of an event is usually interpreted as the degree of belief in occurrence of that particular event, eg. probability of a fair die rolling a six is 1/6. Although probability is expressed as a real number between zero and one, the ground truth is that in the end one and only one *concrete* event is observed. In the case of a die, one concrete number is rolled regardless what the probabilities were. Although our intuition and general experience proves it to be true in our macroscopic world, objects in the microscopic world of quantum mechanics actually are probabilistic in nature and, a particle for example, is in many quantum states at the same time. The particle is then described by a complex *wave function* $\Psi(x, y, z, t)$ and $|\Psi(x, y, z, t)|^2$ describes *density probability* of the particle as a function of space-time coordinates [7, p. 1044]. Despite the obvious differences, certain qualities of both these worlds can be described by the same mathematical apparatus which is probability theory.

Somewhat close to probability theory is the theory of fuzzy sets which operates with membership functions. These functions describe degrees of truth, eg. that a person is regarded as young or old. The main difference, when compared to probability, is that in fuzzy sets theory an object may have more than one quality to some degree and there is no ground truth or observation that would strictly place a person into one single category.

2.2 Probability distributions

Let's begin by defining what a probability distribution is when there is only one single random variable.

Definition 2.1. *$P(X)$ is discrete probability distribution of discrete random variable X such that:*

- *X can be assigned any value from the set $val(X) = \{x_1, x_2, \dots\}$ which is finite or denumerable.*
- $\forall x_i \in val(X) : P(X = x_i) \geq 0$
- $\left(\sum_i P(X = x_i)\right) = 1$

Notice that random variables are denoted by capital letters X, Y, E etc., concrete values of random variables (assignments or instantiations) are denoted by small letters such as x. Further we will use bold capital letters to denote sets of random variables such as $\mathbf{X}$ and their instantiation with bold small letters such as $\mathbf{x}$.

Probability distributions of more than one random variable are called *joint probability distributions*. For example if we were to study a simultaneous throw with two dices, black and white, we would get joint probability distribution $P(X_{black}, X_{white})$. In this probability distribution we know how likely every single outcome of a throw is. In this case there are $6 \times 6 = 36$ possibilities so the probability distribution $P(X_{black}, X_{white})$ would have 36 entries if represented by a table.

So far we have described so called *prior probabilities* which are applicable in situations when no observation has been made and hence everything is governed purely by probability. If, on the other hand, we observe that some random variable takes on a concrete value (eg. the weather today is sunny) then we speak about *posterior probability* or *conditional probability* distribution. Posterior and prior probability distributions are not the same in general because observation of a variable may affect distribution of other variables as we will see later in context of flow of probabilistic influence in Bayesian networks. Conditional probability distribution of variables $\mathbf{X}$ depending on variables $\mathbf{E}$ is denoted $P(\mathbf{X} \mid \mathbf{E})$ and such expression is usually read as "probability of $\mathbf{X}$ given $\mathbf{E}$" where $\mathbf{E}$ are called *evidence* or *observed* variables. Conditional probability distribution can be seen as a collection of probability distributions over variables $\mathbf{X}$ for every possible instantiation $\mathbf{e}$ of variables $\mathbf{E}$.

2.3 Factors

When reasoning about Bayesian networks we express probability distributions as so called *factors*. Factors are interesting for us because factor operations, among other uses, correspond to mathematical operations we need to perform with probability distributions in order to answer queries in a Bayesian network.

Definition 2.2. *Factor ϕ is a function $\phi : X_1 \times X_2 \times \cdots \times X_k \to \mathbb{R}_0^+$. The non-empty set $\{X_1, X_2, \dots, X_k\}$ is called scope of the factor ϕ.*

According to the definition of a factor and to the definition 2.1 of a discrete probability distribution, probability distributions are special cases of factors. They both require their values to be non-negative and, in addition, every probability distribution must sum to one. A factor, of course, may sum to one too in which case we say it is *normalized*.

Factor representation of a joint probability distribution $P(X, \dots, Z)$ is straightforward – simply put $\phi(x, \dots, z) = P(x, \dots, z)$ for every assignment $x, \dots, z$ of variables $X, \dots, Z$. In this case probabilities of all possible assignments must necessarily sum to one since $P(X, \dots, Z)$ is a probability distribution. For further reference, this summation is denoted $\sum_{X,\dots,Z} P(x, \dots, z) = 1$.

Let's examine the case of a conditional probability distribution $P(\mathbf{X} \mid \mathbf{E})$. Suppose that, according to the established notation, $\mathbf{X}$ and $\mathbf{E}$ are sets of variables rather than single variables. Then $scope(\phi) = \mathbf{X} \cup \mathbf{E}$ but probabilities of all possible assignments to $\mathbf{X}$ and $\mathbf{E}$ doesn't sum to one! This is because the conditional probability distribution $P(\mathbf{X} \mid \mathbf{E})$ is more like a collection of probability distributions of variables $\mathbf{X}$ for every assignment to $\mathbf{E}$. So, for any concrete assignment $\mathbf{e}$ of evidence variables $\mathbf{E}$, the probability distribution $P(\mathbf{X} \mid \mathbf{e})$ again sums to one, ie. $\sum_{\mathbf{X}} P(\mathbf{x} \mid \mathbf{e}) = 1$.

There are 4 operations over factors we will need further in this thesis – pointwise product, marginalization, conditioning and renormalization. Formal definitions below are derived from description of these operations in [15, 12]

Pointwise product

Pointwise product of factors ϕ_1 and ϕ_2 is a factor ψ, denoted $\psi = \phi_1 \cdot \phi_2$, defined as follows. The scope of ψ is $scope(\phi_1) \cup scope(\phi_2)$. Let $scope(\phi_1) = \{X_1, \dots, X_m, Y_1, \dots, Y_n\}$ and $scope(\phi_2) = \{Y_1, \dots, Y_n, Z_1, \dots, Z_k\}$ where $\{X_1, \dots, X_m\}$ and $\{Z_1, \dots, Z_k\}$ are disjoint. For every assignment $x_1, \dots, x_m, y_1, \dots, y_n, z_1, \dots, z_k$ of variables in $scope(\psi)$ the value of $\psi(\cdot)$ is given as:

$$\psi(x_1, \dots, x_m, y_1, \dots, y_n, z_1, \dots, z_k) = \phi_1(x_1, \dots, x_m, y_1, \dots, y_n) \cdot \phi_2(y_1, \dots, y_n, z_1, \dots, z_k)$$

Please note that it is also possible for the sets $\{X_i\}_i, \{Y_i\}_i, \{Z_i\}_i$ to be empty (of course, according to the definition 2.2, any factor must have at least one variable in its scope).

Marginalization

Marginalization of a factor ϕ over variables $\mathbf{Y}$ is operation of summing out these variables, effectively producing factor ψ. Then $scope(\psi) = scope(\phi) \setminus \mathbf{Y}$ and for every assignment $\mathbf{x}$ of variables in $scope(\phi)$ holds that $\psi(\mathbf{x}) = \sum_{\mathbf{Y}} \phi(\mathbf{x}, \mathbf{y})$.

Conditioning

Conditioning is operation used when there is some observed evidence $\mathbf{e}$ of variables $\mathbf{E}$. By conditioning we set probabilities of all events inconsistent with observed evidence to 0. Of course, the resulting factor will not be normalized in general.

Renormalization

Renormalization of a factor ϕ ensures that all values of the factor sum to one. Suppose that $\mathbf{X} = scope(\phi)$ and let $\alpha = \sum_{\mathbf{X}} \phi(\mathbf{x})$ be normalizing constant. Values of the resulting factor ψ are given for every assignment $\mathbf{x}$ of variables $\mathbf{X}$ as $\psi(\mathbf{x}) = \phi(\mathbf{x})/\alpha$.

It may be useful to point out that renormalization makes sense only for factors that don't represent conditional probability distributions.

Chapter 3

Theory of Bayesian networks

In this chapter we will first discuss the reasons for using Bayesian networks. Then we will move on to various inference methods and to methods of model learning, both in terms of model parameters as well as in terms of structure.

Let's ask the question why to use Bayesian networks? What do they bring us? First of all, having a complicated joint probability distribution with many random variables inevitably leads to an enormously large table that would represent such probability distribution. Let's assume we have binary random variables $X_1, \ldots, X_k$. Then to represent the joint probability distribution $P(X_1, \ldots, X_k)$ we would need a table with 2^k entries. This exponential growth brings several problems:

- Table completely describing a joint probability distribution may be too big to store. That is because the table would hold probability of every possible event separately in its own record.
- Even if the table could be stored, performing calculations over it (e.g. factor marginalization or factor multiplication) would not be efficient. In fact, operations such as exact inference are known to be NP-hard in the number of variables [12].
- In order to construct probability table for a joint probability distribution:
  a) We would require a huge amount of training data to create an accurate statistical model from, since we would essentially count occurrences of every single event (ie. of *each possible* instantiation of all random variables) and finally divide these counts by the total number of all training examples.
  b) We would need a human expert to determine the probability of every possible assignment of random variables. This is generally not possible because all probabilities would be near to zero and human experts are simply not able to correctly capture probabilities on this level [12].

As will be shown later, Bayesian networks couple with the problem of exponential growth of record counts in probability tables by having several smaller local conditional probability distributions (factors) based on dependencies between random variables. Product of all these conditional probabilities (factors) in the entire network gives the underlaying joint probability distribution which would, if represented naively, require an exponentially large table.

Definition 3.1. *Bayesian network is a tuple* $(\mathcal{G}, \Theta)$. $\mathcal{G}$ *is a directed acyclic graph where each node represents a random variable and oriented edges between nodes express direct dependencies between random variables represented by the connected nodes.* Θ *are parameters of the network representing for each node* X *its probability distribution conditioned on its immediate parents, denoted* $P(X \mid Parents(X))$.

The definition of Bayesian network (further abbreviated BN) implies several things. Acyclic and directed properties tell us that there is a hierarchy of nodes in terms of parent-child or predecessor-successor relation, meaning that a random variable C is dependent on its parent random variables $\{P_1, \ldots, P_m\}$ (denoted $Parents(C)$). This conditional probability distribution $P(C \mid Parents(C))$ is usually[1] expressed via a *Conditional Probability Table (CPT)* that defines probability distribution of variable C for every possible assignment of variables $Parents(C)$. Of course, for every assignment $p_1, \ldots, p_m$ of variables $Parents(C)$ must hold that $\sum_C P(c \mid p_1, \ldots, p_m) = 1$. Otherwise $P(C \mid Parents(C))$ would not be a probability distribution (by definition 2.1).

We say that Bayesian network $(\mathcal{G}, \Theta)$ with nodes $\{X_1, \ldots, X_k\}$ induces joint probability distribution $P(X_1, \ldots, X_k)$ as follows:

$$P(X_1, \ldots, X_k) = \prod_{i=1}^{k} P(X_i \mid Parents(X_i)) \tag{3.1}$$

Equivalent statement is that $P(X_1, \ldots, X_k)$ factorizes over a BN $(\mathcal{G}, \Theta)$ if the equation (3.1) holds. This is because the terms $P(X_i \mid Parents(X_i))$ are *factors* and by constructing a Bayesian network we can factorize otherwise very space-consuming CPD of $P(X_1, \ldots, X_k)$ into several smaller CPDs $P(X_i \mid Parents(X_i))$, one for every node X_i.

I have devised a proof that $P(X_1, \ldots, X_k)$ induced by a BN is indeed a probability distribution, when assuming that every factor $P(X_i \mid Parents(X_i))$ is also a probability distribution. Main purpose of this proof is to provide the reader with some familiarity when mathematically reasoning about BNs because similar tricks will be used later on. Core of the proof is to show that both axioms of a probability distribution (Definition 2.1) are satisfied[2]:

1. $P(X_1, \ldots, X_k) \geq 0$: Trivial since factors $P(X_i \mid Parents(X_i))$ are assumed to be valid probability distributions. Then, by definition of probability distribution 2.1, we know that $\forall i : P(X_i \mid Parents(X_i)) \geq 0$. And product of non-negative numbers (factors) is also non-negative, ie. $\prod_{i=1}^{k} P(X_i \mid Parents(X_i)) \geq 0$.

2. $\sum P(x_1, \ldots, x_k) = 1$: If the remaining variables $X_1, \ldots, X_k$ are all independent, then by definition of independent random variables $P(X_1, \ldots, X_k) = P(X_1) \cdots P(X_k)$. So the sum can be written as $\sum P(x_1, \ldots, x_k) = \sum_{X_1} P(x_1) \sum_{X_2} P(x_2) \cdots \sum_{X_k} P(x_k)$. By assumption, all factors are valid probability distributions, so $\sum_{X_k} P(x_k) = 1$. Then $\sum_{X_1} P(x_1) \sum_{X_2} P(x_2) \cdots \sum_{X_k} P(x_k) = \sum_{X_1} P(x_1) \sum_{X_2} P(x_2) \cdots 1$. Thus we have eliminated variable X_k and transformed the problem to $\sum P(x_1, \ldots, x_{k-1}) = 1$. By eliminating all remaining variables the same way, we get $\sum P(x_1, \ldots, x_k) = 1$.

   If there is a direct dependency between some two variables let's relabel the variables so that their indices correspond to a topological sort of the given BN (every BN is a DAG,

[1] For purposes of this thesis, we assume probability distributions of discrete, not continuous, variables whose events are from a finite set. Such probability distribution can be expressed by a table.

[2] Because of long expressions in the following two paragraphs, $Pars(X)$ denotes $Parents(X)$.

so a topological sort exists). Then we can rewrite the inspected summation in the form $\sum P(x_1, \ldots, x_k) = \sum_{X_1} P(x_1|Pars(X_1)) \sum_{X_2} P(x_2|Pars(X_2)) \cdots \sum_{X_k} P(x_k|Pars(X_k))$ because, thanks to the topological sort, all parent variables of a variable X_i must have smaller index that i. Also note that the maximal element X_k has no child. By assumption, $P(X_k \mid Pars(X_k))$ is a probability distribution, so $\sum_{X_k} P(x_k \mid Pars(X_k)) = 1$. By substituting the term $\sum_{X_k} P(x_k \mid Pars(X_k))$ in the summation we eliminate the variable X_k and transform the problem to question if $\sum P(x_1, \ldots, x_{k-1}) = 1$ which we approach recursively.

3.1 Flow of probabilistic influence

To describe how observing certain facts propagates through a Bayesian network and how it affects probability distributions of unobserved variables on qualitative level, we introduce *flow of probabilistic influence*. Flow of influence also enables us to formally capture conditional dependency and/or independency of random variables in a BN given some facts.

Definition 3.2. *Let $(\mathcal{G}, \Theta)$ be a Bayesian network with nodes $X_1, X_2, \ldots, X_k$ and let $\mathbf{Z}$ be a set of observed variables, where $\mathbf{Z} \subseteq \{X_1, \ldots, X_k\}$. A connected undirected trail $X_a - X_{a+1} - \ldots - X_b$ given facts $\mathbf{Z}$ is active if and only if*

- *For every so called "V-structure" $X_{i-1} \rightarrow X_i \leftarrow X_{i+1}$ on the trail is $X_i \in \mathbf{Z}$.*
- *No other node of given trail is in $\mathbf{Z}$.*

Notes: V-structure is defined as a segment of trail in form $X_{i-1} \rightarrow X_i \leftarrow X_{i+1}$ (in this case, edge directions do matter). Also note that the trail may contain duplicate nodes.

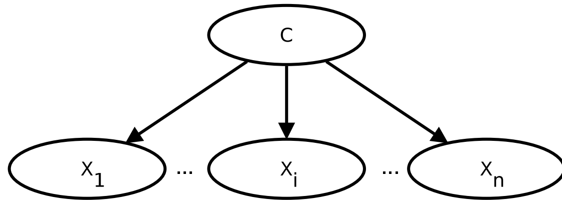
Definition 3.3. *Let $(\mathcal{G}, \Theta)$ be a Bayesian network, X, Y two of its nodes and $\mathbf{Z}$ observed variables. We say that X, Y given $\mathbf{Z}$ are d-separated in $\mathcal{G}$ iff there is no active trail between X and Y given $\mathbf{Z}$.*

Note: X and Y being d-separated given $\mathbf{Z}$ is equivalent to X and Y being conditionally independent given $\mathbf{Z}$, denoted $X \perp Y \mid \mathbf{Z}$.

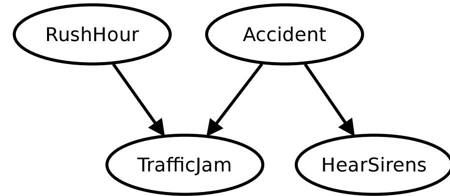
By using d-separation (direction-dependent separation) we can quite elegantly determine if observation of a variable X can or cannot affect distribution of some other random variable. Clearly, two random variables that are directly connected are dependent. But we can also examine other than these intuitively obvious relations. Also, please note that whether two random variables X, Y are independent in Bayesian network depends on our observations $\mathbf{Z}$ as it can activate some trails in V-structures just as it can deactivate direct segments of other trails.

To understand better the notion of active trails and d-separation we will refer to examples in Figure 3.1. In Figure 3.1a you can see a Naïve Bayes model in which all feature variables X_i and X_j are conditionally independent given class variable C because knowing the class variable deactivates all trails between X_i and X_j. In Figure 3.1b we can see an activated V-structure $R \rightarrow T \leftarrow A$, provided the variable T is observed. Intuitively, when we know whether there is a traffic jam then an accident decreases probability of being it in a rush hour because an accident is sufficient to cause a traffic jam by itself, formally $R \not\perp A \mid T$. To go one step further, if it is not a rush hour then knowing there is a traffic jam increases the probability of there being an accident which causally means higher probability of hearing sirens, hence $R \not\perp H \mid T$. The argumentation involving activated

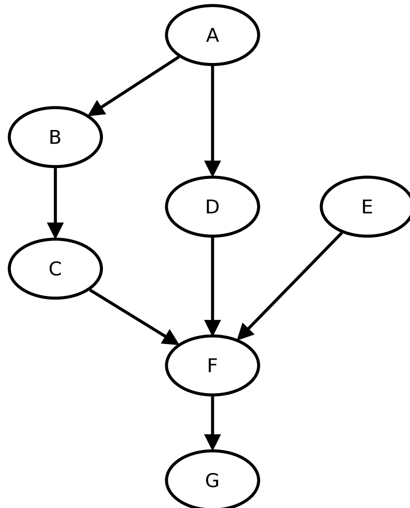
V-structures is an example of *inter-causal* reasoning (causes with common effect are conditionally dependent given the effect variable). In Figure 3.1c we will examine three more complicated cases of interest. The first selected case is $A \not\perp C \mid B, G$ because of the active trail $A - D - F - G - F - C$ (node F is twice on the trail which is perfectly legal). The second selected case is $A \perp F \mid B, D$ because both possible trails between A and F are deactivated. The third selected case is $E \not\perp B \mid F, C$ because there is an active trail $E - F - D - A - B$.



(a) General Naïve Bayes model in which holds $X_i \perp X_j \mid C$ for $i \neq j$



(b) Example of an activated V-structure, $R \not\perp H \mid T$ (active trail $R - T - A - H$)



(c) A more complicated network with multiple possible trails for some pairs of variables and other interesting features.

Figure 3.1: Networks for demonstration of d-separation and flow of influence.

It might be useful to explicitly point out what the notation $P(X \mid Parents(X))$ tells us in context of d-separation. It is a fact frequently used in mathematical derivations in context of BNs that, when given all parents $Parents(X)$ of a variable X, the variable is independent of all its indirect predecessors in the network. In Markov chain Monte-Carlo sampling we will also introduce so called *Markov blanket* of variable X which is minimal set of variables such that X is conditionally independent on any other variable in the network given its Markov blanket.

3.2 Inference in Bayesian networks

There are several types of queries that might be asked in context of a Bayesian network. In my case, inference in BNs is primarily understood as computing the posterior probability distribution for a set of query variables $\mathbf{X}$ given values $\mathbf{e}$ of evidence variables $\mathbf{E}$. Mathematically, the task is to compute the probability distribution $P(\mathbf{X} \mid \mathbf{e})$. Another type of query is so called *maximum a posteriori* (abbreviated as MAP) which is finding $arg\ max_{\mathbf{X}} P(\mathbf{X} \mid \mathbf{e})$. MAP is basically used for finding out the most probable cause (concrete assignment of variables $\mathbf{X}$) for observation $\mathbf{e}$. A practical example could be getting the most probable word (sequence of letters) that corresponds to classifier output for a segmented picture containing that word. Although, this is an example more frequently used in context of Markov networks.

Inference in a BN may be carried out in many ways depending on nature of the query and precision requirements. Exact inference is relatively straightforward in terms of mathematical description and naive algorithmic implementation but exhibits serious running-time disadvantage – exponential running time in the number of variables in the worst case. On the other hand, approximate inference methods allow us to answer queries more quickly but precision of the results may be questionable, especially for very rare events.

3.2.1 Sum-product

At the conceptual level, the simplest method for computing a conditional probability query is called sum-product. The problem at hand is to compute conditional probability distribution $P(\mathbf{X} \mid \mathbf{e})$ where $\mathbf{X}$ is one or more variables, $\mathbf{e}$ is an assignment to evidence variables $\mathbf{E}$ and $\mathbf{Y}$ are variables that are neither query nor evidence variables (the rest of the variables in the BN). The query is computed according to Bayes' theorem as follows:

$$P(\mathbf{X} \mid \mathbf{e}) = \frac{P(\mathbf{X}, \mathbf{e})}{P(\mathbf{e})} = \frac{\sum_{\mathbf{Y}} P(\mathbf{X}, \mathbf{y}, \mathbf{e})}{\sum_{\mathbf{X},\mathbf{Y}} P(\mathbf{x}, \mathbf{y}, \mathbf{e})}$$

The denominator is constant and since $P(\mathbf{X} \mid \mathbf{e})$ has to be a probability distribution, we can simply omit the denominator and in the end normalize numerator by an appropriate constant α. Hence we can write:

$$P(\mathbf{X} \mid \mathbf{e}) = \alpha \sum_{\mathbf{Y}} P(\mathbf{X}, \mathbf{y}, \mathbf{e})$$

Computation of this expression using the sum-product method means performing four steps:

1. reducing all factors in the Bayesian network by evidence $\mathbf{e}$,
2. multiplying reduced factors all together producing single joint factor $P(\mathbf{X}, \mathbf{Y}, \mathbf{e})$ which is not a probability distribution because, in general, all values of the factor don't sum to one,
3. marginalizing over the variables $\mathbf{Y}$ (non-query and non-evidence variables) and finally
4. renormalizing the resulting factor, obtaining the probability distribution $P(\mathbf{X} \mid \mathbf{e})$.

Every item of the sum in the numerator, ie. probability of every possible assignment to variables $\mathbf{Y}$, can be written, according to the joint probability distribution represented by a BN (definition (3.1)), as a product of conditional probabilities of all variables in given BN.

This explains the name of this inference method – we effectively compute many products of conditional probabilities and sum over variables $\mathbf{Y}$.

Sum-product inference algorithm has, by its nature, exponential running time in the number of variables because it effectively generates the whole joint probability distribution encoded by a given Bayesian network and the size of a table representing any joint probability distribution has been earlier shown to be exponential in the number of variables. Because of this quality, the sum-product method is suitable only for inference in considerably small networks.

3.2.2 Variable elimination

Variable elimination algorithm can be seen as a somewhat smarter implementation of the sum-product algorithm explained above. Again, we answer the given query by multiplying all factors of the BN and summing out $\mathbf{Y}$ variables. The innovative ideas behind variable elimination are:

- Suppose we sum over many variables in a product of factors, say one of these variables is V. Then factors whose scopes don't contain V are irrelevant for the summation over V and hence these factors can be factored out in front of the summation over V. This way we can push all summations "as far to the right as possible", so that each summation is ideally done only over product of factors containing the variable V which is being summed out. What is the benefit for us? It can be shown [12] that running time of variable elimination depends on the size of the largest immediate factor during the summing-out process. When we sum only over product of factors containing variable V, the resulting immediate factor is the smallest possible. Because problem of determining the optimal order of elimination is NP-hard, greedy heuristics are usually used instead [12].

- By definition of a conditional probability distribution it holds for any instantiation of parent variables that $\sum_V P(v \mid Parents(V)) = 1$. So, when computing posterior probability distribution $P(\mathbf{X} \mid \mathbf{e})$, any node (factor) V that is not an ancestor of a query or evidence variable can be omitted because, thanks to the order of elimination of variables, the term $\sum_V P(v \mid Parents(V))$ will be the rightmost summation (the innermost). So summing out V is equivalent to totally excluding variable V from the BN. Recursive implementation of this process could be done in the way of eliminating all leaf nodes that are not in $\mathbf{X} \cup \mathbf{E}$ and performing the same operation for as long as some node gets removed [15, p. 510].

Running time of the variable elimination algorithm in polytrees is $O(n)$ [15, p. 510], therefore it is a good method for answering individual queries $P(X \mid \mathbf{e})$ where X is a single variable. Analogically, we can effectively compute probability distribution for small number of query variables $\mathbf{X}$ by computing individual distributions $P(X_i \mid \mathbf{e})$ for all $X_i \in \mathbf{X}$ and returning $P(\mathbf{X} \mid \mathbf{e}) = \prod_i P(X_i \mid \mathbf{e})$.

Running time in multiply connected networks is exponential in general but it can be coped with by clustering algorithms that transform multiply connected network into a polytree which can be processed in linear time [15].

3.2.3 Belief propagation

Belief propagation [12] is an approximate inference algorithm from the family of so called *message passing algorithms*. The main idea is that nodes in the BN exchange information about their believes (probability distributions affected by evidence) and the whole network converges to some result close to true posterior probability. As methods from this family are far from trivial and they are not the aim of this thesis, we will not pay deeper attention to them.

3.2.4 Sampling methods

Monte Carlo methods (or particle methods) are stochastic methods of inference based on sampling the probability distribution induced by a BN. There are three main sampling methods: *direct sampling*, *rejection sampling* and *likelihood weighting*. All these methods basically generate a sufficient number of mutually independent samples and then compute simple statistics over these samples.

Direct sampling

Direct sampling is a method for computing probability distribution in the form $P(\mathbf{U} \mid \mathbf{V})$, ie. without any observed evidence. Of course must hold that $\mathbf{U} \neq \emptyset \wedge \mathbf{U} \cap \mathbf{V} = \emptyset$. For the sake of clarity of formulas we will consider $\mathbf{X} = \mathbf{U} \cup \mathbf{V}$ and compute the probability distribution $P(\mathbf{X})$ from which the conditional distribution $P(\mathbf{U} \mid \mathbf{V})$ can be easily obtained using Bayes' rule. This simplification is applied to all sampling methods further described.

Direct sampling produces a number of samples, ie. of concrete events according to the probability distribution induced by the given BN, and then for every assignment $\mathbf{x}$ computes its probability as $P(\mathbf{x}) = N_{\mathbf{x}}/N$, where $N_{\mathbf{x}}$ is the number of samples in which variables $\mathbf{X}$ have the value $\mathbf{x}$ and N is the number of all samples generated.

A single sample is generated as follows. First, randomly assign values to variables with no parents according to their probability distributions. Then choose a variable V such that all of its parents have already been instantiated and randomly assign a value to V according to the probability distribution $P(V \mid parents(V))$ where $parents(V)$ are the concrete values of the variables $Parents(V)$ in the current sample. Repeat this step until all variables have been instantiated, thereby obtaining a sample. The process be done efficiently if we first compute topological sort of the BN and then sample variables in ascending order.

It can be shown [12, p. 491] that to obtain an estimate with error bounded by ϵ with probability at least $1 - \delta$ we need to generate M samples:

$$M \geq \frac{\ln(2/\delta)}{2\epsilon^2}$$

Problem is that for a very unlikely event we may not generate a single sample and therefore the obtained probability distribution would indicate that this event is impossible (its probability equals to zero). Still, it would be a sound result within the error estimate (ϵ, δ).

Rejection sampling

Direct sampling presented earlier doesn't allow for any evidence. Rejection sampling could be viewed as a simple extension of direct sampling for computing posterior probability distribution with evidence $\mathbf{E} = \mathbf{e}$, ie. distribution in the form $P(\mathbf{X} \mid \mathbf{e})$. The idea is to exclude

samples inconsistent with evidence $\mathbf{e}$ meaning that when an evidence variable is instantiated during the production of a sample, the sample is discarded unless the randomly assigned value corresponds with the observed value in $\mathbf{e}$. The desired distribution is determined by computing $P(\mathbf{x} \mid \mathbf{e}) = N_{\mathbf{x},\mathbf{e}}/N_{\mathbf{e}}$ for every instantiation of $\mathbf{X}$ where $N_{\mathbf{e}}$ basically is the number of all generated samples that haven't been discarded.

This simple approach bears the disadvantage that if the observed evidence is very unlikely then many samples get discarded and hence a lot of computational time is wasted because rejected samples don't contribute to the final probability distribution estimate. Unfortunately, when considering somewhat uniform distribution, the probability $P(\mathbf{e})$ decreases exponentially with the number of observed variables (eg. with symptoms of a patient) and by the same rationale the number of samples that are not rejected decreases just as rapidly.

Likelihood weighting

Rejection sampling described earlier can theoretically be used for answering queries of the form $P(\mathbf{X} \mid \mathbf{e})$ but the number of rejected samples may be unbearable in practice. The likelihood weighting copes with this problem by forcing every sample to be consistent with observed evidence, ie. by artificially setting variables $\mathbf{E}$ to $\mathbf{e}$. Then, of course, not every sample is equally likely, so we cannot determine $P(\mathbf{x} \mid \mathbf{e})$ by simply dividing number of samples $N_{\mathbf{x},\mathbf{e}}$ by $N_{\mathbf{e}}$. For each sample we need some additional information which is the probability of assigning values $\mathbf{e}$ to evidence variables given values of other variables in the current sample [15, p. 514]. Let $\mathbf{Y}$ denote set of all variables in a BN other than the variables $\mathbf{X} \cup \mathbf{E}$. Then we can state that weight of a sample $\mathbf{x}, \mathbf{e}, \mathbf{y}$ is $w_{\mathbf{x},\mathbf{e},\mathbf{y}} = \prod_{\mathbf{E}} P(e \mid parents(E))$ where $parents(E_i)$ denotes assignment to variables $Parents(E_i)$ in the given sample $\mathbf{x}, \mathbf{e}, \mathbf{y}$.

The algorithm of direct sampling needs to be changed in the following way. First of all, the counters of samples won't be integral but real as we don't count mere number of occurrences of a sample but its weighted number of occurrences. Also, when generating a new sample we will keep not only the values of variables but also weight of the sample. When generating a new sample, first we set the weight of the current sample to 1. Then we sample all variables in topological order. For a non-evidence variable X_i we proceed as in the case of direct sampling – randomly assign a value x_i according to the probability distribution $P(X_i \mid parents(X_i))$ which is governed by the current instantiation of X_i's parents. When we encounter an evidence variable E_i, we set it to value e_i according to the given evidence $\mathbf{e}$ and multiply weight of the current sample by $P(e_i \mid parents(E_i))$. When a sample is completed we increase the counter $C_{\mathbf{x},\mathbf{e}}$ by the weight of the generated sample. The final probability distribution is computed for every $\mathbf{x} \in val(\mathbf{X})$ as follows:

$$P(\mathbf{x} \mid \mathbf{e}) = \frac{C_{\mathbf{x},\mathbf{e}}}{C_{\mathbf{e}}} \tag{3.2}$$

Because the principle of weighted sampling might not be as obvious as in the cases of direct and rejection sampling I will try to elaborate and provide some insight. The key difference to previous sampling methods is that we want each sample to contribute to the final probability distribution estimate but then each generated sample cannot be accounted for by the same amount. Therefore the weight of a sample is used which literally bears the information how likely is the observed evidence $\mathbf{e}$ in that sample (for direct and rejection sampling the weight would be always one). To understand weighted sampling better I have devised a formal derivation of the formula (3.2). Let $\mathbf{Y}$ denote set of all variables in a BN

other than the variables $\mathbf{X} \cup \mathbf{E}$. First we will derive an alternative expression for $P(\mathbf{x}, \mathbf{e})$ which will be useful later.

$$
\begin{aligned}
P(\mathbf{x}, \mathbf{e}) &= \sum_{\mathbf{Y}} P(\mathbf{x}, \mathbf{e}, \mathbf{y}) \\
&= \sum_{\mathbf{Y}} \Big(\prod_{E_i \in \mathbf{E}} P(e_i \mid parents(E_i)) \prod_{X_i \in \mathbf{X}} P(x_i \mid parents(X_i)) \prod_{Y_i \in \mathbf{Y}} P(y_i \mid parents(Y_i)) \Big) \\
&\approx \sum_{\mathbf{Y}} \Big(w_{\mathbf{x},\mathbf{e},\mathbf{y}} \cdot \frac{N_{\mathbf{x},\mathbf{e},\mathbf{y}}}{N_{\mathbf{e},\mathbf{y}}} \cdot \prod_{Y_i \in \mathbf{Y}} P(y_i \mid parents(Y_i)) \Big) \\
&\approx \sum_{\mathbf{Y}} \Big(w_{\mathbf{x},\mathbf{e},\mathbf{y}} \cdot \frac{N_{\mathbf{x},\mathbf{e},\mathbf{y}}}{N_{\mathbf{e}} \cdot \prod_{Y_i \in \mathbf{Y}} P(y_i \mid parents(Y_i))} \cdot \prod_{Y_i \in \mathbf{Y}} P(y_i \mid parents(Y_i)) \Big) \\
&= \sum_{\mathbf{Y}} \Big(w_{\mathbf{x},\mathbf{e},\mathbf{y}} \cdot \frac{N_{\mathbf{x},\mathbf{e},\mathbf{y}}}{N_{\mathbf{e}}} \Big) = \frac{1}{N_{\mathbf{e}}} \cdot \sum_{\mathbf{Y}} \Big(w_{\mathbf{x},\mathbf{e},\mathbf{y}} \cdot N_{\mathbf{x},\mathbf{e},\mathbf{y}} \Big) \qquad (3.3)
\end{aligned}
$$

Let's go through the derivation step by step. The first line merely states we have to marginalize over all the remaining variables $\mathbf{Y}$ in the network. On the second line we just rewrote the joint probability according to the definition of a Bayesian network. On the third line we replaced exact probabilities by their sampled equivalents (therefore the approximation symbol instead of equality). Namely, weight of the sample $w_{\mathbf{x},\mathbf{e},\mathbf{y}}$ is equal to the probability $P(\mathbf{e} \mid \mathbf{x}, \mathbf{y})$ and ratio $N_{\mathbf{x},\mathbf{e},\mathbf{y}}/N_{\mathbf{e},\mathbf{y}}$ approximately equals to $P(\mathbf{x} \mid \mathbf{e}, \mathbf{y})$. On the forth line we needed to express the number $N_{\mathbf{e},\mathbf{y}}$ of samples with concrete values $\mathbf{e}, \mathbf{y}$ using the probability $P(\mathbf{y} \mid \mathbf{x}, \mathbf{e})$ and the total number of samples generated $N_{\mathbf{e}}$ (remember that in weighted sampling all samples are consistent with the evidence, so $N_{\mathbf{e}}$ really is the total number of samples). Finally, on the fifth line the big products over $\mathbf{Y}$ cancel out and we obtain the final expression for $P(\mathbf{x}, \mathbf{e})$. Now let's make use of it:

$$
\begin{aligned}
P(\mathbf{x} \mid \mathbf{e}) &= \frac{P(\mathbf{x}, \mathbf{e})}{P(\mathbf{e})} = \frac{P(\mathbf{x}, \mathbf{e})}{\sum_{\mathbf{X}} P(\mathbf{x}, \mathbf{e})} = \frac{\frac{1}{N_{\mathbf{e}}} \cdot \sum_{\mathbf{Y}} \Big(w_{\mathbf{x},\mathbf{e},\mathbf{y}} \cdot N_{\mathbf{x},\mathbf{e},\mathbf{y}} \Big)}{\sum_{\mathbf{X}} \frac{1}{N_{\mathbf{e}}} \cdot \sum_{\mathbf{Y}} \Big(w_{\mathbf{x},\mathbf{e},\mathbf{y}} \cdot N_{\mathbf{x},\mathbf{e},\mathbf{y}} \Big)} \\
&= \frac{\sum_{\mathbf{Y}} \Big(w_{\mathbf{x},\mathbf{e},\mathbf{y}} \cdot N_{\mathbf{x},\mathbf{e},\mathbf{y}} \Big)}{\sum_{\mathbf{X}} \sum_{\mathbf{Y}} \Big(w_{\mathbf{x},\mathbf{e},\mathbf{y}} \cdot N_{\mathbf{x},\mathbf{e},\mathbf{y}} \Big)} \\
&= \frac{C_{\mathbf{x},\mathbf{e}}}{\sum_{\mathbf{X}} C_{\mathbf{x},\mathbf{e}}} = \frac{C_{\mathbf{x},\mathbf{e}}}{C_{\mathbf{e}}}
\end{aligned}
$$

On the first line we rewrote the conditional probability using Bayes' rule and then we substituted for $P(\mathbf{x}, \mathbf{e})$ the expression (3.3) derived earlier. On the second line the $1/N_{\mathbf{e}}$ cancel out. To understand the third line we need to revisit how the weighted sampling works, especially what values are accumulated in a counter $C_{\mathbf{x},\mathbf{e}}$ and how. There are two key points. First, the weight $w_{\mathbf{x},\mathbf{e},\mathbf{y}}$ of each sample $\mathbf{x}, \mathbf{e}, \mathbf{y}$ is added to the counter $C_{\mathbf{x},\mathbf{e}}$ exactly $N_{\mathbf{x},\mathbf{e},\mathbf{y}}$-times. The second critical observation is that the counters don't care for values of the $\mathbf{Y}$ variables, so weights of two samples $\mathbf{x}, \mathbf{e}, \mathbf{y}'$ and $\mathbf{x}, \mathbf{e}, \mathbf{y}''$ are by the weighted sampling algorithm added to the same counter $C_{\mathbf{x},\mathbf{e}}$; this is effectively the same as marginalizing over the variables $\mathbf{Y}$ (ie. summing $\mathbf{Y}$ out) during the sampling. Combining these two pieces of information we conclude that the second and the third line have to equal. Formal derivation of the formula (3.2) for weighted sampling is now complete.

I believe that the sampling process can furthermore be accelerated by omitting a set of variables, denoted $\mathbf{Z}$, from the network being sampled where $\mathbf{Z}$ is maximal set of variables satisfying the following two conditions: (1) $\mathbf{Z} \cap (\mathbf{X} \cup \mathbf{E}) = \emptyset$, (2) no direct or indirect child of any variable from $\mathbf{Z}$ is in $\mathbf{X} \cup \mathbf{E}$. Rationale behind this is the same as in the variable elimination algorithm – when computing the final probability distribution, we effectively sum out variables $\mathbf{Z}$ and it is irrelevant what random values they are assigned because if two samples differ only in the values of $\mathbf{Z}$ then these samples contribute to the same counter $C_{\mathbf{x},\mathbf{e}}$ by the same weight because $\mathbf{Z} \cap \mathbf{E} = \emptyset$. This observation is applicable to direct and rejection sampling as well.

Important drawback of likelihood weighting is that it doesn't account well for evidence in leaf nodes or near to them. This is because in that case we effectively sample the prior probability distribution and finally we modify weight of the generated sample by evidence but the sampling process is most of the time unaffected by the evidence and virtually only weights of samples take evidence into account. So, for a very rare instantiation of evidence variables (eg. for some rare and/or big set of medical symptoms) all of the generated samples might have very small weight and, in case of medical diagnosis, no sample representing the "real cause" of observed evidence might actually get generated at all. Once again, this is because we sample prior probability distribution rather than posterior and these two distributions may be very different [12, p. 503].

3.2.5 Markov Chain Monte Carlo

The likelihood weighting method presented earlier doesn't account well for evidence near leaf nodes, ie. for so called *evidential reasoning*. Markov Chain Monte Carlo (MCMC) is also a stochastic method that generates samples but rather than generating each sample completely from scratch it modifies the last sample by resampling one of the non-evidence variables. This way, the information about all evidence variables propagates through the network and, with time, we get closer to the true posterior probability distribution $P(\mathbf{X} \mid \mathbf{e})$.

To explain how the MCMC algorithm works we first need to introduce so called *Markov blanket* of a variable X (also see Figure 3.2), denoted $MB(X)$. The Markov blanket of X is the minimal set of variables not including X such that X is conditionally independent of all other variables in the BN given $MB(X)$. It is fairly obvious that the Markov blanket includes variables $Parents(X)$ and direct children of X, denoted $Children(X)$, because there is a direct connection between these variables and X. Furthermore, $MB(X)$ also needs to include parents of every variable in $Children(X)$ because observing any of the children $C_i \in Children(X)$ activates a V-structure and X becomes conditionally dependent on $Parents(C_i)$ given C_i. Formally the Markov blanket could be defined as follows:

$$MB(X) = Parents(X) \cup Children(X) \cup \bigcup_{C_i \in Children(X)} Parents(C_i) \setminus \{X\}$$

Now, in order to resample some non-evidence variable X and thereby to produce a new MCMC sample, we need to compute the probability distribution $P(X \mid mb(X))$ and sample new value of X from this distribution. In a sense, Markov blanket of variable X defines context of the resampling, ie. all the variables whose values we need to know.

The MCMC algorithm [15, p. 516] starts by producing one sample consistent with the evidence, eg. by one iteration of likelihood weighting with the weight discarded. Then we repeat the following: For each non-evidence variable V we resample this variable from the current distribution $P(V \mid mb(V))$ producing a new sample with value of variable V

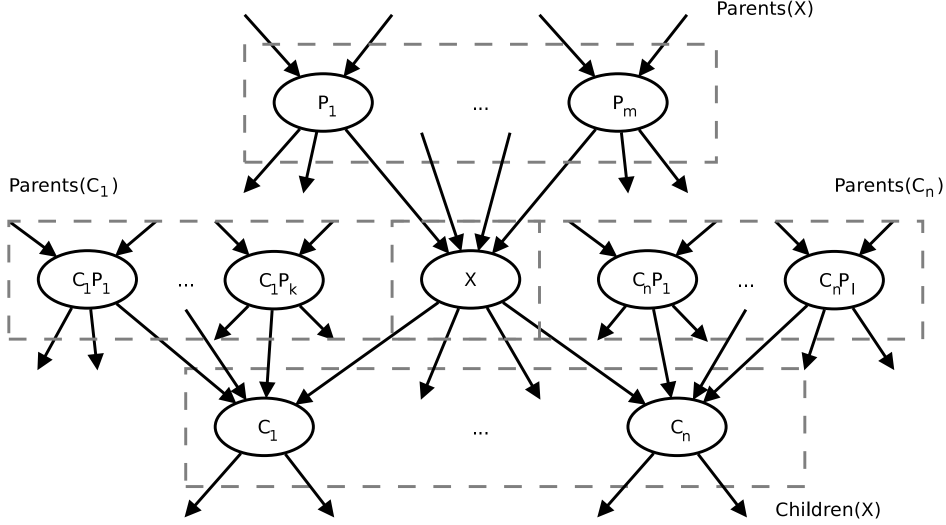


Figure 3.2: Markov blanket $MB(X)$ of variable X is the set of all variables in this figure except for X.

potentially changed. For the new sample we increase the corresponding counter $N_{\mathbf{x},\mathbf{e}}$ and proceed with resampling of another variable. The distribution $P(V \mid mb(V))$ needed for resampling of variable V can be obtained by computing $P(v \mid mb(V))$ for every v as follows:

$$P(v \mid mb(V)) = \alpha \cdot P(v \mid parents(V)) \prod_{C_i \in Children(V)} P(c_i \mid parents(C_i))$$

where α is a normalizing constant. Note that $mb(\cdot)$ and $parents(\cdot)$ denote concrete instantiation of variables $MB(\cdot)$ and $Parents(\cdot)$ in the current sample. As we can see, when computing the probability distribution $P(V \mid mb(V))$ we only need to account for probabilities of variables V and $Children(X)$, not of the whole Markov blanket. This is because when we set V to concrete value v, only probabilities in nodes $\{V\} \cup Children(X)$ are affected due to the definition of a Bayesian network where probability at a node depends purely on its parents.

Intuition behind the MCMC algorithm is that the sampling process will reach a dynamic equilibrium in which time spent with each instantiation of non-evidence variables (ie. the counter of samples $N_{\mathbf{x},\mathbf{e}}$) is proportional to the probability of this instantiation. More formally, MCMC is based on state space induced by a BN represented by a Markov model [15, p. 516] but this theory is not necessary for understanding the MCMC inference method.

3.3 Model learning

So far, we have been reasoning about Bayesian networks that were already given to us, so that both the structure of the BN and the CPDs associated with nodes were known. Now we are going to examine the problem of creating a Bayesian network so that the probability distribution it induces somehow corresponds to the real probability distribution of the target domain.

Basically, there are two approaches to creating a BN. First option is to cooperate with an expert of the target domain who can make correct dependency and independency as-

sumptions and provide us with conditional probability distributions, ie. with CPDs for all the nodes. The other approach is to use some fully automated techniques for creating models based on a sufficiently big dataset. The dataset can be viewed as a set of samples of the target probability distribution $P*$ which we attempt to reconstruct. The aim is to create a model $\tilde{\mathcal{M}}$ such that the probability distribution $P_{\tilde{\mathcal{M}}}$ induced by $\tilde{\mathcal{M}}$ is very close to the original distribution $P*$.

Complete construction of a BN through cooperation with an expert is problematic because, for a non-trivial BN, the task often requires significant time (several months [12]), the expert might not correctly capture CPDs, especially for nodes with large number of parents, and furthermore there might not even be an expert of the target domain at all. On the other hand, automated techniques of model construction are constrained by limited computational power and, more importantly, by the size of the supplied dataset. As will be shown later, we encounter classical problem of overfitting and bias-variance trade-off present in the whole field of artificial intelligence and machine learning. In practice, these two approaches are often combined in the way that an expert defines structure (all variables and their dependencies) and an automated process determines CPDs from a supplied dataset.

3.4 Parameter estimation

Goal of parameter estimation methods is to supply CPDs to a BN whose structure is already known. For the estimation to be reasonably correct, we need to have sufficiently large dataset with respect to complexity of the BN structure.

3.4.1 Maximum likelihood estimation

Maximum likelihood estimation views the dataset $\mathcal{D}$ as a set of independent samples $x_1, \ldots, x_m$ obtained from a parametrized probability distribution with parameter Θ. The parameter Θ can be seen as a vector of entries of all CPDs in the given BN (roughly speaking, Θ is "serialized" equivalent of tabular CPDs). Core of the maximum likelihood estimation is to choose Θ in such a way that the probability of obtaining samples $x_1, \ldots, x_m$ from the parametrized distribution is maximal. Formally $\Theta = \arg\max P(\mathcal{D} \mid \Theta)$ where $P(\mathcal{D} \mid \Theta)$ is called the *likelihood function*.

Let's inspect a simple example of m coin tosses $x_1, \ldots, x_m$ with a biased coin (inspired by [12]) for which $P(X = heads) = \theta$. Let's suppose we get H times heads and T times tails. Then we can write $P(x_1, \ldots, x_m \mid \theta) = \theta^H (1-\theta)^T$ because the coin tosses are mutually independent given θ. The task of maximizing the expression $\theta^H (1-\theta)^T$ is equivalent to maximizing its logarithm $H \cdot \ln(\theta) + T \cdot \ln(1-\theta)$. The equation $\frac{\partial}{\partial \theta}\big(H \cdot \ln(\theta) + T \cdot \ln(1-\theta)\big) = 0$ yields global maximum at $\theta = H/(H+T)$ which is a fairly intuitive conclusion – probability of heads is, according to the experiments, computed as the number of times we got heads divided by the number of all tosses. So, maximum likelihood estimation is the approach we would normally apply, probably without even knowing its proper justification. Similar formulas as for the binomial variable can be obtained for a general set $\mathbf{X}$ of multinomial variables, formally $P(\mathbf{X} = \mathbf{x}) = N_{\mathbf{X}}/N_{all}$.

Now suppose we want to compute probability distribution $P(X \mid Parents(X))$ for some node X according to our dataset, ie. to compute the CPD for that node. Maximum likelihood estimation is, as demonstrated earlier, an intuitive approach when we partition the dataset into disjoint subsets, each for a concrete instantiation of variables $\{X\} \cup Parents(X)$. Operating just with cardinalities of these subsets, a CPD entry

$P(X = x_i \mid Parents(X) = \mathbf{p_j})$ is computed as the ratio $N_{x_i,\mathbf{p_j}}/N_{\mathbf{p_j}}$. Potential problem with this approach is that the number of subsets grows exponentially with the number of parents and hence the estimated CPD looses precision. This exponential explosion of possible instantiations is called *fragmentation* and is one of the main problems of learning BNs from data. Because of fragmentation, a perfect model (in terms of structure) capturing all real dependencies among variables might be outperformed by a simpler (and thereby wrong) model just because the amount of data is insufficient to compute CPDs for the more complicated structure accurately[3] [12]. This is a typical AI problem of overfitting. In extreme cases, $N_{x_i,\mathbf{p_j}}$ could even be zero which is usually very wrong because, according to computed CPDs, there could never be an event in which $X_i = x_i$ and $Parents(X_i) = \mathbf{p_j}$. This can be prevented by applying the Laplace's correction commonly used in context of Naïve Bayes classifier.

3.4.2 Bayesian estimation

An important drawback of the maximum likelihood estimation presented earlier is that it doesn't really account for the size of the dataset – small dataset may not have enough samples for every instantiation of every variable and its parents and the dataset might also be noisy. The key idea of Bayesian estimation is to view the parameters Θ themselves (ie. CPDs) as random variables with some prior distribution $P(\Theta)$ and then, according to the supplied dataset, compute the posterior distribution $P(\Theta \mid \mathcal{D})$ which will be our estimated CPDs. So, parameter learning could in this case be viewed as a type of inference. More formally, we define a prior distribution $P(\Theta)$ over parameters Θ with some degree of strength of this initial distribution (will be explained later) and then, using the Bayes' rule, compute the posterior probability distribution $P(\Theta \mid \mathcal{D}) \propto P(\mathcal{D}, \Theta)P(\Theta)$ as we get more empirical data; the posterior distribution corresponds to the network parameters determined by the Bayesian estimation method.

From its nature, Bayesian estimation can capture prior believes regarding the CPDs. For example, in case of a die the general expectation is that the probability of rolling a chosen number is 1/6. Now, when inferring parameters Θ for a die, Bayesian estimation can distinguish between rolling a six three times out of twelve throws, which may be accounted merely to statistical noise, and rolling a six 3 000 times out of 12 000 throws. It is clear that these two datasets tell us something different about the die, although probability of rolling a six, according to the maximum likelihood estimation discussed earlier, is 1/4 in both cases – ratio $N_6/N_{all} = 3/12 = 3\,000/12\,000$. With Bayesian estimation the inferred probability of rolling a six will still be close to the generally expected 1/6 when working with just twelve samples. On the other hand, in the situation with 12 000 samples the inferred probability of rolling a six will be much closer to 1/4. Another example of shortcomings of the maximum likelihood estimation could be an insurance company attempting to evaluate driving skills of a brand new young driver. If this driver were to crash on his very first ride, certainly he shouldn't get a good rating but it would be unreasonable to conclude that probability of him crashing the next day is $1/1 = 100\%$ as suggested by the maximum likelihood estimation.

In context of Bayesian estimation, besides the real dataset $\mathcal{D}$ containing real samples, we also work with imaginary *pseudo-samples*. These pseudo-samples define the prior distribu-

[3]Such observation has often been made, for example, with the Naïve Bayes model which assumes that any two effect variables are independent given the cause variables. Such assumption is seldom justified, nevertheless Naïve Bayes models have proved themselves to perform well.

tion $P(\Theta)$ by plain maximum likelihood estimation as a ration of pseudo-sample counts – for example, in the prior distribution $P(\Theta)$ holds that $P(X = x) = \alpha_x/\alpha$ where the α symbol, optionally with a subscript, is a counter of pseudo-samples similar to N counters of real samples introduced earlier. The exact mathematical derivation [12, p. 733] of formulas for Bayesian estimation is based on *Dirichlet distribution* with hyperpameters $(\alpha_1, \ldots, \alpha_k)$ that correspond to the pseudo-sample counters. The derivation is not trivial and also is not needed further in this thesis, so we jump straight to the practical conclusions and interpretations. Bottom line is that for a multinomial variable the Dirichlet distribution has the nice property that the prior distribution $P(\Theta)$ and the posterior $P(\Theta \mid \mathcal{D})$ have exactly the same form as a ratio of some counters, so when the prior distribution is computed by maximum likelihood estimation over a set of pseudo-samples using α counters, then posterior distribution is computed also using maximum likelihood estimation but over dataset containing both real samples and pseudo-samples as well, ie. involving both α and N counters. For a given dataset $\mathcal{D}$ and some vector of hyperparameters $\vec{\alpha}$ the probability of variable X having value x_i is given as follows (for a node without and with parents respectively):

$$P(X = x_i \mid \mathcal{D}) = \frac{\alpha_{x_i} + N_{x_i}}{\sum_j (\alpha_{x_j} + N_{x_j})}$$

$$P(X = x_i \mid Parents(X) = \mathbf{p_i}, \mathcal{D}) = \frac{\alpha_{x_i,\mathbf{p_i}} + N_{x_i,\mathbf{p_i}}}{\sum_j (\alpha_{x_j,\mathbf{p_i}} + N_{x_j,\mathbf{p_i}})}$$

Let's take a closer look at the hyperparameters. It is easy to see from the formulas above that the relative difference between α and N terms determines strength of the prior distribution $P(\Theta)$, ie. how many real samples accounted for in the N counters does it take to significantly deviate from the prior distribution governed purely by hyperparameters α. We have already encountered two special cases of hyperparameters – $(0, \ldots, 0)$ is the pure maximum likelihood estimation when we compute the CPDs based just on the dataset $\mathcal{D}$; hyperparameters $(1, \ldots, 1)$ correspond the maximum likelihood estimation with Laplace's correction. It is clear that the maximum likelihood estimation and the Bayesian estimation are asymptotically the same for large datasets. However, Bayesian estimation generalizes better with sparse datasets and exhibits lower sensitivity to noise in the data [12, p. 749]. In addition, we can quite simply enforce that each possible instantiation of variables in a network occurs with non-zero probability.

At this point the only thing we haven't covered yet is how can the hyperparameters be defined in a readable and easy-to-understand fashion. There will be two approaches, both working with so called *equivalent sample size* α which can be viewed as the total number of pseudo-samples. First, and more general, option is to use another BN whose CPD entries define the prior distribution $P(\Theta)$ and equivalent sample size determines strength of the prior. Structure of this BN is identical to the BN, whose CPDs we want to estimate, therefore it is easy to specify the prior distribution simply by filling in prior CPD entries. Hyperparameters are then determined using inference (eg. by sampling) of that BN as $\alpha_{x_i,\mathbf{p_j}} = \alpha \cdot P(x_i, \mathbf{p_j})$. Second approach is to use *uniform prior* in which each event is equally likely, ie. $P\big(X = x_i, Parents(X) = \mathbf{p_j}\big) = \alpha / card\big(\{X\} \cup Parents(X)\big)$ where $card(\cdot)$ denotes the number of all possible instantiations of the specified set of variables.

Parameter learning methods explained so far deal strictly with table CPDs and assume that parameters Θ of the BN are independent given complete data. Advanced methods of parameters learning include learning structured probability distributions (eg. tree-structured CPDs) and dependent or shared parameters [12].

3.5 Structure learning

So far we have seen how to compute CPDs for a given graph structure of a BN based on some dataset. Learning the structure itself can be useful for a number of reasons. For example, we might want to create an accurate model of certain domain to be able to perform inference later (eg. medical diagnosis or prediction of traffic conditions [10]). Other application could be in the field of knowledge discovery when we are interested in finding out the causal dependencies between random variables so finding the structure itself is our goal (eg. discovering protein-signaling networks purely based on statistical data [5]).

Learning structure of a BN is a complicated task. One of the problems is to pick a good trade-off between high bias (restricting complexity of the graph structure) and high variance (allowing more dependencies to get a better fit for training data with the cost of higher fragmentation). As we have already discussed, with a small dataset we may actually benefit by having a simpler structure. On the other hand, with a large dataset we don't want to restrict our hypothesis space too much because the dataset is sufficient to learn higher number of parameters with reasonable accuracy.

In this thesis we are going to focus on *score-based* methods of structure learning, concretely likelihood score and Bayesian score. In the case of score-based methods, structure learning is basically a discrete optimization task and for such tasks there are many well known approaches – greedy search, simulated annealing, genetic algorithms etc. Nevertheless, in the field of learning Bayesian networks the majority of structure learning methods are local state-space search algorithms [12, p. 814] (eg. hill climbing, greedy search, tabu search). A common requirement for using any optimization technique is a mechanism that evaluates quality of a candidate solution. Such role is carried out by the scoring functions which we will discuss in detail.

Other than the score-based approach there are also constraint-based approaches [12, p. 786] which make a series of one-time decisions regarding dependency and independency of variables (eg. by performing independency tests using χ^2) and construct a model satisfying those assumptions. The fact that decisions are irreversible, as opposed to score-based methods, makes the constraint-based approach somewhat limited.

3.5.1 Optimization algorithm

As we have already discussed, in context of Bayesian networks, the techniques most frequently used are local state-space search methods. We will use the tabu-search algorithm [12, p. 816] which is an extension of simple hill climbing and works as follows. We have an initial graph structure with fixed set of variables. The initial structure may be empty, it may be the best-scoring tree/forest or some network capturing our prior knowledge and believes regarding the target domain. At each iteration we evaluate the change of score for all possibilities of adding a new edge, reversing direction or deleting an existing edge in the current structure and we accept the structural modification with the best score gain. Those actions are repeated until the solution improves. Furthermore, we maintain a list of last n structural modifications (a *tabu-list*) and disallow those modifications to be reversed. If we hit a local optima, ie. no feasible structural modification leads to a better score, we make a predefined number of random steps (called *random restart*) and continue the local search.

3.5.2 Likelihood score

The likelihood score has information-theoretic foundations and basically quantifies how well a given network structure matches our dataset. We assume that for a given structure $\mathcal{G}$ the parameters Θ are learnt using the maximum likelihood estimation, so the likelihood score really evaluates the hypothesis as a complete BN $(\mathcal{G}, \Theta)$, only the parameters are implicit.

Let's suppose we have a dataset $\mathcal{D} = \{\xi^{(1)}, \ldots, \xi^{(m)}\}$. Again, learning corresponds with maximizing probability of sampling exactly the data $\mathcal{D}$ from the learnt network, only now we search for optimal BN structure rather than parameters. Concretely, we will focus on maximizing the likelihood function $P(\mathcal{D} \mid \Theta, \mathcal{G})$ which is basically probability of obtaining exactly the samples contained in dataset $\mathcal{D}$ from the network $(\mathcal{G}, \Theta)$, ie. one can write $P(\mathcal{D} \mid \Theta, \mathcal{G}) = P(\xi^{(1)} \mid \Theta, \mathcal{G}) \cdots P(\xi^{(m)} \mid \Theta, \mathcal{G})$. Furthermore, we will use *log-likelihood function* $\log P(\mathcal{D} \mid \Theta, \mathcal{G})$. This is a useful trick because logarithm transforms a product into a sum which allows us to compute score of a network based purely on computations over individual nodes as we will see later (the likelihood score decomposes over a network).

To understand how the likelihood score changes for two networks that differ only by presence of a single edge, we will examine network $\mathcal{G}_1$ with two independent variables X and Y and network $\mathcal{G}_2$ with structure $X \to Y$. From the previous text we know likelihood scores of these networks (logarithms of likelihood functions) can be computed as follows:

$$\begin{aligned} score_L(\mathcal{G}_1 : \mathcal{D}) &= \sum_{i=1}^{|\mathcal{D}|} \Big(\log \hat{P}(x^{(i)}) + \log \hat{P}(y^{(i)}) \Big) \\ score_L(\mathcal{G}_2 : \mathcal{D}) &= \sum_{i=1}^{|\mathcal{D}|} \Big(\log \hat{P}(x^{(i)}) + \log \hat{P}(y^{(i)} \mid x^{(i)}) \Big) \end{aligned}$$

where $score_L(\mathcal{G}_1 : \mathcal{D})$ denotes the likelihood score of dataset $\mathcal{D}$ in network with structure $\mathcal{G}_1$, $x^{(i)}$ is value of variable X in sample $\xi^{(i)}$ and $\hat{P}$ denotes so called *empirical distribution*. Empirical distribution is computed from the dataset $\mathcal{D}$ according to the maximum likelihood principle (eg. $\hat{P}(x_i) = N_{x_i}/N$ or $\hat{P}(x_i \mid y_j) = N_{x_i,y_j}/N_{y_j}$) and therefore $\hat{P}(\cdot)$ also corresponds to the CPD entries (we have said that parameters Θ of a network with given structure are assumed to be computed using maximum likelihood estimation).

Now let's examine the difference between having an edge between variables X and Y or not, how does the likelihood score change? We obtain the following result (derivation inspired by [12, p. 791]):

$$\begin{aligned} score_L(\mathcal{G}_2 : \mathcal{D}) - score_L(\mathcal{G}_1 : \mathcal{D}) &= \sum_{i=1}^{|\mathcal{D}|} \log \hat{P}(y^{(i)} \mid x^{(i)}) - \sum_{i=1}^{|\mathcal{D}|} \log \hat{P}(y^{(i)}) \\ &= \sum_{X,Y} N_{x,y} \log \hat{P}(y \mid x) - \sum_{Y} N_y \log \hat{P}(y) \\ &= \sum_{X,Y} N \cdot \hat{P}(x,y) \log \hat{P}(y \mid x) - \sum_{Y} N \cdot \hat{P}(y) \log \hat{P}(y) \\ &= \sum_{X,Y} N \cdot \hat{P}(x,y) \log \hat{P}(y \mid x) - \sum_{X,Y} N \cdot \hat{P}(x,y) \log \hat{P}(y) \\ &= N \sum_{X,Y} \hat{P}(x,y) \log \frac{\hat{P}(y \mid x)}{\hat{P}(y)} = N \sum_{X,Y} \hat{P}(x,y) \log \frac{\hat{P}(x,y)}{\hat{P}(x)\hat{P}(y)} \\ &= N \cdot I_{\hat{P}}(X;Y) \end{aligned}$$

First, I will explain the derivation step by step and the next paragraph will address the question of usefulness of the derived result. The first step (performed on the second line) is to change the logic of the summation from "over the dataset $\xi^{(1)}, \ldots, \xi^{(m)}$" to "over possible instantiations of X, Y"; such change is legal because each $\hat{P}(\cdot)$ value will be taken into account just as many times as before. On the third line the counts $N_{x,y}$ and N_y are expresses using size of the dataset N and empirical probabilities of respective instantiations x, y and y. After that, on the fourth line, we can extend the second summation from just over Y to over X, Y. The last step, the fifth line, is merging the two sums and logarithms and applying Bayes' rule on the $\hat{P}(y \mid x)$ term. Finally, we introduce special notation for the final expression.

The term $I_{\hat{P}}(X; Y)$ is called *mutual information* between variables X and Y in the distribution $\hat{P}$ and expresses the average distance between the joint distribution $\hat{P}(X, Y)$, in which the variables may be dependent, to the distribution given as product of marginal distributions $\hat{P}(X)$ and $\hat{P}(Y)$, when the variables are independent. The bigger the distance the more variables X and Y are correlated which in context of a BN means they should be connected by an edge. So, mutual information is a natural way of measuring strength of dependency among variables. In a similar fashion mutual information is defined for sets of variables $\mathbf{X}, \mathbf{Y}$ rather than for single variables X, Y, we just sum over all possible instantiations $\mathbf{x}, \mathbf{y}$ of $\mathbf{X}, \mathbf{Y}$. It can be shown (for details see [12, p. 792]) that the overall likelihood score decomposes over a BN with general structure $\mathcal{G}$ as follows:

$$score_L(\mathcal{G} : \mathcal{D}) = N \sum_{X_i \in Nodes} I_{\hat{P}}(X_i; Parents(X_i)) - N \sum_{X_i \in Nodes} H_{\hat{P}}(X_i) \tag{3.4}$$

where $H_{\hat{P}}(\cdot)$ is entropy of an individual variable in the distribution $\hat{P}$. As we can see, $H_{\hat{P}}(\cdot)$ is a constant relative to the structure $\mathcal{G}$. So, in order to compare two network structures by their likelihood scores we only need to consider value of the sum over the mutual information terms and computation of the $H_{\hat{P}}(\cdot)$ may be ommited.

The difference of likelihood scores between a superstructure with more edges and its substructure is always non-negative and, in fact, is equal to zero if and only if the two variables X, Y in the dataset $\mathcal{D}$ appear to be *perfectly independent* which is due to statistical noise almost never true. Therefore the likelihood score itself almost always suggests the greedy heuristics to add an edge which would eventually lead to a very densely connected network. In other words, the likelihood score is very prone to overfitting. This problem can be addressed by thresholding of the score increase or by restraining complexity of the network (defining maximal number of parents or maximal number of overall network parameters). Another approach is to account for network complexity in the scoring function itself, thus imposing a penalty on complicated structures. The latter is exactly what the BIC score does.

3.5.3 BIC score

The likelihood score discussed earlier never favors a simpler structure over a more complex one. We will discuss a variant of Bayesian score called *BIC score* which is, in the end, just the likelihood score extended by a penalty term although theoretical foundations and mathematical derivations leading to the final formulas are entirely different.

The BIC score takes into account both network complexity as well as the size of the dataset and combines these two pieces of information in such a way that with a small

dataset the BIC score tends to keep the structure simple and allows only for the strongest dependencies to be be reflected in the network. As the dataset gets larger, the BIC score allows for a more complicated structure that encodes also weaker dependencies indicated by the data.

The BIC score is defined as follows (for complete derivation see [12, p. 794]):

$$score_{BIC}(\mathcal{G} : \mathcal{D}) = N \sum_{X_i \in Nodes} I_{\hat{P}}(X_i; Parents(X_i)) - N \sum_{X_i \in Nodes} H_{\hat{P}}(X_i) - \frac{\log N}{2} Dim[\mathcal{G}] \quad (3.5)$$

where $Dim[\mathcal{G}]$ is dimension of the network (number of its parameters). Notice that the two sums are the likelihood score $score_L(\mathcal{G} : \mathcal{D})$. We can see that the BIC score increases linearly in N with variables, that appear to be dependent, being connected (the first term) and decreases logarithmically in N with the network complexity (the third term). Thus for a sparse dataset only the strongest dependencies will be reflected in the final network, whereas for a large dataset the penalization grows at slower rate allowing the network to have a more complicated structure including also weaker dependencies.

Note that the BIC score decomposes over the graph $\mathcal{G}$, so when we want to evaluate some structural change, we only need to consider the score difference of nodes affected by this change, so called *delta score*. This observation is crucial for an effective implementation of structure learning based on local optimization of the score function.

3.5.4 Bayesian score

Bayesian score for a graph structure is yet another application of the Bayesian principle which we have already encountered in context of parameter estimation – whatever we are uncertain about should be modeled as a random variable. In case of Bayesian score we are uncertain both about parameters Θ (as in Bayesian parameter estimation) and even about the network structure $\mathcal{G}$. To state the idea formally, graph structure $\mathcal{G}$ is a random variable for which holds $P(\mathcal{G} \mid \mathcal{D}) = P(\mathcal{D} \mid \mathcal{G})P(\mathcal{G})/P(\mathcal{D})$. The denominator is independent of the network structure and parameters, so we can safely consider just the numerator. Further we will consider logarithm of the numerator which is perfectly legal for a scoring function because logarithm is a monotone function and probability is always non-negative. By applying logarithm to the numerator we obtain the Bayesian score for structure $\mathcal{G}$ and data $\mathcal{D}$ as:

$$score_B(\mathcal{G} : \mathcal{D}) = \log P(\mathcal{D} \mid \mathcal{G}) + \log P(\mathcal{G}) \quad (3.6)$$

The structure prior $P(\mathcal{G})$ in the equation (3.6) allows us to penalize certain structures. However, effect of the structure prior is rather minor, especially in asymptotic analysis [12, p. 804], because this term doesn't change with the number of samples. Literature suggests using a uniform prior or a prior that exponentially penalizes complexity of the structure which is useful for sparse datasets [12].

The term $P(\mathcal{D} \mid \mathcal{G})$ in equation (3.6) will be our major concern. It is a marginal distribution (because all possible parameter settings Θ are marginalized out) mathematically defined in the following integral form:

$$P(\mathcal{D} \mid \mathcal{G}) = \int_{\Theta} P(\Theta \mid \mathcal{G}) P(\mathcal{D} \mid \mathcal{G}, \Theta) \; d\Theta \quad (3.7)$$

The integral expression can be interpreted as computing the average probability of $\mathcal{D}$ given $\mathcal{G}, \Theta$ weighted by probabilities of all possible parameter settings $P(\Theta \mid \mathcal{G})$. So, the Bayesian

estimation is less optimistic when determining probability of a dataset given structure. On the other hand, the BIC score considers only single parameter setting Θ computed by the maximum likelihood estimation which is "tailored" for the specific dataset to maximize probability of the data given structure. Therefore Bayesian score is less prone to overfitting [12, p. 795]. The integral expression (3.7) is hard to deal with practically. Fortunately, for a BN with multinomial variables, whose prior parameter distribution is a Dirichlet distribution with hyperparameters $(\alpha_1, \dots, \alpha_n)$, the expression $P(\mathcal{D} \mid \mathcal{G})$ can be written using the chain rule as follows:

$$P(\mathcal{D} \mid \mathcal{G}) = \prod_{i=1}^{|\mathcal{D}|} P\left(\xi^{(i)} \mid \mathcal{G}, \xi^{(1)}, \dots, \xi^{(i-1)}\right) \tag{3.8}$$

where $\mathcal{D} = \{\xi^{(1)}, \dots, \xi^{(m)}\}$ is a dataset. If you think about the equation (3.8), we could view each of the product terms as making a prediction of how likely is the data instance $\xi^{(i)}$ given the previous instances and the structure $\mathcal{G}$, ie. as if we were predicting probability of an unseen instance $\xi^{(i)}$ based on a model learnt using the previous instances. In other words, we measure predictive power the network structure for the whole dataset, only without making any explicit testing or validation.

Before we get to the general expression for $P(\mathcal{D} \mid \mathcal{G})$ in context of Bayesian networks let's consider a simple coin flipping example, ie. a binomial distribution over values $\{H, T\}$. Say our dataset is $\mathcal{D} = \{H, H, T, H, T\}$ and the prior distribution is a Dirichlet distribution with hyperparameters α_H, α_T and $\alpha = \alpha_H + \alpha_T$. Also, in this simple example there is really no structure $\mathcal{G}$ to talk about, therefore we can write $P(\mathcal{D} \mid \mathcal{G}) = P(\mathcal{D})$ in the following form (explanation follows immediately):

$$\begin{aligned}
P(\mathcal{D}) &= P(\xi^{(1)})P(\xi^{(2)} \mid \xi^{(1)})P(\xi^{(3)} \mid \xi^{(1)}, \xi^{(2)})P(\xi^{(4)} \mid \xi^{(1)}, \xi^{(2)}, \xi^{(3)})P(\xi^{(5)} \mid \xi^{(1)}, \dots, \xi^{(4)}) \\
&= \frac{\alpha_H}{\alpha} \cdot \frac{\alpha_H + 1}{\alpha + 1} \cdot \frac{\alpha_T}{\alpha + 2} \cdot \frac{\alpha_H + 2}{\alpha + 3} \cdot \frac{\alpha_T + 1}{\alpha + 4} = \frac{\alpha_H(\alpha_H + 1)(\alpha_H + 2) \cdot \alpha_T(\alpha_T + 1)}{\alpha \cdots (\alpha + 4)} \\
&\left(= \left| \text{if were } \alpha_H, \alpha_T \in \mathbb{N} \right| = \frac{(\alpha - 1)!}{(\alpha + 4)!} \cdot \frac{(\alpha_H + 2)!}{(\alpha_H - 1)!} \cdot \frac{(\alpha_T + 1)!}{(\alpha_T - 1)!} \right) \\
&= \frac{\Gamma(\alpha)}{\Gamma(\alpha + 5)} \cdot \frac{\Gamma(\alpha_H + 3)}{\Gamma(\alpha_H)} \cdot \frac{\Gamma(\alpha_T + 2)}{\Gamma(\alpha_T)}
\end{aligned}$$

As states the equation (3.8), probability of $\xi^{(i)}$ is computed based on samples $\xi^{(j)}$ with index $j < i$ and on the pseudosamples from the prior distribution. The computation is fairly straightforward – suppose $\xi^{(i)} = H$, then the corresponding probability is given as the number of samples and pseudosamples of heads we have seen before $\xi^{(i)}$ divided by the total number of samples and pseudosamples before $\xi^{(i)}$. For the first training example $\xi^{(1)}$, respectively for $P(\xi^{(1)})$, we have only pseudosamples, so $P(\xi^{(1)}) = \alpha_H/\alpha$. For $\xi^{(2)}$ we have already seen one occurrence of heads and also one real sample, so $P(\xi^{(2)}) = (\alpha_H + 1)/(\alpha + 1)$. In case of $\xi^{(3)}$ we have not seen any real tails yet but there already have been two real samples, hence $P(\xi^{(3)}) = \alpha_T/(\alpha + 2)$ a so on. Rest of the work is just to rewrite the expression using *gamma function* which is a continuous generalization of factorial for which holds $\Gamma(n) = (n - 1)!$ provided $n \in \mathbb{N}$. Notice on the second line of the formula above that we grouped terms concerning the same value of our random variable (values heads or tails) – the value of $P(\mathcal{D})$ doesn't depend on the ordering of samples in the dataset and the final line tells us that $P(\mathcal{D})$ can be computed purely based on occurrence counts rather

than by going through the dataset sample by sample. This observation will apply also in context of Bayesian networks.

Based on the concrete example for binomial distribution of coin flips, that has just been presented, one can generalize and obtain a formula for a multinomial distribution over some variable X with values $\{x_1, \dots, x_m\}$ with a Dirichlet prior $\alpha_1, \dots, \alpha_m$ [12, p. 798]:

$$P(\mathcal{D}) = \frac{\Gamma(\alpha)}{\Gamma(\alpha + N)} \prod_{i=1}^{m} \frac{\Gamma(\alpha_i + N_i)}{\Gamma(\alpha_i)} \tag{3.9}$$

where $\alpha = \alpha_1 + \cdots + \alpha_m$, $N = |\mathcal{D}|$ and N_i is the number of occurrences of x_i in $\mathcal{D}$. Further, for a Bayesian network, in which each sub-CPD[4] $P(\Theta_{X|\mathbf{pa}} \mid \mathcal{G})$ is a Dirichlet distribution with hyperparameters $\{\alpha_{x|\mathbf{pa}} \mid x \in val(X)\}$, the term $P(\mathcal{D} \mid \mathcal{G})$ can be written in a convenient factorized form as [12, p. 801]:

$$P(\mathcal{D} \mid \mathcal{G}) = \prod_{X_i \in Nodes} \left[\prod_{\mathbf{pa} \in val(Parents(X))} \left(\frac{\Gamma(\alpha_{X_i|\mathbf{pa}})}{\Gamma(\alpha_{X_i|\mathbf{pa}} + N_{\mathbf{pa}})} \prod_{x \in val(X_i)} \frac{\Gamma(\alpha_{x|\mathbf{pa}} + N_{x,\mathbf{pa}})}{\Gamma(\alpha_{x|\mathbf{pa}})} \right) \right] \tag{3.10}$$

where $val(\mathbf{X})$ is the set of all instantiations $\mathbf{x}$ of variables $\mathbf{X}$, $N_{x,\mathbf{pa}}$ is number of samples for which $X = x \wedge Parents(X) = \mathbf{pa}$, similarly for $N_{\mathbf{pa}}$ and $\alpha_{X|\mathbf{pa}} = \sum_{x \in val(X)} \alpha_{x|\mathbf{pa}}$, $N_{\mathbf{pa}} = \sum_{x \in val(X_i)} N_{x,\mathbf{pa}}$.

Although the formal derivation of (3.10) is omitted, I will attempt to provide an explanation of how the formula could be derived. Intuitively, we would begin with the chain rule (3.8) multiplying probability of each sample given the previous ones. A sample is an instantiation $x_1, \dots, x_k$ of all variables in the BN and, by the definition of a BN, we would obtain probability of some concrete sample by multiplying $P(x_i \mid parents(X_i))$ in all nodes of the network. At this point let me stress out that the CPD in some node X_i contains a number of probability distributions $P(X_i \mid parents(X_i))$, one for every possible assignment $parents(X_i)$ of variables $Parents(X_i)$. As I already argued by the coin flipping example, when computing $P(\mathcal{D} \mid \mathcal{G})$ we can either go though the dataset sample by sample (as in the chain rule) or we can use a summary based on total occurrence counts of all possible instantiations of all variables which leads to a formula with gamma functions; equation (3.10) uses the latter approach. Translated to a BN, for a concrete sample, each node X_i selects a concrete probability distribution $P(X_i \mid parents(X_i))$ based on concrete assignment to $Parents(X_i)$ in this particular sample. So, for the concrete sample we get a product of multinomial distributions $P(X_i \mid parents(X_i))$, one at each node, and we already know how to compute $P(\mathcal{D})$ for a single multinomial distribution, ie. for a single node X_i, namely by equation (3.9). Rest of the process is to iterate over all nodes $X_i \in Nodes$ and over possible assignments $\mathbf{pa} \in val(Parents(X_i))$ and combine them into one big product, obtaining (3.10).

In the equation (3.10) for Bayesian score of a BN we can notice the way it factorizes over the network structure – over respective nodes X_i and, more importantly, over assignments $\mathbf{pa} \in val(Parents(X))$. What does that mean? For one, this very feature is where network structure is reflected in the Bayesian score. For two, if you think about $P(\mathcal{D} \mid \mathcal{G})$ as being $\prod_{X_i} \prod_{\mathbf{pa}} P(X_i \mid \mathbf{pa})$ then it can be shown that if X_i and its parents are somehow nontrivially dependent then the Bayesian score of such structure is higher when compared to a structure which is the same except for missing some edge between $Parents(X_i)$ and X_i.

[4] By a "sub-CPD" of $P(X \mid Parents(x))$ a mean a distribution over variable X for a concrete assignment $\mathbf{pa}$ of variables $Parents(X)$. From its nature, it is a valid probability distribution (sums to one, always non-negative).

In practice, we take a logarithm of (3.10) because computation of $log\,\Gamma(x)$ is numerically manageable whereas $\Gamma(x)$ of a big number may be infinity in double precision arithmetics (similar to $n!$ for a big value of n) [12, p. 801].

Specifying structure and parameter priors

To be able to use Bayesian scores we also need to define prior distribution over structures $P(\mathcal{G})$ and over parameters $P(\Theta \mid \mathcal{G})$. As has already been said, prior over structures doesn't depend on the size of our dataset, so it plays rather minor role. Nonetheless, it can make a difference for a particularly small dataset. The suggestion of literature [12, p. 804] is to use a prior penalizing the network complexity in the general form $P(\mathcal{G}) \propto c^{Dim(\mathcal{G})}$ where $c \in (0, 1)$, or a completely uniform prior (corresponding to $c = 1$).

When speaking about prior distribution over parameters Θ given concrete structure $\mathcal{G}$ (and thereby speaking about determining hyperparameters α for a concrete structure) the key problem is to represent the prior in some space-efficient form because the naive approach would be to define a prior for every possible structure $\mathcal{G}$. A simple approach is to use a Dirichlet distribution with uniform prior, so called *K2 prior*. Unfortunately, K2 prior is from its nature inconsistent in the sense that the equivalent sample size α depends on the number of parents of a node which doesn't make sense [12, p. 806]. As an example, consider a binary variable X and equivalent sample size $\alpha = 2$. If X has no parents, then the we have effectively seen two pseudosamples for X. But if X had a single binary parent Y, the uniform prior would say we have effectively seen four pseudosamples for X, two for $Y = y_0$ and two for $Y = y_1$.

Another way to approach the parameter prior is to encode the prior distribution P' by a BN and to infer the terms $\alpha_{x|\mathbf{p}} = \alpha \cdot P'(x, \mathbf{p})$ as they are needed. The latter option is called the *BDe prior* [12, p. 806] and its main advantage is that we can use a single network to encode a general prior distribution over parameters for any structure $\mathcal{G}$. Also, it doesn't suffer from the inconsistency as K2 prior does. On the other hand, BDe prior relies on inference which is a non-trivial task in terms of time complexity.

3.5.5 Learning specific structures

We already know the necessary specifics of evaluating how well a graph structure matches our data. Now we will discuss the specifics of learning two types of structures – trees/forests and general graphs.

Learning a tree-structured network

Let a tree-structured network be any network such that every node has at most one parent. Then for a tree-structured network the likelihood score or the BIC score doesn't distinguish orientations of the edges because in this situation the mutual information for any combination of edge orientations and for any two variables is exactly the same. Therefore we can for each pair of variables compute the difference of BIC score between the two situations when these two variables are directly connected and when not. Now, if the BIC score difference represents weight of an edge, we can compute the maximal spanning tree using Kruskal's algorithm or similar and finally remove edges with non-positive weights, thereby obtaining a forest. The whole procedure is carried out in polynomial time $O(n^2)$ as opposed to NP-hardness of a general structure learning.

The best-scoring forest is useful, for example, as a starting point for search of a general graph structure. Trees are also not prone to overfitting because the structure doesn't allow for a complicated hypothesis and CPD fragmentation.

Learning a general graph

The search for a graph with general structure has already been characterized as an optimization task of maximizing score of the whole structure. We usually explore the space of all hypotheses using some greedy algorithm (best-first-search, hill climbing, tabu search etc.) but, in principle, other more sophisticated optimization methods are applicable as well. The search starts with some initial structure which can be graph with no edges, the best-coring forest, random graph or structure capturing our prior knowledge and believes regarding the target domain. The search uses the following three operators: edge addition, edge removal and edge reversal. The problem of local maxima in the context of greedy algorithms is usually addressed by introducing a tabu list or by random restarts making a number of random transformations regardless the score difference. Another problem that often arises is the problem of *plateaux* which means that structures with low edit-distance often have the same score (reversing an edge in a tree etc.). A plateau effectively makes the search space locally "flat" in terms of the scoring function and the space search algorithm can't pick the right direction to go. It turns out that the problem of plateaux is also solvable by random restarts or by a tabu list [12, p. 815].

When considering computational complexity of greedy search, let's remember that the scoring functions are decomposable with the structure of the graph. So, in order to evaluate a structural change, we only need to consider local change of score in the nodes affected by this change and score of the rest of the network remains the same [12, p. 818]. Furthermore, the state-space exploration has local character because the structure changes only locally by applying the edge addition/removal/reversal operators. Therefore, in consequent search steps we often need to reexamine many structural changes again to pick the best one and, in this case, caching of previously examined changes leads to a significant speedup [12, p. 819].

3.5.6 Method for finding all possible alterations of a network

During structure learning, at some point we always need to determine the set of all possible alterations of a concrete network, ie. which edges may be added, removed or reversed. Method of determining this set should be time-efficient as well as mathematically plausible. With these criteria in mind I have devised the following approach.

The main problem we have to face is that an alteration mustn't introduce a directed cycle into the network because Bayesian networks are purely DAGs. Let r be binary relation over the set of all nodes such that $\forall X, Y : XrY \Leftrightarrow X \in Parents(Y)$. By convention, r^+ denotes transitive closure of the relation r. Let's now inspect the three possible cases of structure alteration – edge removal, addition and reversal. In the following text we always assume arbitrary variables X, Y such that $X \neq Y$.

Removing an edge

Condition for removing an edge (X, Y) is trivial – it can be removed iff the edge is present in the network (no cycle may be introduced by removing an edge).

Formally, edge (X, Y) may be removed iff XrY.

Adding an edge

Adding an edge is not a trivial operation since it may introduce an oriented cycle. At this point we will exploit the r^+ relation to inspect whether there exists an oriented path from Y to X in the original network. If $Y r^+ X$ then adding edge (X, Y) would introduce a cycle.

Formally, edge (X, Y) may be added iff $\neg(Y r^+ X) \wedge \neg(X r Y)$.

Reversing an edge

To tackle the problem of edge reversion we make use of a topological ordering of the given network, denoted $\prec$. An existing edge (X, Y) may be reversed iff there is no other directed path from X to Y other than directly using the edge (X, Y). Suppose there were a directed path $X = U_0 U_1 \dots U_{n-1} U_n = Y$ where $n \geq 2$, then because of edges (U_{i-1}, U_i) would have to hold $\forall i \in \{1, \dots, n-1\} : (X \prec U_i \prec Y) \wedge (X\, r^+\, U_i\, r^+\, Y)$. So, we just need to inspect nodes U_i in the topological ordering such that $X \prec U_i \prec Y$ and check if they all fail to satisfy $X\, r^+\, U_i\, r^+\, Y$. Even better, we may narrow down the search to just $U_i \in Children(X)$ because then $X\, r^+ U_i$ is satisfied trivially and we just need to check whether $U_i\, r^+\, Y$. But then, of course, we have to store the information about position in topological ordering within the data structure of each node.

Formally, edge (X, Y) may be reversed iff $(X\, r\, Y) \wedge \left(\nexists U : X \prec U \prec Y \wedge X\, r^+\, U\, r^+ Y\right)$, where $\prec$ is topological ordering of the original network.

In the text above I have proposed a method for efficient determination of all possible alterations of a Bayesian network using transitive closure r^+ of adjacency relation r (computable by Warshall's algorithm) and topological ordering $\prec$ (computable by a DFS-based algorithm [4]).

3.6 I-equivalent structures

This thesis introduced structure learning as an optimization task of finding a network structure that maximizes some scoring function. One potential problem is that there might be multiple network structures having exactly the same score and then the learning algorithm finds not just one best-scoring network but a whole set of networks. Such collision in score is dependent on the scoring function we use and, as I will explain shortly, it is actually a desired property of the scoring function to produce such "collisions".

If we were to say about two network structures $\mathcal{G}_1, \mathcal{G}_2$ (with CPDs) that they are, in a sense, equal they would have to be able of inducing the same probability distribution. Such situation happens when the set of independencies implied by the network $\mathcal{G}_2$ is a subset of independencies implied by the structure of $\mathcal{G}_1$ [12, p. 76], ie. $I(\mathcal{G}_2) \subseteq I(\mathcal{G}_1)$ where $I(\mathcal{G})$ denotes set of independencies (both conditional and marginal) implied by structure $\mathcal{G}$. If $\mathcal{G}_2$ encodes less independencies than $\mathcal{G}_1$ then it is fine because $\mathcal{G}_2$ can still capture any probability distribution induced by $\mathcal{G}_1$. Say both the networks contain variables X, Y only structure of $\mathcal{G}_1$ is discrete and structure of $\mathcal{G}_2$ is $X \to Y$. So X and Y are marginally independent in any distribution $P_1(X, Y)$ induced by $\mathcal{G}_1$, in $\mathcal{G}_2$ the independency of X and Y clearly isn't necessarily true and the condition $I(\mathcal{G}_2) \subseteq I(\mathcal{G}_1)$ is satisfied. At this point in any distribution P_2 induced by $\mathcal{G}_2$ we can always put $P_2(Y \mid x_0) = P_2(Y \mid x_1)$ thereby making X and Y independent in $\mathcal{G}_2$. We can see that $\mathcal{G}_2$ can capture any probability distribution induced by $\mathcal{G}_1$. However, in the opposite direction this statement doesn't hold true.

If for two structures $\mathcal{G}_1, \mathcal{G}_2$ holds that $I(\mathcal{G}_1) = I(\mathcal{G}_2)$ we say $\mathcal{G}_1, \mathcal{G}_2$ are *I-equivalent* (as in independency equivalent) [12, p. 76]. The reason why I mention the notion of I-equivalence is that I-equivalent network structures have the same BIC score and, under certain conditions, the same Bayesian score [12, p. 807]. That means that the structure learning process may yield not only one but a whole set of best-scoring network structures. Furthermore, these networks are not only equivalent in terms of score but also in terms of causal relationships among variables as will be demonstrated in Chapter 5 in further detail. To briefly elaborate, consider three networks $A \to B \to C$, $A \leftarrow B \to C$ and $A \leftarrow B \leftarrow C$. In these three networks holds one and only one conditional independency $A \perp C \mid B$ and no marginal independency, therefore these three networks are I-equivalent. As a result, theese networks are capable of representing the same set of probability distributions and, practically speaking, these three network structures are equally good as a result of structure learning because there is really no causal relationship between variables A, B, C, at least none that could be derived purely from data.

Chapter 4

Implementation

This chapter will present the program realization of this thesis and should be also considered program documentation and reference for anyone who intends to write an extension.

The implementation relies heavily on design patterns and object-oriented programming principles to be flexible and easily extensible, ideally with no need to modify the existing code. For better understanding, I will describe the inter-class dependencies in terms of names of used design patterns as presented in [6].

4.1 Package structure

The application and library is decomposed into the following packages:

- `bna.bnlib`: Classes for internal representation of a Bayesian network as a set of variables, their dependencies and CPDs represented as factors. Also contains all library exceptions.

- `bna.bnlib.io`: Classes for reading/writing `.net` files with Bayesian network specification, writing `.gv` files for BN structure visualization using Graphviz and reading/writing `.csv` files with datasets. Also contains logic for parsing textual probabilistic queries.

- `bna.bnlib.learning`: Parameter learning and structure learning classes.

- `bna.bnlib.misc`: Classes that are inconvenient to put elsewhere (cache implementation, multipurpose toolkit, a general digraph representation).

- `bna.bnlib.sampling`: Classes for stochastic inference by sampling (weighted samping and Markov chain Monte-Carlo).

- `bna.bnlib.view`: All the GUI logic; mostly visualization, network layout generation, dialogs, GUI components etc.

4.2 Data structures

This section introduces the key classes used for internal representation of a Bayesian network – data structures `Variable`, `Factor`, `Node`, `BayesianNetwork` and auxiliary objects `VariableSubsetMapper` and `AssignmentIndexMapper`.

A variable, represented by the `Variable` class, has two components: a unique name (attribute `String name`) and a non-empty list of mutually unique assignments (attribute `String[] values`). Assignment of a variable is one of its values, ie. a string, but because operations over strings (most of all equality) would be costly, an assignment is internally represented as an integer which is interpreted as an index to the `values` array.

A factor, represented by the `Factor` class, has a scope, which is a list of variables `Variable[] scope`, and defines a mathematical function from the set of possible assignments of its scope to the set $\mathbb{R}_0^+$. This mathematical function is internally represented by a 1D real vector `double[] values`. Mapping of a concrete assignment `int[] assignment` to an integral index into the `values` vector (and vice versa) is defined as follows: `values[0]` corresponds with the assignment $(0, \ldots, 0)$, then the subsequent assignments corresponding with `values[1]`, `values[2]` etc. are obtained by incrementing assignment of the leftmost variable which may possibly overflow and cause to increment integer assignment of the next variable, ie. assignment of the leftmost variable changes the most rapidly. To make this mapping efficient there is an extra class `AssignmentIndexMapper` that keeps some internal information which needs to be computed only once, not with every assignment-index or index-assignment transformation.

A node, represented by the `Node` class, is a component of a Bayesian network. It holds a variable, a CPD in the form of a factor with scope $\{X\} \cup Parents(X)$ and structural information – parent nodes and child nodes. An instance of the `BayesianNetwork` is nothing more than a set of nodes.

A frequent operation is transformation of an assignment of a list of variables to an assignment of a subset of these variables (eg. extracting assignment of parent variables during sampling, determining some counter $N_{x,\mathbf{pa}}$ from a dataset etc.). This transformation includes two operations: (1) extraction of a subset of the original assignment (for each subset variable we need to know on which index this variable is in the superset) and (2) transforming the integer value representing assignment of a variable X in the superset to another integer representing the same assignment of the same variable X in the subset; this step is necessary because there may be two same variables (same name and same set of possible values) having different ordering of their possible values and therefore the integer values serving as indices into the `values` arrays don't match.

4.3 Exceptions

In the whole `bnlib` library all exceptions are unmanaged and share a single base exception `BNLibException`. Each method specifies a list of throwable exceptions via the `throws` keyword and its javadoc comment explains conditions under which the listed exceptions are thrown.

4.4 Input, output

There are two types of input/output operations. The first class of operations deals with specification of a Bayesian network, the other class is for datasets. There are generic format-independent classes in the `bna.bnlib.io` package that employ template method pattern to enable extensibility for currently unsupported file formats.

Bayesian networks can be imported and exported in the `.net` (also known as LibB) format. The file itself contains only the network structure and CPDs, graphical layout of

a network is always automatically generated. The format is very intuitive and, I believe, rather then providing formal specification the direct approach of looking at some existing file is a lot faster way to understand the format. I will provide just a few comments. A `.net` file contains three sections. The first section `net {...}` is ignored. The second section is an enumeration of all variables and their possible values. The third section specifies the network structure and CPDs; the CPDs are basically written as 1D arrays with exactly the same value ordering logic as I use in the `Factor` class (see Section 4.2). Parentheses in the CPD specification are mandatory.

For datasets I use a CSV-like format. The first line must be a header that specifies for each column its name (variable name) and complete list of possible values in this column. The format of a single header record is `<variable-name>(<value1>|...|<valueN>)`. Records in the whole file, including header, are separated by a specified character.

4.5 Sampling

Sampling is implemented by classes in the `bna.bnlib.sampling` package. Basically, there are two parallel hierarchies of objects which will be further described in detail.

One family of objects with common base class `SampleProducer` aims to provide samples from a general distribution $P(\mathbf{X} \mid \mathbf{Y}, \mathbf{E} = \mathbf{e})$ where $\mathbf{X} \neq \emptyset$. Two sampling approaches are implemented – weighted sampling (class `WeightedSampleProducer`) and Markov chain Monte-Carlo (class `MCMCSampleProducer`).

Concern of the second family of objects, that share common interface `SamplerInterface`, is what to do with generated samples. We can maintain statistics from a sufficient number of samples and in the end compute an approximation of the real distribution $P(\mathbf{X} \mid \mathbf{Y}, \mathbf{E} = \mathbf{e})$ (class `QuerySampler`); the same task may be carried out in multiple threads at once (class `QuerySamplerMultithreaded`). Another goal might be to create an artificial dataset by sampling all the variables in a given BN and to write such dataset to a file in CSV format (class `DatasetCreationSampler`) or to store the samples in memory within a `Dataset` instance. Dataset creation is useful when benchmarking learning algorithms for Bayesian networks – we already have a BN and we attempt to learn that network again based just on dataset of chosen size, finally we compare the network just learnt with the original one according to some metric.

4.5.1 Producing a single sample

Generally, a sample can be produced using many different sampling techniques (eg. direct sampling, weighted sampling, MCMC etc.) and the abstract class `SampleProducer` is a common base class for such specific samplers. My implementation uses template method design pattern to enable easy implementation of custom sampling methods. The class mainly keeps the specification of the distribution $P(\mathbf{X} \mid \mathbf{Y}, \mathbf{e})$ we want to sample and reference to the BN to be sampled.

The class `SampleProducer` and its subclasses use an instance of `SamplingContext` to maintain thread-local variables needed for sampling – array to hold assignment of variables in the network (to avoid frequent allocation) and a random number generator (importance will be explained later).

List of noteworthy methods of the `SampleProducer` class:

- `void determineSamplingOrder()`: A template method that computes list of variables whose values we need to keep track of during sampling and also determines the order of sampling, ie. specific sequence of variables. First of all, the order of sampling is computed using a linear time DFS-based topological sort algorithm described in [4]. Then there is a hook `filterVariablesToSample(...)` by which the specific samplers (children of `SampleProducer`) can reduce the list of variables we need to take into account during sampling – this is useful for example in the case of weighted sampling. The final array of variables, whose values need to be known/determined during sampling, is stored in the instance variable `sampledVars`.

- `Variable[] filterVariablesToSample(Variable[] allVars)`: A hook of the template method `determineSamplingOrder()` described above. By default returns the argument unchanged (ie. no filtering).

- `void initializeSample(SamplingContext context)`: Abstract method in which a concrete sampler may prepare the assignment of variables before the first sample is produced (needed by the MCMC sampling to set the evidence variables to given values). Specific values in the `context.sampledVarsAssignment` array are instantiations of variables of the array in `SampleProducer.sampledVars` on the same indices.

- `void produceSample(SamplingContext context)`: Abstract method to produce one sample consisting of a concrete assignment to variables and weight of the sample. The sample is stored in the `context` argument. The `context.sampledVarsAssignment` array is filled with a concrete instantiation of corresponding variables of the array `SampleProducer.sampledVars`.

- `SamplingContext createSamplingContext()`: Creates a thread-local context for sampling. This context is meant to be reused when generating a number of samples using the same instance of `SampleProducer`.

Weighted sampling

Weighted sampling is implemented by the class `WeightedSampleProducer`. As it is a non-abstract direct child of `SampleProducer` it needs to define methods `initializeSample(...)`, `produceSample(...)` and it also redefines the hook `filterVariablesToSample(...)` for optimization purposes.

List of noteworthy methods of the `WeightedSampleProducer` class:

- `Variable[] filterVariablesToSample(Variable[] allVars)`: At this point the implementation of weighted sampling takes advantage of the fact that, in order to produce relevant samples for determining the distribution $P(\mathbf{X} \mid \mathbf{Y}, \mathbf{e})$, we don't have to consider variables such that they are not in $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{E}$ and also none of their descendants is in $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{E}$. So, this method filters the given array of variables and preserves only those variables that don't satisfy the previously stated conditions. This way weighted sampling can be accelerated for those queries in which a significant part of the BN doesn't need to be sampled at all.

- `void initializeSample(SamplingContext context)`: Empty method – in weighted sampling values of all variables are rewritten with each generated sample, so there is no need to produce some initial instantiation of variables.

- `void produceSample(SamplingContext context)`: The weighted sampling is implemented as follows: We keep an array of abstract `WeightedSamplingAction` objects. Each object is responsible for handling one variable of the `sampledVars` array (ie. the array of variables that need to be taken into account) so, one sample is produced by performing all the sampling actions in a sequence.

  The associated action objects are different for evidence and for non-evidence variables. For an evidence variable E_i the associated `WeightedSamplingEvidenceAction` object extracts assignment of the variables $Parents(E_i)$ from the current sample (ie. current values in `assignment`) and determines the probability $P(E_i = e_i \mid parents(E_i))$ which will affect the weight of the sample. For a non-evidence variable X_i the associated `WeightedSamplingVariableAction` extracts assignment of the variables $Parents(X_i)$ from the current sample and then samples the variable X_i from distribution $P(X_i \mid parents(X_i))$; sampling itself is actually implemented in the `Node` class. Finally, the value x_i of variable X_i is written to `context`.

Markov chain Monte-Carlo sampling

MCMC sampling is implemented by the class `MCMCSampleProducer`. As a subclass of `SampleProducer` it has to define methods `initializeSample(...)` and `produceSample(...)`.

List of noteworthy methods of the `MCMCSampleProducer` class:

- `void initializeSample(SamplingContext context)`: MCMC actually is a sampling method that views the last generated sample as a state of the network (concrete instantiation of variables) and the next sample is produced by perturbing this state. So, in initialization we need to produce some valid instantiation of all `sampledVars` variables. Such instantiation has to (1) respect given evidence and (2) be a “normal” sample (not something very improbable). Therefore we internally set all evidence variables to the observed values and then resample each non-evidence variable a few times, so that the initial state isn’t a very unlikely one.

- `void produceSample(SamplingContext context)`: Sample is in MCMC produced by taking the last generated sample and resampling a single variable while values of the other variables remain unchanged. For the purpose of resampling we keep an array named `resamplingActions` with one instance of `MCMCResamplingAction` for each non-evidence variable (evidence variable cannot be resampled). New sample is then produced by performing a randomly chosen resampling action.

  The `MCMCResamplingAction` for a variable X works as follows: For each instantiation $x \in val(X)$ we compute the probability $P(x \mid parents(X)) \prod_{C_i \in Children(X)} P(c_i \mid parents(C_i))$ – we don’t need to consider variables other than $X \cup Children(X)$ because for other nodes different values x of X don’t affect probability at these nodes given parents. Then by normalization we obtain the distribution $P(X \mid \text{rest-of-sample})$ and by sampling X from this distribution we obtain the new value of X which is written to the `context` object.

4.5.2 Friendly query specification

The query $P(\mathbf{X} \mid \mathbf{Y}, \mathbf{e})$, where $\mathbf{X} \neq \emptyset$, may be specified in the constructor of `SampleProducer` and any of its subclasses in a user-friendly way as a string. Format of the query string is described bellow using EBNF-like notation:

```
query            ::  ( "P" , "(" , var-list , ")" )
                  |  ( "P" , "(" , var-list , "|" , var-list , ")" )
                  |  ( "P" , "(" , var-list , "|" , assignment-list , ")" )
                  |  ( "P" , "(" , var-list , "|" , var-list , "," , assignment-list , ")" ) ;
var-list         ::  var , { "," , var } ;
assignment-list  ::  var , "=" , val , { "," , var , "=" } ;
```

Variable name *var* and variable value *val* may consist, for simplicity, of any characters except for `,` `|` `=` `(` `)` and whitespaces. Variable names and values are case sensitive.

Some examples of valid query strings are `P(RAIN)`, `P(CLOUDY | WETGRASS = TRUE)`, `P(SPRINKLER, CLOUDY)` or `P(CLOUDY | RAIN, WETGRASS = TRUE)`.

4.5.3 Processing a larger number of samples

Classes with common ancestor `SampleProducer` described in the previous section are able to generate a single sample on demand using a specific sampling technique. Another problem, treated by the class `Sampler` and its subclasses, is what to do with those samples. The abstract class `Sampler` defines a general sample-handling framework using template method design pattern. In constructor it recieves a `SampleProducer` instance which is later used to produce larger number of samples. `Sampler` has within its `sample(...)` method a loop that generates a large number of samples using the concrete `SampleProducer` and for each generated sample the method `void registerSample(int[] assignment, double weight)` is called; within this method a subclass defines what should be done with a sample (write it to a CSV file, insert into a dataset, account for in some statistics etc.).

List of noteworthy methods of the `Sampler` class:

- `final void sample(SamplingController controller)`: Main sampling loop which generates samples one by one and for each invokes the `registerSample(...)` method. The `controller` is used to limit the number of generated samples as well as to instantly stop the sampling (typically from user interface). This is a template method and uses the following abstract methods that need to be defined within a subclass: `presamplingActions()`, `postsamplingActions()` and `registerSample(...)`.
- `abstract void presamplingActions()`: Method called from `sample(...)` before the first sample is produced, ie. before the first invocation of `registerSample(...)`.
- `abstract void postsamplingActions()`: Method called from `sample(...)` after the last sample is produced, ie. after the last invocation of `registerSample(...)`.
- `abstract void registerSample(int[] assignment, double weight)`: In this method a subclass defines what to do with a generated sample. The `assignment` array contains concrete values of variables $\mathbf{X}, \mathbf{Y}$; order of those variables is the same as was specified in the constructor of used `SampleProducer` instance.

Answering a probabilistic query

To provide an approximation of a general distribution $P(\mathbf{X} \mid \mathbf{Y}, \mathbf{e})$ based on a number of generated samples we use an instance of `QuerySampler` which is a subclass of `Sampler` and as such has to implement its abstract methods, mainly `registerSample(...)`. The class maintains a counter for all instantiations of variables $\mathbf{X}, \mathbf{Y}$ and, using statistics, computes an approximation of the desired probability distribution in form of a factor.

Creating an artificial dataset

General framework provided by the abstract `Sampler` class may be also used to create an artificial dataset through sampling of an existing network. Again, abstract methods of `Sampler` have to be defined be the subclasses. Subclass `DatafileCreationSampler` opens a CSV file within the method `presamplingActions()`, in `registerSample(...)` writes values of each sample to the file and in `postsamplingActions()` closes the output file. Subclass `DatasetCreationSampler` keeps all samples in memory as an instance of the `Dataset` class.

Multi-threaded sampling

Class `QuerySamplerMultithreaded` is a multi-threaded variant of the class `QuerySampler` and also aims to compute an approximation of distribution $P(\mathbf{X} \mid \mathbf{Y}, \mathbf{E} = \mathbf{e})$ through computing statistical properties of samples obtained from a given BN. We pass to the constructor of `QuerySamplerMultithreaded` a `SampleProducer` object that provides samples on demand and the requested number of threads. The sampler instance creates and executes threads that perform sampling in parallel, each using a standalone `QuerySampler` instance, and finally their results are combined. For simple centralized control of the sampling carried out in multiple threads a single instance of `SamplingController` is used. This object keeps the information about the number of samples to generate and also can be used to instantly stop the sampling process in all threads by invoking `controllerInstance.setStopFlag()`, typically through the user interface.

Speedup of the multi-threaded variant has been measured on multiple computers with different processors (see Table 4.1). On a multi-processor system the speedup is significant, eg. on a system with 4 physical and 8 logical cores the speedup is according to measurements virtually 2.0 for two threads and about 3.8 for four threads.

4.5.4 Implementation notes

In this section I will discuss some of the difficulties I encountered during implementation. Also, the code snippet in Figure 4.1 demonstrates the usage of classes described earlier to answer a probabilistic query via sampling.

```
final long SAMPLES_COUNT = 10 * 1000;
final int THREAD_COUNT = 3;
final String NETWORK_FILE = "networks/sprinkler.net";
final String QUERY_STR = "P(RAIN | SPRINKLER = TRUE, WETGRASS = TRUE)";
long samplesPerThread = SAMPLES_COUNT / THREAD_COUNT;

BayesianNetwork bn = BayesianNetwork.loadFromFile(NETWORK_FILE);
SampleProducer sampleProducer = new WeightedSampleProducer(bn, QUERY_STR);
SamplerInterface sampler = new QuerySamplerMultithreaded(sampleProducer,
                                                         THREAD_COUNT);
SamplingController controller = new SamplingController(samplesPerThread);
sampler.sample(controller);
Factor result = ((QuerySamplerMultithreaded)sampler).getSamplesCounterNormalized();
```

Figure 4.1: Usage of the BN library classes to perform inference by sampling.

		Number of threads					
		1	2	3	4	6	8
Intel i5-2450M	**Samples/sec**	3.068e6	5.564e6	6.061e6	5.481e6	5.537e6	5.521e6
(2.5 GHz, 2 cores)	**Speedup**	1.00	1.81	1.98	1.79	1.80	1.80
2x Opteron 2387	**Samples/sec**	2.429e6	4.765e6	6.911e6	9.160e6	10.803e6	11.672e6
(2.8 GHz, 4 cores)	**Speedup**	1.00	1.96	2.85	3.77	4.45	4.81
Xeon E5-2640	**Samples/sec**	1.977e6	4.106e6	5.867e6	7.307e6	9.190e6	10.739e6
(2.5 GHz, 6 cores)	**Speedup**	1.00	2.07	2.97	3.70	4.65	5.43

(a) Weighted sampling of the classical "sprinkler network" for query $P(Rain \mid WetGrass = true)$.

		Number of threads					
		1	2	3	4	6	8
Intel i5-2450M	**Samples/sec**	6.807e5	10.734e5	11.472e5	12.447e5	12.317e5	12.174e5
(2.5 GHz, 2 cores)	**Speedup**	1.00	1.57	1.69	1.83	1.81	1.79
2x Opteron 2387	**Samples/sec**	4.880e5	9.786e5	13.718e5	18.237e5	26.643e5	32.237e5
(2.8 GHz, 4 cores)	**Speedup**	1.00	2.00	2.81	3.73	5.46	6.61
Xeon E5-2640	**Samples/sec**	4.384e5	9.398e5	13.775e5	17.493e5	23.337e5	27.361e5
(2.5 GHz, 6 cores)	**Speedup**	1.00	2.14	3.14	3.99	5.32	6.24

(b) Weighted sampling of the "ICU alarm network" for query $P(PVSAT \mid ECO2, SAO2 = high)$.

Table 4.1: Benchmark results of multi-threaded sampling speedup on various processors. Each value *samples/second* was computed as follows: 20 million samples divided by the trimmed-mean computation time of sampling from total of 25 runs.
The first machine is my notebook and the others are student servers merlin (Linux) and krok (SunOS) at the university. We can see that in case of krok the measurements suggest an impossible speedup. I believe this issue may be caused partly by inaccuracy of measurements and, perhaps, by internal optimization of the time-intensive code which could have added some execution time to the single threaded experiment and at the same time accelerate the consequent multi-threaded experiments.

During implementation there have been some difficulties regarding thread-safety and sharing of resources among threads. Considering the overall performance of sampling, the ultimate problem turned out to be that generating pseudorandom numbers consumes the majority of time of sampling and therefore an instance of `java.util.Random` (further just `Random`) cannot be shared among threads. Because the method `Random.next(int bits)` for generating pseudorandom bits is synchronized, sharing a `Random` instance among n threads degrades computation time to virtually running the sampling n-times sequentially. The problem is easily solved by using `java.util.concurrent.ThreadLocalRandom` class that provides the caller with a thread-local instance of `Random` on demand. Still, generating random data remains the bottleneck of sampling.

Also, we cannot save time by storing references to dynamically allocated objects (eg. arrays) holding intermediate results if theses objects aren't read only, since concurrent threads would rewrite each others values. Fortunately, it turns out that frequent memory allocation for such intermediate results in Java doesn't introduce a bottleneck.

4.6 Structure learning

To perform structure learning in the form of local optimization of a scoring function we need a state-space search algorithm, a dataset and implementation of a scoring function. All these components are implemented separately as will be explained in the following text.

4.6.1 Learning algorithm

A general state-spate search algorithm, that explores the space of feasible network structures, is represented by the abstract class `StructureLearningAlgorithm`. There is only one concrete implementation, namely the `TabuSearchLearningAlgorithm` class implementing tabu-search with random restarts. The learning itself is controlled via an instance of `LearningController` which specifies the number of steps of the local search and also can be used to stop the learning process instantly by invoking `setStopFlag()` method, typically from GUI. The learning algorithm also receives restrictions of feasible structures as an instance of `StructuralConstraints`; the structural restrictions are represented by a boolean matrix in which we may selectively disallow directed edge for any pair of variables.

4.6.2 Scoring functions

A scoring function is the essential ingredient of structure learning. A general scoring method is represented by the abstract `ScoringMethod` class, if the score is decomposable there is a specialized abstract subclass `DecomposableScoringMethod` whose direct descendants are `BICScoringMethod` and `BayesianScoringMethod`. Any non-abstract subclass of the `DecomposableScoringMethod` class has to define two methods: `computeFamilyScore(...)` and `computeComplexityPenalty(...)`.

To make structure learning bearable in terms of time requirements we need to employ clever caching that takes advantage of score decomposability. This thesis considers two scoring functions – BIC score and Bayesian score:

$$score_{BIC}(\mathcal{G} : \mathcal{D}) = \sum_{X_i \in Nodes} N \cdot I_{\hat{P}}(X_i; Parents(X_i)) - \frac{\log N}{2} Dim[\mathcal{G}]$$

$$score_{B}(\mathcal{G} : \mathcal{D}) = \sum_{X_i \in Nodes} \left[\sum_{\mathbf{pa}} \left(log \frac{\Gamma(\alpha_{X_i|\mathbf{pa}})}{\Gamma(\alpha_{X_i|\mathbf{pa}} + N_{\mathbf{pa}})} + \sum_{x \in val(X_i)} log \frac{\Gamma(\alpha_{x|\mathbf{pa}} + N_{x,\mathbf{pa}})}{\Gamma(\alpha_{x|\mathbf{pa}})} \right) \right] + logP(\mathcal{G})$$

Both of these scores have the nice quality that they decompose over network structure (over each node X_i) and, furthermore, they have the same format of a sum over all nodes plus some term penalizing structural complexity. So, the scores can be rewritten in a unified format (suggested in [12, p. 818]):

$$score(\mathcal{G} : \mathcal{D}) = \sum_{X_i \in Nodes} FamScore(X_i, Parents(X_i)) + Penalty(\mathcal{G})$$

As we can see, the family score $FamScore(\cdot)$ of a node depends on its parents, ie. on network structure. Also, family score is in both our scoring functions computed by a pass through the dataset, therefore the change of family scores is the value that should be cached for a network alteration. The penalty term regarding network complexity is relatively easy to compute.

Caching exploits the property that most of the possible structural alterations remain the same in consecutive steps of local search, we only need to employ careful bookkeeping. For one, we need to cache the change of family scores for each network alteration that is feasible in the current state of local state-space search; for two, when a structural alteration is accepted we need to drop those cached changes of family scores that are invalidated by the structural change. This way the cache of delta family scores can be maintained at minimum size and remain effective. To achieve the correct dropping behavior we need to understand how the delta family score is computed for the three possible network alterations – edge addition, removal and reversal (for reference please see the definition of BIC or Bayesian score above):

- For addition or removal of edge (X, Y) the delta family score is the increase of $FamScore(Y, Parents(Y))$ because set of parents is unchanged for all nodes except for Y. So, whenever the action accepted in the current step of local search changes the set $Parents(Y)$ all cached delta family scores for actions add or remove (X, Y) (for any X) have to be removed from cache because those cached values are no longer valid and need to be recomputed when needed.

- For reversal of edge (X, Y) the delta family score is the increase of two terms: $FamScore(X, Parents(X))$ and $FamScore(Y, Parents(Y))$ because the two variables' sets of parents are affected. In this case, whenever the alteration action currently taken changes either of the sets $Parents(X)$ or $Parents(Y)$, all cached delta family score values for action reverse (X, Y) have to be dropped.

Size of the delta family score cache is upper-bounded by the number of possible alterations. For a network with n nodes I determined the upper bound of possible alterations to be $n(n-1)$. To see why let's consider a general BN and let $v_1, \ldots, v_n$ be topological ordering of the network that cannot be violated. Then, from any node v_i there may be edges $v_j \to v_i$ for $j < i$ and $v_i \to v_j$ for $i < j$, which allows up to $n(n-1)/2$ edges. For any concrete BN the sum of possible edge additions and removals is upper-bounded by $n(n-1)/2$ since edges that are present in the network can be removed and the complementary edges allowed by the topological ordering, but not currently present, can be added. For edge reversal there probably is an even better upper bound but it is surely at most $n(n-1)/2$ since that is the number edges in a DAG with maximum connectivity and we may reverse only an edge already present in the network (only in this simple consideration we don't address the question of introducing a cycle). Thereby we obtain an upper bound for the overall count of alterations as $2 \cdot n(n-1)/2$.

4.6.3 Implementation notes

This short section provides a code snippet (see Figure 4.2) that demonstrates the usage of classes described earlier to perform structure and parameter learning.

4.6.4 Dataset

The `Dataset` class has already been introduced in context of sampling. For structure learning is important that the `Dataset` class provides methods for computing the mutual information $I_{\hat{P}}(X; Parents(X))$, where $\hat{P}$ is the empirical distribution induced by this dataset, and for counting number of occurrences of an event $\mathbf{x}$, ie. determining some value $N_{\mathbf{x}}$. Furthermore, to accelerate the learning process it is very useful to cache the

```
DatasetFileReader datasetReader = new DatasetCSVFileReader(FILENAME, SEPARATOR);
Dataset dataset = datasetReader.load();
Variable[] allVars = dataset.getVariables();
dataset = new CachedDataset(dataset, LRU_CACHE_SIZE);

LearningController controller = new LearningController(MAX_ITERATIONS);
StructuralConstraints constraints = new StructuralConstraints(allVars);
constraints.setMaxParentCount(MAX_PARENT_COUNT);
BayesianNetwork bnDiscrete = new BayesianNetwork(allVars);

DecomposableScoringMethod scoringMethod = new BICScoringMethod(dataset);
StructureLearningAlgorithm learningAlgorithm = new TabuSearchLearningAlgorithm(
        scoringMethod, TABU_LIST_SIZE, RANDOM_RESTART_STEPS);
BayesianNetwork bnStructure = learningAlgorithm.learn(
        bnDiscrete, controller, constraints);

BayesianNetwork bnComplete = ParameterLearner.learnBayesianEstimationUniform(
        bnStructure, dataset, ALPHA);
```

Figure 4.2: Usage of the BN library classes to perform structure and parameter learning.

least recent queries over a dataset because a pass through the entire dataset is potentially a costly operation. Also, from the nature of local search, which examines all possible network alterations and picks the best one, majority of the possible alterations will be the same in consecutive steps of local search, therefore majority of the dataset queries will be the same. Usefulness of the cached entries depends on the course the local state-space search takes and as this is impossible to predict, LRU caching strategy is applied.

For a network with n variables and for BIC or Bayesian score the minimal cache size needed to accommodate for all possible dataset queries during a single step of local state-space search is $n+n(n-1)$; derivation of this expression will be provided shortly. Practical experiments have showed that the cache of dataset queries needs to be at least twice the minimal size even with the caching of delta-family-score which effectively prevents a great deal of dataset queries from happening.

The functionality of caching dataset queries is implemented in the `CachedDataset` class by caching proxy design pattern.

The last thing is to derive the upper bound of possible dataset queries during a single step of local search. As with the upper bound for size of delta family score cache, suppose we can make up to $n(n-1)/2$ edge additions and removals in total and up to $n(n-1)/2$ edge reversals. Now let's analyze how the delta family scores are affected by edge addition, removal and reversal (for reference see the definition of BIC or Bayesian score above):

- For addition or removal of edge (X, Y) the delta family score is the increase or decrease in the family score of Y, ie. $FamScore(Y, Pa(Y)^+) - FamScore(Y, Pa(Y))$ in case of edge addition or $FamScore(Y, Pa(Y)^-) - FamScore(Y, Pa(Y))$ in case of edge removal. $Pa(\cdot)$ is a shorthand for $Parents(X)$.

- For reversal of edge (X, Y) both families of X and Y are affected by this structural change, so the delta family score is effectively given as $FamScore(X, Pa(X)^+) + FamScore(Y, Pa(Y)^-) - FamScore(X, Pa(X)) - FamScore(Y, Pa(Y))$.

Now we need to recognize how many different family scores we may need to evaluate in a single step of local search, hence how many dataset queries we may need to perform. Clearly, we need $FamScore(X, Pa(X))$ for every node X, which means n dataset queries. The need for the same value $FamScore(X, Pa(X)^-)$ is shared between edge removal and reversal actions and, both the number of edge removals and edge reversals is upper bounded by $n(n-1)/2$. The value of $FamScore(X, Pa(X)^+)$ may be needed by edge addition or by edge reversal. Let's observe that only one of the actions add/reverse, which would result in extending $Pa(X)$ to $Pa(X)^+$, at a time might be possible. Upper bound of the sum of possible edge additions and reversals is the same as for the sum of edge additions and removals, $n(n-1)/2$. Thereby we obtain the total upper bound of dataset queries in a single step of local search as $n + 2n(n-1)/2 = n + n(n-1)$.

4.7 User interface

When speaking about user interface of the realization outcome of this thesis, we may consider the `bnlib` library interface, which has been described in detail in earlier sections, and also graphical user interface (GUI) of the application presented in this thesis. This section will deal with the latter interpretation. We will briefly take a look at the overall internal logic and at some of its non-trivial parts.

A very important principle the whole GUI is build upon is the state machine principle. All the actions the user is enabled to do at the moment depend on the current state of things – do we have a network loaded, does the current network have valid CPDs, what is the current dataset if any, do the current dataset and the current network contain compatible variables? For example, in order to learn a complete network (both structure and parameters) and then to test its predictive capabilities on a test set we need to do the following steps: Load a training dataset, run structure learning, select a learnt structure so that it becomes the current network, learn CPDs of the current network (still using the training dataset), load the test dataset (containing the same set of variables as the training set) and finally perform the test of predictive power. So, the user is given a toolbox rather than a set of complex dialogs for all each imaginable use case, very much in the spirit of UNIX tools.

To improve the user experience the GUI saves last state of its components to an `.ini` file, ie. window sizes and positions, textfields' contents, learning method that was last used, directory of the last imported network etc.

4.7.1 Network layout

Specific task that needs to be addressed when presenting a BN in graphical user interface is generating graphical layout of given BN so that it would be esthetically pleasing. Under this vague criteria we could imagine, for example, minimizing the number of crossing edges, minimizing edges lengths or forcing the graph or its parts to be symmetric and compact. To generate such layout I used a modification of hierarchical approach described in [2, p. 22]. The basic outline of my method is as follows:

- For a given DAG assign nodes to layers. Let's denote $l(n)$ label of the layer the node n is assigned to. Initially, for nodes without any parents $l(n) = 0$, for the other nodes $l(n) = 1 + max\{l(p) \mid p \in Parents(n)\}$. After the initial layer assignment is complete I move nodes without parents from layer 0 to layer $min\{l(p) \mid p \in Children(n)\} - 1$.

- Break down edges connecting nodes from layers, that are not adjacent, by inserting a dummy node on every layer in-between. This way we achieve that any edge is strictly from some layer i into the layer $i + 1$.
- Optimize the permutation of nodes on each layer so that there would be minimum of crossed edges. At this point we take full advantage of the graph structure from previous step. Note that we don't need to work with absolute positioning of nodes to determine the number of crossed edges. As optimization procedure I use simulated annealing with operators "swap two nodes on the same layer" and "remove a node and reinsert it at randomly selected position within the same layer".
- Determine the absolute positioning within layers, ie. the x-coordinate of each node. I use a grid-like coordinate system with integral scale where x-distance of any two nodes within layer has to be at least 2. This way we elegantly achieve for a node with two children that x position of the parent is between x positions of the children while the sum of edge lengths is the smallest possible. Absolute positioning is an optimization task which I address, again, by simulated annealing with single operator "shift a random node left/right". The operator shifts also other adjacent nodes as needed to maintain the x distance of any two nodes at the value of at least 2. To allow for larger changes the shift magnitude is randomly selected from normal distribution with parameters $\mu = 0, \sigma = 1$ over the set $\{-3, \ldots, 3\} \setminus \{0\}$. The score being optimized during the absolute positioning step is sum of the following terms: squared length of every edge, number of different edge-lengths for each node and for each layer and squared size of the largest gap between nodes in each layer.

Parameters of the optimization process are set at such values that the network layout generation doesn't introduce an unpleasant delay for a reasonably-sized network (for a network with 40 nodes the whole process takes about 1 second). Once the automatic layout is complete, nodes can be manually rearranged by mouse dragging. The application maintains layouts of last ten networks in a LRU cache so that a particular network looks the same when the user doesn't look at too many other networks in-between.

4.7.2 Third party packages

The implementation relies on the following third party products:

- Library *Apache Commons Math v3.3.2* for implementation of gamma function $\Gamma(x)$ and $log\,\Gamma(x)$ (released under Apache License v2.0).
- Library *Ini4j v0.52* for reading and writing configuration `.ini` files (released under Apache License v2.0).
- Collection of classes for vertical rendering of `JTable` column headers from the website http://tips4java.wordpress.com/ (no license restrictions). These classes are placed in the `bna.view.darrylbu` package.

Chapter 5

Selected applications

This chapter will present collection of selected applications suitable for Bayesian network learning. We will always examine the problem itself, the reason why it is interesting, approach to its solution and the achieved results.

5.1 Learning of a known model

Core of the first application is a series of benchmarks of the parameter and structure learning methods presented in this thesis. We will use a collection of well known networks and attempt to relearn them based on artificial datasets sampled from these networks. The goal is to get a feel for optimal learning parameters for networks of various sizes as well as for the quality of obtained results depending of the size of our dataset. These findings will be referred to to provide justification for conclusions stated in the subsequent sections in this chapter.

We will separately examine two classes of problems. The first one is learning parameters for a known network structure and the second one is learning both structure and parameters at the same time. The following networks will be used[1]:

- "Cancer" network (5 variables, 4 edges, 10 CPD entries)
- "Asia" network (8 variables, 8 edges, 18 CPD entries)
- "Child" network (20 variables, 25 edges, 230 CPD entries)
- "ICU alarm" network (37 variables, 46 edges, 509 CPD entries)

5.1.1 Learning parameters

The first step will be to inspect how well maximum likelihood estimation and Bayesian parameter estimation methods work for networks and datasets of various sizes. In this case, inputs of the parameter learning algorithms are the original network structure without CPDs and a dataset; in the case of Bayesian estimation we also need to specify the equivalent sample size α (only uniform prior distribution is considered).

[1] The used networks are freely available in on-line repositories at The Hebrew University of Jerusalem at http://www.cs.huji.ac.il/site/labs/compbio/Repository/networks.html or as a part of an R package for Bayesian networks at http://www.bnlearn.com/bnrepository/. They are also on the enclosed CD in the 1-benchmarks/networks/ directory.

To measure how closely the original and the learnt networks match (respectively probability distributions they induce) I use so called *KL-divergence* (a.k.a. relative entropy) [12]. KL-divergence of two probability distributions P and Q (both over the same set of variables), denoted $D_{KL}(P \parallel Q)$, is defined as follows:

$$D_{KL}(P \parallel Q) = \sum_{X_1,\ldots,X_m} P(x_1,\ldots,x_m) \log \frac{P(x_1,\ldots,x_m)}{Q(x_1,\ldots,x_m)}$$

which can be in context of Bayesian networks rewritten to the following form (P is distribution induced by the original network, Q is induced by the learnt network) [12, p. 273]:

$$D_{KL}(P \parallel Q) = \sum_{X \in Nodes} \sum_{\mathbf{pa} \in val(Par(X))} P(\mathbf{pa}) \sum_{x_i \in val(X)} \log \frac{P(x_i \mid \mathbf{pa})}{Q(x_i \mid \mathbf{pa})}$$

As we can see, KL-divergence nicely decomposes over a BN so that we make use of plain CPD entries rather than exhaustively inspecting probabilities of all atomic events $x_1, \ldots, x_m$ which would take $O(2^m)$ time. However, probability for parents' assignment $P(\mathbf{pa})$ needs to be inferred. In accordance with the requirements of this thesis I use a stochastic inference method, namely weighted sampling, with such number of samples that leads to reasonably accurate results in the KL-divergence analysis[2].

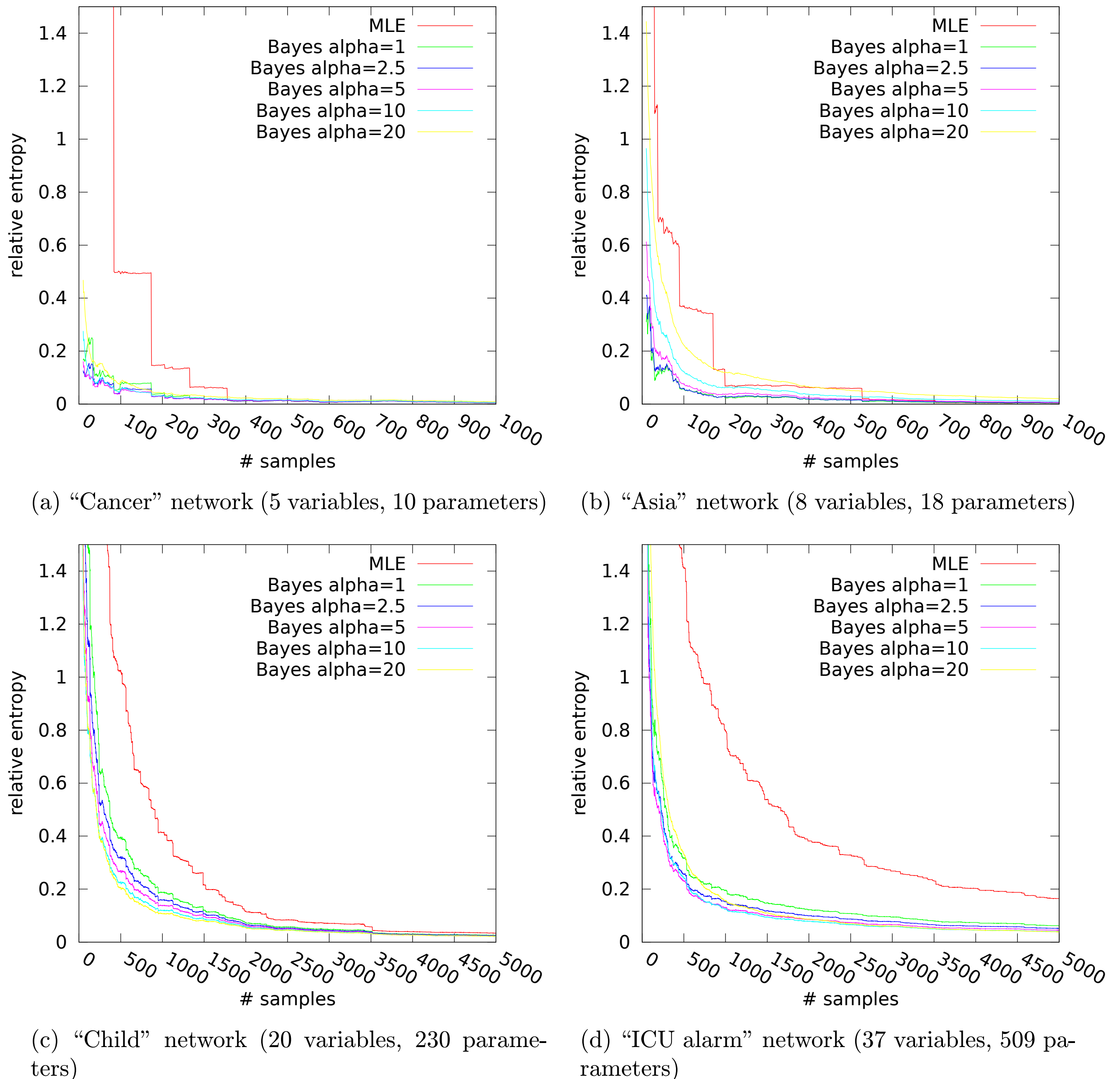
Strictly speaking, the KL-divergence is not a metric as $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$ but it is commonly used for evaluation of how closely two probability distributions match because the true metrics for probability distributions don't decompose over structure of a Bayesian network and therefore are impractical, for larger networks even unfeasible.

From mathematical point of view, the KL-divergence is defined only if for any x_i and any $\mathbf{pa}$ always holds that $\big(Q(x_i \mid \mathbf{pa}) = 0\big) \Rightarrow \big(P(x_i \mid \mathbf{pa}) = 0\big)$, ie. we can't have non-zero P probability and zero Q probability at the same time. However, in practice this is impossible to guarantee when computing network parameters using the maximum likelihood estimation with a small dataset. According to [13, 17], this problem is addressed by artificially setting a small positive lower bound for $Q(\cdot)$ probabilities when $P(\cdot)$ is nonzero. The suggested value is 10^{-6} and its contribution to the KL-divergence can be interpreted as a penalty for probability $Q(\cdot) = 0$ being incorrectly inferred by maximum likelihood estimation. On the other hand, when some of the CPDs in the original network contains a zero, the case of $P(\mathbf{x}) = 0 \wedge Q(\mathbf{x}) \neq 0$ produced by Bayesian estimation with equivalent sample size $\alpha > 0$ is not penalized (in fact, in my experiments doesn't affect the KL-divergence value at all) because the prior for Bayesian estimation can always selectively capture for certain events to be impossible but we select to work with uniform priors as these are orders of magnitude easier to specify.

Graphs in Figure 5.1 capture benchmark results of parameter estimation methods that are considered in this thesis – the maximum likelihood estimation (further just MLE) and Bayesian estimation. As expected, MLE is sensitive to random noise in input data whereas learning curve of the Bayesian estimation with a higher equivalent sample size α is considerably smoother. We can see that with a smaller network ("Cancer" or "Asia") the parameter estimation methods converge faster than with larger networks. On the MLE curves, especially in the case of "Cancer" and "Asia" networks, we can notice rapid instant

[2] I use 10^8 samples to approximate the probability distribution $P(Parents(X))$ when computing the KL-divergence. Such number of samples has been chosen because the KL-divergence graphs have been empirically proved to converge (for the ICU alarm network with 37 variables) when approximating probability distributions $P(Parents(X))$ from 10^7, 10^8 and 10^9 samples.

drops of the relative entropy. Cause of such a drop is a new sample that erases a zero entry from some CPD and thereby reduces the penalty imposed on the network learnt by MLE as discussed earlier in this section.



(a) “Cancer” network (5 variables, 10 parameters)

(b) “Asia” network (8 variables, 18 parameters)

(c) “Child” network (20 variables, 230 parameters)

(d) “ICU alarm” network (37 variables, 509 parameters)

Figure 5.1: Performance analysis of parameter estimation methods for networks and datasets of various sizes using KL-divergence.

Also, according to the tests performed, MLE seems to be steadily outperformed. Of course, if we had an excessive amount of training data, which is usually not the case in practice, then MLE would actually be more suitable than Bayesian estimation because the pseudosamples representing prior distribution for Bayesian estimation would turn into a source of inaccuracy.

If a concrete application demands estimated network parameters to be as close to the “real ones” as possible, similar analysis, as I just performed, is a sensible thing to do – pick

a network of similar complexity, plot the learning curves for this network and determine what is the best parameter estimation method with respect to the size of our dataset.

5.1.2 Learning structure

Evaluation of structure learning proved to be a little problematic. We cannot directly use KL-divergence because the network structures aren't identical (although [9] suggests a way of computing the KL-divergence) and other metrics for measuring distance of two probability distributions aren't suitable as they are based on exhaustive enumeration of atomic events and don't decompose over the network structure as KL-divergence does. Instead, I rely on manual inspection of the best-scoring network structures found over multiple runs of the learning algorithm and comparing these to the original structure. This approach is perfectly legal as I am later going to demonstrate usage of Bayesian networks in the field of data-mining which is a largely human-assisted discipline.

Because of I-equivalence there usually are multiple network structures sharing the same highest score and being, in a sense, equal. So, as a result of structure learning we might get a whole set of I-equivalent networks which differ only by direction of some edges. For such edges there is no clear causality between variables they connect, at least none that could be derived mathematically based on the dataset at hand, however, an expert may be able to decide the causality quite easily. So, not only we cannot tell whether any of the I-equivalent structures is better in terms of causality or in terms of explaining the data at hand, the learning algorithm driven by maximizing a scoring function cannot tell either. As we don't know the "true" network structure, we need to inspect all the best-scoring structures. To ease the manual evaluation of structure learning I also maintain statistics for each pair of variables U, V how often the final learnt structure contains the edge $U \to V$. This statistics is presented by a table in GUI.

As it turned out during my experiments, in context of Bayesian score the property whether or not two I-equivalent structures have the same score depends on the parameter prior we use. The simple K2 prior doesn't satisfy this property, the BDe prior does [12, p. 807]. It is unfortunate because the K2 prior is easy to specify whereas a general BDe prior needs to be specified by creating a prior network (both network structure and its parameters). As a compromise I came up with a simple solution – I use BDe prior as if it were represented by a BN and as if the hyperparameters were inferred as $\alpha_{x|\mathbf{p}} = \alpha \cdot P'(x, \mathbf{p})$ where P' is distribution induced by our prior BN. The trick is to make the prior BN discrete (without any edges) and to make prior distribution over each variable in the prior BN uniform. This way we get $\alpha_{x|\mathbf{p}} = \alpha / \big|val\big(\{X\} \cup Parents(X)\big)\big|$ quite elegantly without the need to perform any real inference or to even keep inner representation of the prior BN. Thereby the BDe prior is both easy to specify (only the value of α is needed) as well as it satisfies the property of I-equivalent structures having the same score.

In my experiments I have found two I-equivalent structures to occur frequently. The first kind of structure a call a *chain*. A chain $X_1 - \ldots - X_n$ is a structure in which variables $X_1, \ldots, X_n$ are connected in a linear fashion in specified order $1, \ldots, n$ with no V-structure and with no incoming edges to any of the variables; outcoming edges are unrestricted. At this point I introduce my own graphical notation for chains in Figure 5.2 (not to be confused with graphical notation of *plate models* [12]). Another set of I-equivalent structures, that also arises, I call *central variable structure*, meaning there is a "central" variable X and a set of variables $Y_1, \ldots, Y_n$, all directly connected to X. At most one Y variable may be parent of X while all the other Y variables are children of X. There are no other edges among

the variables and also no incoming edges to the whole structure. All such central node structures are I-equivalent. Please see graphical notation of the central variable structure in Figure 5.3.

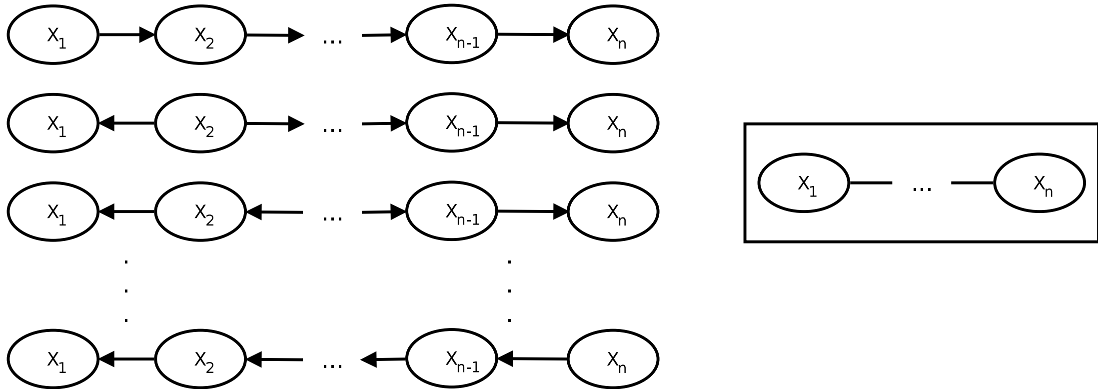


Figure 5.2: A general chain $X_1 - X_2 - \ldots - X_n$ of variables $X_1, \ldots, X_n$ is a set of network structures where the variables are connected in a linear fashion in the order $1, \ldots, n$ and there is no V-structure. Left side of this figure demonstrates the whole set of n structures for the same chain and on the right side of this figure you can see my own graphical notation for the whole set of chains (linear structure with undirected edges enclosed in a box).

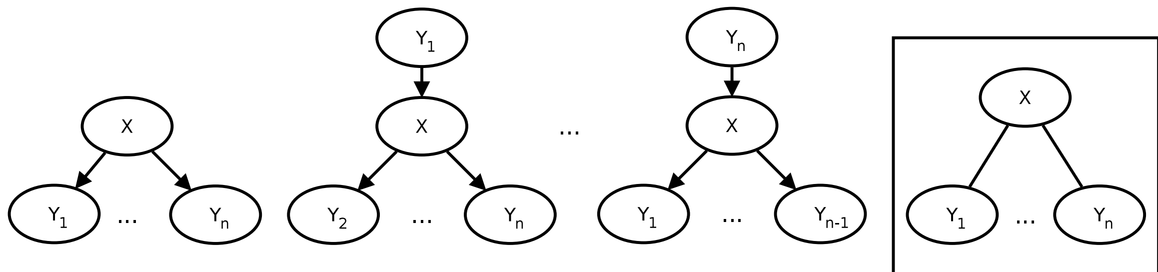


Figure 5.3: Central variable structure $X, Y_1, \ldots, Y_n$ is a set of I-equivalent structures where the X variable is central in the sense that is connected with all Y_i variables, at most one Y_i variable is parent of X and all the other Y_i variables are children of X. There also may not be an incoming edge to any of the variables X or Y_i as that would potentially introduce a V-structure in some of the networks. Structures without a bounding box represent the whole set of $n+1$ central variable structures and on the right side of this figure, enclosed in a box, you can see my own graphical notation for the whole set of central variable structures.

Results of the following experiments have been obtained primarily using the BIC score. The Bayesian score leads often to the same best-scoring structures and differences will be pointed out explicitly. The datasets I have used are available on the enclosed CD at the location `1-benchmarks/datasets/`.

Before we proceed to the experiments with structure learning, please note that outcome of the experiments relies on the concrete dataset we sample before learning. So, for two datasets of equal size we quite frequently obtain different sets of best-scoring structures, especially when the number of samples is small (hundreds or thousands of samples, depending on the network).

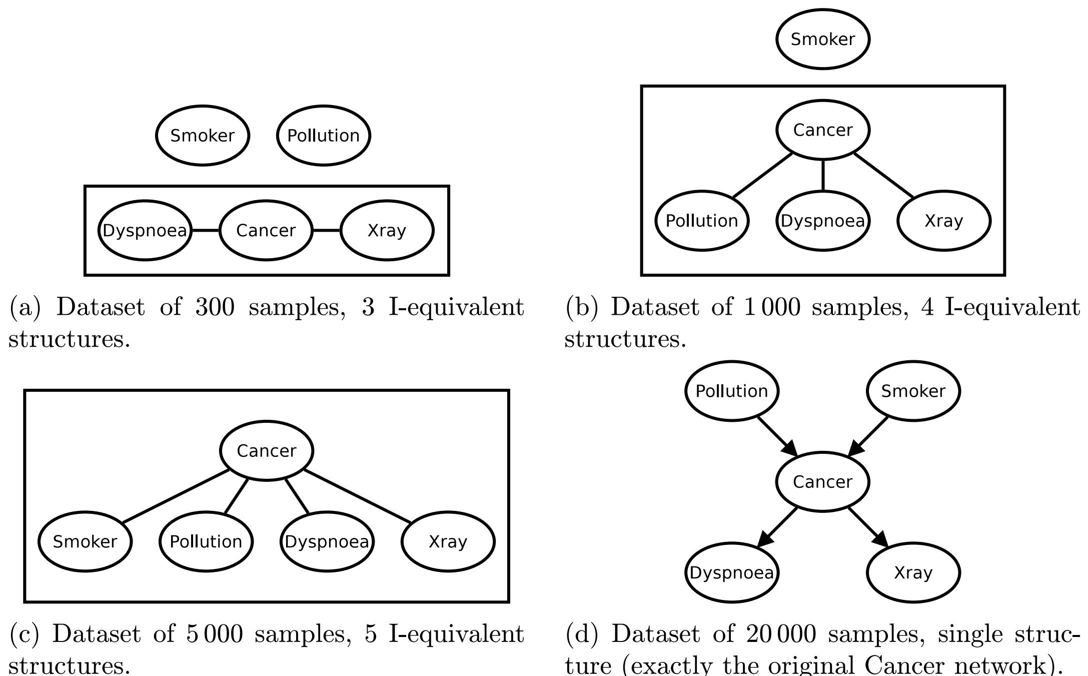
Experiments with the Cancer network

The Cancer network with five variables is the simplest network I consider in my experiments. With a dataset of 300 samples the structure learning with BIC score discovers only a chain $Xray - Cancer - Dyspnoea$, respectively its three I-equivalent variants, the variables $Smoker$ and $Pollution$ are isolated (see Figure 5.4a).

With a dataset of 1 000 samples the structure learning algorithm discovers only dependency among four variables in form of a central variable structure (see Figure 5.4b). The choice of central variable to be the $Cancer$ variable is consistent with the original network. The variable $Smoker$ is isolated in the found structures as $P(Cancer \mid Smoker)$ differs only by the value 0.02 at maximum for different assignments of $Smoker$ and the potential edge $Smoker \rightarrow Cancer$ is in the score more penalized by the increase of structure complexity than it is favored for its contribution to the sum of family scores.

For a dataset with 5 000 samples and the BIC score we finally obtain a connected graph, only there is similar situation as we had with 1000 samples – we obtain a central variable structure with $Cancer$ being the central variable (see Figure 5.4c). The results of this test case and of the previous one suggest that V-structures need to be strongly implied by the dataset, otherwise the learning algorithm avoids introducing a V-structure.

Finally, with a dataset of 20 000 samples the learning process with the BIC score yields the exact original structure of cancer network and no others (see Figure 5.4d). The original structure is unique as it contains a V-structure in $Cancer$ variable which ruins any possibility of there being a chain of variables that could be somehow rearranged. With the Bayesian score and $\alpha = 0.1$ we still obtain the central variable structure as in Figure 5.4c even with a dataset this big, the exact original structure is found with a dataset of 30 000 samples. With $\alpha \in [1, 10]$ the exact structure is found also with 20 000 samples.



(a) Dataset of 300 samples, 3 I-equivalent structures.

(b) Dataset of 1 000 samples, 4 I-equivalent structures.

(c) Dataset of 5 000 samples, 5 I-equivalent structures.

(d) Dataset of 20 000 samples, single structure (exactly the original Cancer network).

Figure 5.4: Best-scoring I-equivalent network structures found for the Cancer network with datasets of various sizes using the BIC score.

Parameters of the experiments: 200 runs, 200 iterations per run, 15 random restart steps, 0.15 relative size of tabu-list, maximum of 3 parents, $\alpha \in [0.1, 10]$.

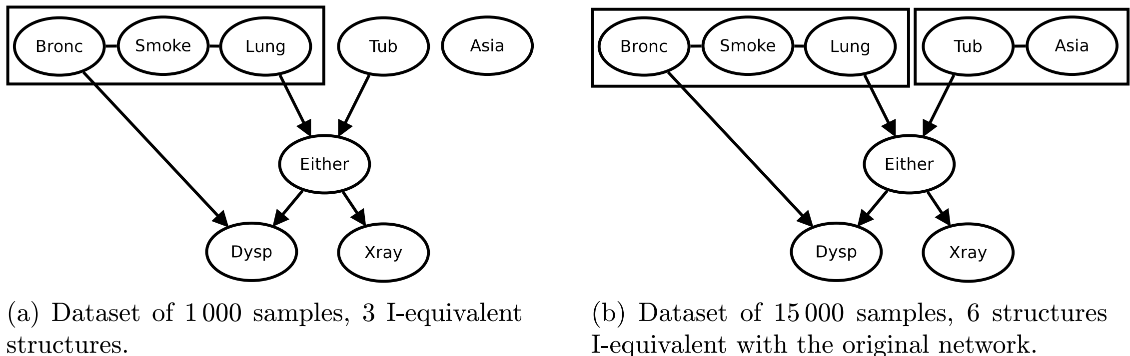
Experiments with the Asia network

The Asia network contains 8 variables and two V-structures $Bronc \rightarrow Dysp \leftarrow Either$ and $Lung \rightarrow Either \leftarrow Tub$. The CPD for *Dysp* variable suggests strong dependency on both its parents – the distribution $P(Dysp \mid Either, Bronc)$ changes by up to 0.7 for different assignments of *Either* and by up to 0.6 for different assignments of *Bronc*. Therefore for as little as 100 samples the found networks contain the V-structure at the *Dysp* variable. Furthermore, *Dysp* has always the same set of parent variables but that can be accounted to the strong dependency in the original network.

With a dataset of 1 000 samples we almost get structures I-equivalent to the original network (see Figure 5.5a), only the variable *Asia* is isolated thanks to a weak dependency $Asia - Tub$. There are three I-equivalent structures because variables $Smoke - Bronc - Lung$ form a chain. Topology of the other variables is the same as in the original network.

For a dataset of 15 000 samples we obtain correct structure of the Asia network, respectively its six I-equivalent variants (see Figure 5.5b) thanks to two chains $Asia - Tub$ and $Lung - Smoke - Bronc$ which are present in the original network.

In the case of the Asia network the Bayesian score worked best with $\alpha = 1$. With higher values of α (eg. 5 or 10) the learnt networks contained additional edges and V-structures that aren't present in the original network.



(a) Dataset of 1 000 samples, 3 I-equivalent structures.

(b) Dataset of 15 000 samples, 6 structures I-equivalent with the original network.

Figure 5.5: Best-scoring I-equivalent network structures found for the Asia network with datasets of various sizes (same results for the BIC score and the Bayesian score).

Parameters of the experiments: 200 runs, 200 iterations per run, 15 random restart steps, 0.15 relative size of tabu-list, maximum of 3 parents, $\alpha = 1$.

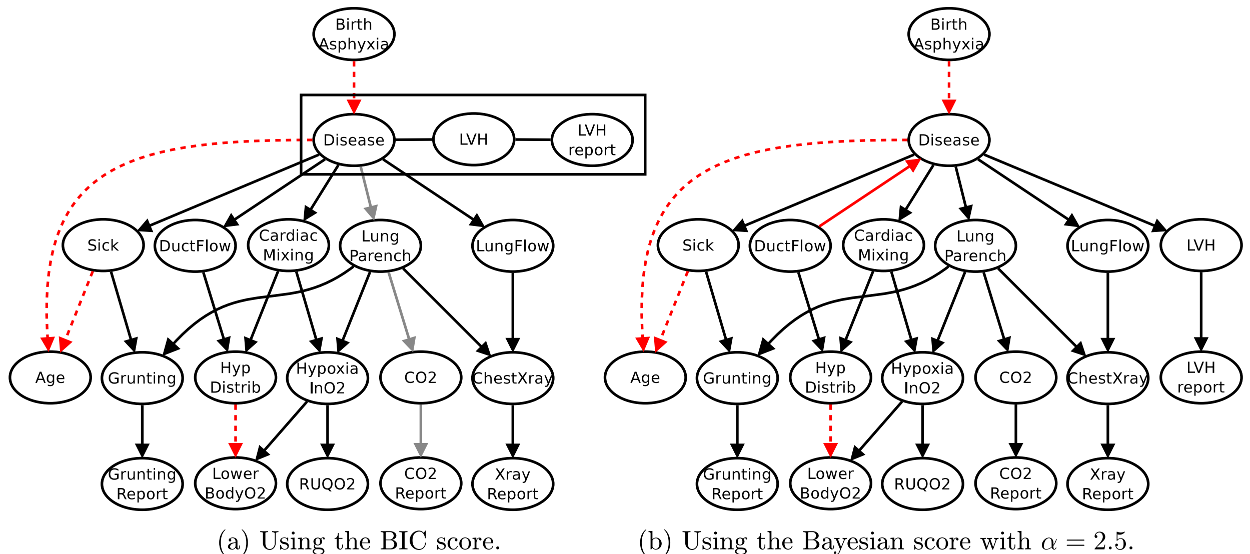
Experiments with the Child network

The Child network is a bit larger, contains 20 variables, 6 V-structures and a variable with a high number of children (the *Disease* variable with 7 children). At first we will inspect the best structures found with datasets of different sizes (best in the sense of minimal differences from the original network) and explore what inconsistencies they have. Then we will discuss how to proceed when attempting to learn the best structure possible when the original structure is unknown.

With a dataset of 500 samples and with the BIC score structure learning yields a structure with two isolated variables (*Age* and *BirthAsphyxia*), the rest of the network is

to an almost surprisingly high degree consistent with the original network, considering how little samples we have (see Figure 5.6a). The best-scoring networks contain four V-structures out of the six which are present in the original network, again suggesting that once a V-structure is learnt it is a fairly stable feature of the network unlike a chain or a central variable structure. When not considering edge directions, the learnt network is missing 4 out of 25 connections present in the original network and from the remaining 21 edges up to three may be reversed due to chain structures. There is one obvious chain structure $Disease - LVH - LVHreport$ causing three I-equivalent variants of the learnt network. Notice that when the edge $Disease - LungParench$ is reversed then $LungParench - CO2 - CO2Report$ effectively becomes another chain structure because then there are no incoming edges to any of these variables (edges that might be reversed are painted in gray). In either case, only those variants of the chain structures are allowed such that they don't introduce a V-structure in the $Disease$ variable as this would imply conditional dependencies and marginal independencies that aren't supported by the data. Furthermore, in the learnt network there isn't any redundant connection that wouldn't be in the original network.

With the Bayesian score and 500 samples the best-scoring network is similar to that of the BIC score (achieved using $\alpha = 2.5$, see Figure 5.6b). The only difference is that the edge $Disease - DuctFlow$ is reversed instead of allowing for variations of the chain structure $Disease - LVH - LVHreport$. I believe this difference may be caused by a local maximum in the Bayesian scoring function which makes it for the optimization algorithm difficult to find the I-equivalent structures which the BIC score does find.



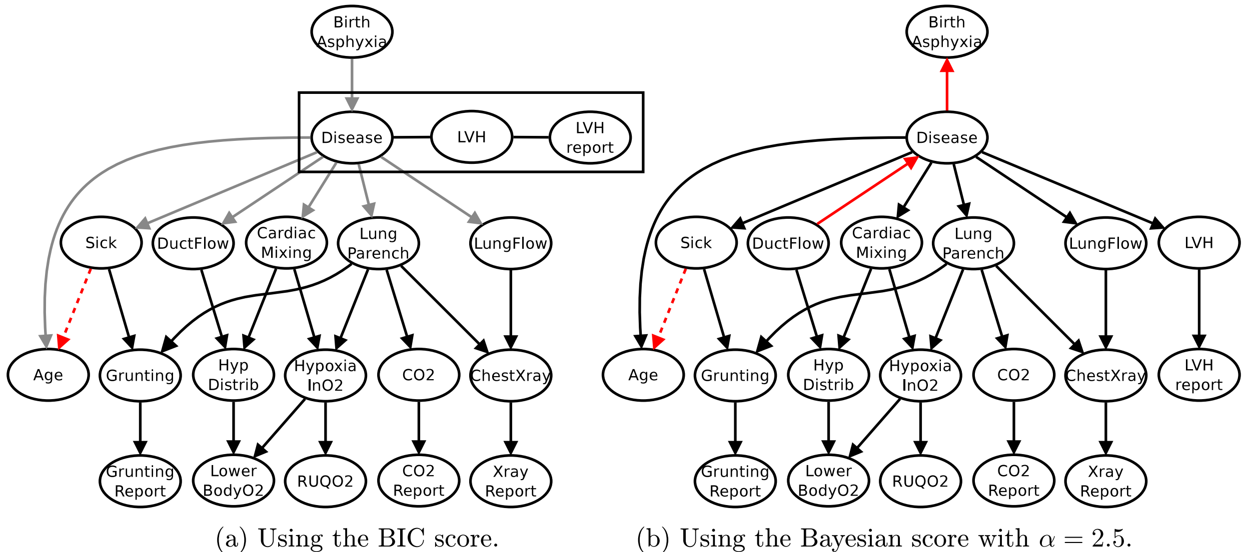
(a) Using the BIC score. (b) Using the Bayesian score with $\alpha = 2.5$.

Figure 5.6: Best-scoring network structure found for the Child network with a dataset of 500 samples.
Legend: Red dotted edges are missing, red edges are reversed, gray edges may be reversed (under conditions stated in the text).

With a dataset of 2 000 samples and the BIC score we get a connected graph in which only the edge $Sick - Age$ is missing and one or two of many edges may be reversed (see Figure 5.7a). There are many chain structures involving the $Disease$ variable (one with three variables, expressed by the boxed graphical notation, and seven chain structures with

two variables, expressed with a gray edge instead). Legal variants of the chain structures are only those that don't introduce a V-structure in the *Disease* variable, eg. if *Age* were parent of *Disease* then no other variable of the other chain structures may be a parent of *Disease*. Furthermore, if the edge $Disease - LungParench$ is reversed then $LungParench - CO2 - CO2Report$ effectively become a chain because then there is no incoming edge to any of those variables.

With the Bayesian score (best results achieved with $\alpha = 2.5$, see Figure 5.7b) and 2 000 samples the learnt structure has no isolated variable, it is missing the edge $Sick - Age$, just as with the BIC score, and only two edges are reversed which is a particularly nice result.



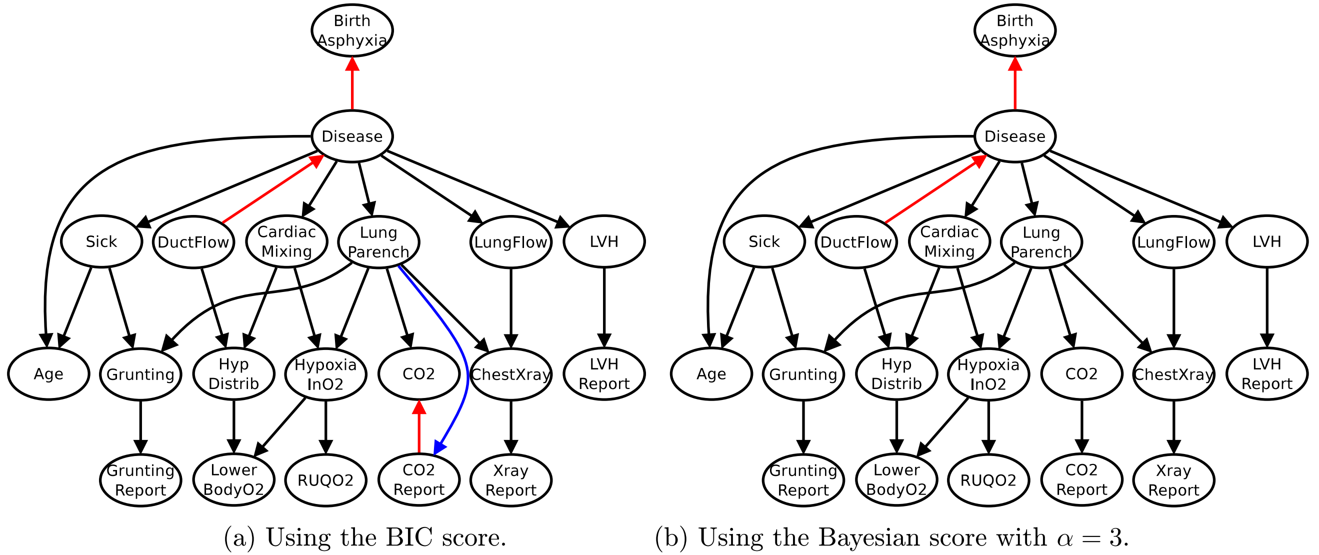
(a) Using the BIC score. (b) Using the Bayesian score with $\alpha = 2.5$.

Figure 5.7: Best-scoring network structure found for the Child network with a dataset of 2 000 samples.
Legend: Red dotted edges are missing, red edges are reversed. One of the gray edges may be reversed if it doesn't introduce a V-structure in the *Disease* variable wrt. concrete variants of the other chain structures in the *Disease* variable.

Dataset of 5 000 samples and the BIC score get us as close to the original network as no missing connections, three reversed edges and one redundant edge. The mistake introduced by the redundant edge is somewhat local, involving a closed structure of three nodes *LungParench*, *CO*2 and *CO*2*Report* (see Figure 5.8a) and no other variables are involved. Curious thing is that there has actually been formed a V-structure in the *CO*2 variable which might be caused by noise in the used dataset.

The Bayesian score (best with $\alpha = 3$, see Figure 5.8b) in this particular test did significantly better than the BIC did. They both agree on the reversed edge $Disease - BithAsphyxia$ and, other than that, with the Bayesian score there is only one reversed edge and no redundant edges.

In a situation when our goal is to discover the original structure the question is how to pick the right equivalent sample size α for the Bayesian score so that the learnt structure would be as close as possible to the correct one. For a range of α values I have inspected the relation between Bayesian score of a learnt structure and how closely the learnt structure matches the original one. I have found out that within the approximate range $\alpha \in [1, 4]$ structure with the highest score is also the closest. I have made the observation that higher



(a) Using the BIC score. (b) Using the Bayesian score with $\alpha = 3$.

Figure 5.8: Best-scoring network structure found for the Child network with a dataset of 5 000 samples.
Legend: Red dotted edges are missing, red edges are reversed, blue edges are redundant.

value of α means higher chance of the learnt network having extra edges and also doesn't improve the number of reversed edges. So, α should be kept at a small value. It is true that with higher values of α we can discover a structure with even better Bayesian score but, in my experience, this structure typically has some redundant edges that aren't present in the original network and also the number of other deviations isn't pleasing. So, when attempting to learn the best possible structure (and the number of variables is comparable with the child network) we should try the learning procedure for $\alpha \in [1, 4]$ with a step 0.5 and pick the value of α so that the Bayesian score of the learnt structures is the highest.

Parameters of the learning procedure: 1 000 runs, 600 iterations per run, 50 random restart steps, 0.15 relative size of tabu-list, maximum of 4 parents, $\alpha \in [1.5, 3]$.

Conclusion of the the experiments

Based on the experiments regarding structure we can make some general observations:

- There are often more than one best-scoring structures due to I-equivalence and, with some practice, chain structures and central variable structures are actually relatively easy to spot. Other than these two types of structures we sometimes observe swapping two variables B, C in a cascade of the form $A \rightarrow B \rightarrow C$ while preserving edge $A \rightarrow B$ and introducing an edge $A \rightarrow C$. The experiments suggest that such structural mistakes are just local and don't involve other variables.

- V-structures introduce conditional dependencies and marginal independencies, therefore a V-structure is not learnt unless it is strongly implied by the data and incorrect V-structures are rare (for reasonable values of the parameter α for the Bayesian score).

- Best value of α for the Bayesian score is usually relatively small – for the used networks and datasets the best values of α were from the interval $[1, 3]$ in all cases. Also, for $\alpha \in [1, 3]$ learnt structures with higher Bayesian score were generally closer to the

original network. This provides us with a powerful rule of thumb on how to determine the right value of α when learning a network structure whose original structure is unknown to us – look for a local maximum among the scores of networks learnt using α within the range about $[1, 4]$.

- The Bayesian score is more conservative, the BIC score often leads to a specific structure close to the original one faster (with a smaller dataset) whereas the Bayesian score requires a larger dataset. Although with a tuned value of α, results with the Bayesian score are usually the same or even better than those of the BIC score.

5.2 Crime, its causes and countermeasures

This application will take advantage of the fact that Bayesian networks are intuitive visual models that capture dependencies among random variables. With these ideas in mind BNs can be used to visualize the structure of a given domain in order to better understand the underlaying dynamics and relations within that domain.

In our particular problem we are given a dataset called *Communities and Crime*[3] containing records that characterize various communities across the United States. A single record consists of features such as racial representation, income, family completeness attributes, housing etc. and the relative number of violent crimes in this community. Our goal is to find a BN that describes given data the best and that will hopefully provide us with some useful insight. Understanding dependencies among features can help us identify the main causes of crime, their relations and, based on these findings, to propose justified precautions.

5.2.1 Data preprocessing

Because the used dataset contains continuous features and because this thesis considers strictly discrete Bayesian networks we need to perform some kind of discretization. I have applied binning into three equi-depth bins labeled *low*, *medium* and *high*. Such discretization seems reasonable with respect to the relatively small dataset and because the new values are, for a human, easily interpretable. Even with as little as three discrete values, four attributes $LemasPctOfficDrugUn$, $NumStreet$, $MedNumBR$, *pctUrban*[4] are after the discretization uneven, ie. one of the bins is almost or entirely empty. So, we will keep in mind to primarily discard those attributes if some of them is member of a set of similar variables (will be explained later).

There is also a subset of attributes whose value is roughly in 50 % of the records unknown. All of these are attributes of the police force (eg. number of policemen, police cars, requests, people working in narcotics etc.) and, from their nature, they don't really describe a community, instead they are directly affected by the amount of criminality. However, we are interested in causes of criminality, not in the results among which surely is the overall police budget. Furthermore, filling in such a big number of missing values, eg. based on some prediction model, doesn't seem reasonable. Therefore we discard those attributes altogether which leaves us with 100 attributes.

[3] The dataset *Communities and crime* is available at `http://archive.ics.uci.edu/ml/datasets/Communities+and+Crime`.

[4] I use original names of attributes as they are in the Communities and crime dataset. Please see `.names` file on the enclosed CD for detailed description of all attributes.

5.2.2 Experiments

I have tried to approach this data-mining task in several ways because the achieved results weren't as good as I had hoped. When using the complete dataset there were too many variables to see anything useful in the learnt network structure and when I attempted to somehow reduce the feature set, effect of many key variables on the target variable $ViolentCrimesPerPop$ (further referred to just as $Crime$) changed radically. Nevertheless, I believe that at least some useful information can be read from the models I have created.

To briefly elaborate, I have experimented with the following three approaches:

a) Start with a collection of networks learnt using the Bayesian score with $\alpha_S \in [1, 15]$ and using the full feature set. Then, iteratively identify sets of variables that are in all the structures, that have been learnt with different α_S, topologically close, have similar effect on the target variable $Crime$ (stimulating or suppressing) and their semantics is similar (eg. divorce rate of males vs. divorce rate of females or percentage of employed people vs. percentage of unemployed). Each such set of similar or antagonistic variables has been replaced by just one representative variable. This way I have managed to reduce the network from 100 variables to 67 variables within three iterations.

   To be more formal, variable X and its representative R influence the crime variable C approximately the same way when $P(C \mid X) \approx P(C \mid R)$ or the opposite way when $P(C \mid X = low) \approx P(C \mid R = high)$, $P(C \mid X = medium) \approx P(C \mid R = medium)$ and $P(C \mid X = high) \approx P(C \mid R = low)$. These probability distributions are inferred by sampling after the network parameters are learnt using Bayesian estimation with $\alpha_P = 30$. The value of α_P has been chosen based on the KL-divergence analysis of parameter learning with the Hepar-II network – this network is the most similar I could find to the learnt networks and also gives us a good breathing space while the learnt networks get simpler through gradual elimination of attributes. In Figure 5.9 we can see that for the Hepar-II network and 2 000 samples, which is the size of our dataset, the value $\alpha_P = 30$ gets us closest to the original probability distribution.

   If it is the case that $P(C \mid X) \approx P(C \mid R)$ then mutual information of X and R has to be relatively high. To make the manual search of similar variables easier, mutual information of a pair of directly connected variables X, R is visually represented in the program by color of the edge $X - R$ on "cold-to-hot" color scale, ie. blue to red[5].

b) Start when the case (a) ended. We cannot further apply the iterative elimination of similar attributes and 67 still is a large number of attributes. At this point, using the network structures learnt with the 67 attributes left, I analyzed what attributes influence the target variable $Crime$ the most, dropped those attributes with least influence and finally, learnt new network structures based on the surviving 52 attributes.

   To express the degree of influence some variable X has on the target variable $Crime$ I determine how significantly the target variable deviates from its prior distribution, which is the uniform distribution $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ due to binning, when X is given. Ie. for variable X find the maximal value of the expression $\left|1/3 - P(Crime \mid X = x)\right|$ over all the networks learnt with various values of α_S and over all $x \in val(X)$. I will

[5] In order to display mutual information between pairs of directly connected variables the current dataset has to be compatible with the current network and the option `Network -> Display edge weights` in the main menu needs to be checked.

further refer to this particular value as the *crime impact factor* of variable X. The probability distributions needed in order to determine the crime impact factor are, again, inferred by weighted sampling while the network parameters have been learnt using the Bayesian estimation with $\alpha_P = 30$.

c) Initially use the collection of networks learnt from the full feature set with various values of α_S, as in the (a) case. Then, right away analyze what attributes X influence the target variable the most (using the crime impact factor introduced in case (b)), pick the top 40 most influential attributes and finally learn new network structures based on just the top 40 attributes.

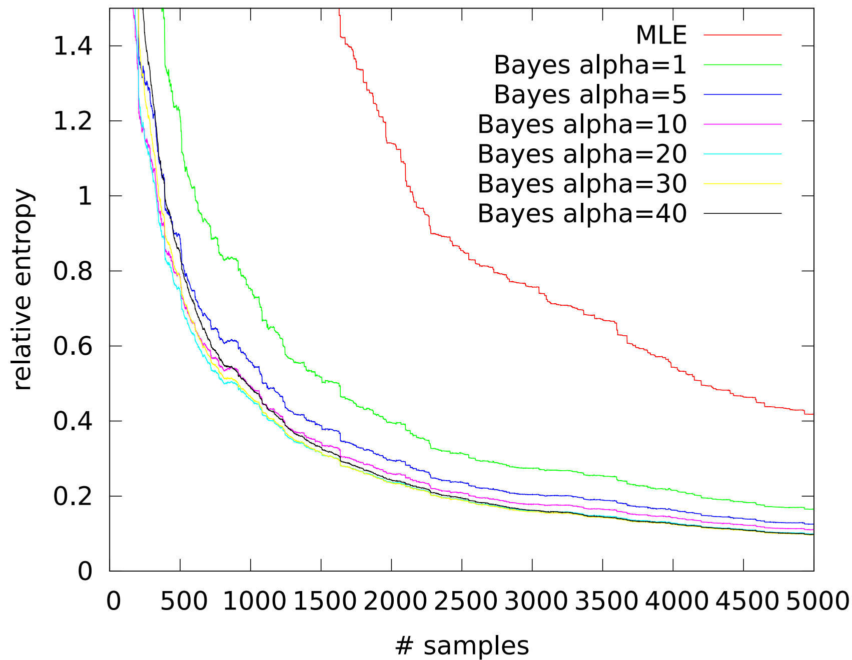


Figure 5.9: Performance analysis of parameter estimation methods for the Hepar-II network (70 variables, 1453 CPD entries). For a dataset of 2 000 samples we can see that the Bayesian estimation with $\alpha = 30$ is the best option.

Because we don't know the right parameter α_S of the Bayesian score for networks of this size and because the learnt networks differ in some edges, for each set of final networks we determine their structural intersection. In this context, by intersection I mean identifying those directed edges that are present in majority of the networks while tolerating a missing edge at most once (with 0 tolerance the networks contain too many isolated variables and with tolerance higher than one the networks become too dense). For the intersection network we compute crime impact factors of all variables, compare with crime impact factors in the original network and highlight variables that preserve the crime impact factor (green color) and variable that have become totally misleading (red color). This way we can easily spot clusters of variables that are still relevant and ignore those that aren't. The structural intersection is computed from seven networks learnt using $\alpha_S \in \{1, 2, 5, 7.5, 10, 12.5, 15\}$.

Now is the time to inspect the final networks for the cases (a), (b) and (c). In the first case we have in three iterations eliminated attributes that were, to some degree, duplicate

and we have obtained the final network with 67 nodes shown in Figure C.2. This network is still very large and, in the printed version, names of some of the variable might not even be readable[6]. It is hard, thanks to its structure, to read some useful information out of the learnt network. The target variable is significantly dependent only on the attributes capturing family completeness, illegal immigrants, racial representation and housing (variables whose crime impact factor is correct are highlighted in green, other than green variables are not considered because the results of analysis would be questionable). How exactly those variables affect the target variable $Crime$ can be easily determined by probabilistic queries which converge very fast. Generally, high divorce rate means high criminality. It can be seen that the illegal immigrants variable $PctIlleg$ and the $racepctblack$ variable are particularly negatively correlated with $PctFam2Par$. The key variable $PctPersDenseHous$ (suggesting ghettos or questionable city quarters) is positively influenced by the $PctIlleg$ variable and not so strongly influenced by the $racepctblack$ variable which is a little surprising. Still, based on the model we can conclude that illegal immigrants tend to live in relatively big numbers in homeless shelters.

The (b) case takes the variables that have survived filtering in the (a) case, determines crime impact factors of these variables and eliminates those with crime impact factor less than 0.07 which leads to a network with 53 nodes (the threshold 0.07 seemed like a reasonable compromise between not dropping influential variables and reducing the feature set to less than 67 variables). Before we get into analyzing the network itself it is important to say that among other variables, that were eliminated because of low crime impact factor, was the $racePctAsian$ variable; also, this variable has low crime impact factor in the networks learnt using the full feature set as well. This fact suggests that the percentage of Asians is not very discriminative in terms of the target variable $Crime$, ie. that Asians in general don't significantly contribute to criminality, not in the positive nor in the negative sense. Now we get to analyzing the learnt network shown in Figure C.3. The green variables, whose crime impact factors are correct, can be divided into two groups. The group of variables on the left from $Crime$ describes negative factors like illegal immigrants, dense housing, number of black people and people living on the street; most influential are the variables $PctIlleg$, $racepctblack$ and $PctPersDenseHous$. There is also the variable reflecting the number of white people which is in very negative relation with all the other variables including $Crime$ (in other words, when speaking about criminality, white people are generally well-behaved). The group of variables on the right from $Crime$ captures family completeness and wealth ($PctHousLess3BR$, $pctWWage$). The family completeness variables influence the $Crime$ variable heavily, the wealth variables aren't so powerful but there is still noticeable tendency of increased criminality with lower wealth.

In the (c) case we have analyzed crime impact factors of all variables right in the networks learnt using all 100 features and selected top 40 attributes with the highest crime impact factor to obtain the network shown in Figure C.1. This network is probably the nicest of all we have learnt because it contains a single cluster of variables whose crime impact factor is correct and those variables are relatively densely connected, so we can expect that many of the real dependencies are actually reflected in the network. As opposed to the other two networks, in this network most of the family completeness variables, except for $PctKids2Par$, have incorrect crime impact factors. We can observe four smaller clusters of nodes describing mutually similar traits: (1) the green cluster right above $Crime$ describing illegal immigrants, black and white people, people living on the streets or in shelters (this

[6]If it is the case, please see vector images of the learnt networks in electronic version of this thesis or on the enclosed CD in the `2-crime/final networks` directory.

cluster has been present in the previous two networks as well); (2) the green cluster forming a vertical belt on the left from $Crime$ capturing erudition, job quality and wealth; (3) red cluster below $Crime$ with family completeness and wealth variables; (4) red cluster on the far left with additional wealth and poverty variables. From cluster (1) the most influential variables are in descending order $PctKids2Par$, $PctIlleg$, $racePctWhite$, $racepctblack$. The $NumInShelters$ variable is influenced mostly by the $NumIlleg$ variable meaning that illegal immigrants often end up in homeless shelters where they live at the expense of productive people and contribute to higher criminality. The variable $PctPersDenseHous$ is strongly influenced by illegal immigrants and, to lesser extent, also by $racepctblack$ which partly suggests the presence of ghettos. Variables from cluster (2) aren't by far as powerful in terms of their effect on $Crime$ as variables of the first cluster but there is still a steady influence (better education and jobs' quality suppress criminality).

Parameters of the learning procedure: 500 runs, 1 500 iterations per run, 50 random restart steps, 0.1 relative size of tabu-list, maximum of 4 parents, $\alpha \in [1, 15]$.

5.2.3 Conclusion

This particular application of Bayesian networks turned out difficult and I found myself repeatedly in a dead end. Nevertheless, I believe that I have managed to find networks that correctly describe at least some aspects of the data. Maybe the most convincing argument is that in the learnt networks can be identified clusters of variables that capture similar traits (eg. family completeness, erudition combined with types of jobs etc.) and furthermore the the same clusters are repeatedly present within different networks.

Conclusions supported by the learnt networks will be summarized in this paragraph. When speaking about racial representation, high percentage of white people greatly suppresses criminality, Asians don't appear to be extreme in any direction and high percentage of black people means also noticeably higher criminality. There cannot be said anything about Hispanics. The number of black people also heightens the dense housing variable, in this case, probably the number of ghettos and black city quarters. Logical proposition is to fight against forming of such closed communities and to integrate ethnic groups into the whole population. Illegal immigrants contribute strongly to criminality and also, according to the networks, often stay in homeless shelters where they live at the expense of productive society. It seems that a sensible thing to do is to enforce strict immigration policy and to let into the country only those that will presumably be good citizens. Family completeness has major impact on criminality and is a sign of healthy society. However, low divorce rate is about mentality rather than about forcing people to stay married, and it is hard, from my point of view, to offer any suggestion. The need of public assistance income, lack of education and quality of jobs also contribute to criminality but not as significantly as one might think. Again, in this case it is about mentality and introduces a grand challenge for sociology to "make" people think the right way without actually forcing them do it.

5.3 Spam filtering

Since the 1990s spam has become an unpleasant part of the Internet. The working mechanism of spam is to flood as many users as possible with unsolicited emails in the hope that a small percentage of these users will respond the way spammer wants them to. Spam, as a stand-alone class of malware, has many subtypes distinguished by their content and purpose, for example advertisement, virus spreading or phishing emails. The main reasons for addressing the problem of spam is that manual spam deletion requires non-trivial portion of time and it may also pose a serious threat to a careless or trustful user.

If not stated otherwise, all knowledge regarding data-mining in this section has been drawn from a university course based on [8].

5.3.1 Overview of document classification

The problem of spam detection falls into the area of document classification with two classes – spam and ham (term used for a legitimate email). General classification methods work with data representation in the form of feature vectors which typically contain polynomial or continuous values, each carrying information regarding presence of a specific predetermined word or term[7]. In context of document classification a complicated preprocessing pipeline needs to be employed to obtain relevant feature representation of a document – removal of irrelevant words using a stop list, stemming which reduces morphological variety, selection of the most relevant subset of terms and finally creating feature representation of the document. Because document preprocessing and feature extraction is not the subject of this thesis I choose to rely on a publicly accessible spam dataset whose emails have already been preprocessed and converted into vector feature representation. Clear advantage of using such dataset is the fact that obtained results are easily comparable with results presented in other papers. On the other hand, if someone benchmarks a new spam detector using a dataset containing raw emails then the reported classification accuracy is not necessarily that relevant for the sake of comparing two classification methods because quality of the result relies heavily on the preprocessing and feature extraction steps which are usually documented just briefly.

According to the papers I have studied, probabilistic approaches to spam filtering are very successful and, in fact, the Naïve Bayes model is among the most popular spam detectors even in commercial solutions [3]. Probabilistic techniques have been applied to the problem of spam detection in many forms: a simple Bayesian framework [11], raw Naïve Bayes model [16] (with impressive results) or Naïve Bayes with a SVD-like input preprocessing so that the strong independency among features holds true [14]. In context of Bayesian networks naturally arises the question whether the assumption of independency between feature variables given the class variable in Naïve Bayes model isn't too limiting and whether a properly trained Bayesian network can perform better. Goal which we will pursue in this section is simple – compare the best performance achievable with a Naïve Bayes model and with a more general Bayesian network.

[7] More elaborate forms of features don't consider words or terms but so called *concepts*. A concept can be viewed as a family of words with the same meaning which has the advantage of compensating for synonyms.

5.3.2 Dataset and data preprocessing

We will use a standard dataset called *Spambase*[8] which contains 4601 records, each having 57 features and a class label. Other than being used as a benchmark in spam-related papers, another advantage of Spambase is that the emails have already been preprocessed and converted into vector representation. For this dataset we will explore best performance achievable with a more general Bayesian network and compare it to the performance of Naïve Bayes. We will consider Bayesian networks structurally somewhat similar to Naïve Bayes in that regard that the class variable *spam* stands above all others, ie. cannot be a child; feature variables have no restrictions except for the maximal number of three parents to limit the search space of feasible network structures.

Because this thesis considers strictly discrete Bayesian networks and because the Spambase features are continuous we need to perform some kind of discretization. At first I experimented with partitioning values of each feature by binning into three bins of equal depth but it turned out that many features are distributed very unevenly and therefore the equal depth condition couldn't be met. So, instead of binning I am using k-means clustering with $k = 3$ and transform values of each attribute to the discrete set $\{low, medium, high\}$. As other techniques of spam detection work with a continuous feature representation it is to be expected that the discretization step will reduce accuracy of the classifiers I will use.

5.3.3 Experiments

As the Bayesian score for structure learning and the Bayesian estimation for parameter estimation are both parametrized, we need to undertake a number of time-costly experiments to find the optimal parameter setting. That is why I will initially use a simple holdout testing with 80:20 split (produced by stratified sampling) to get a broader view of the classification accuracy over space of reasonable parameter settings. Then, accuracy in the region of the best settings will be thoroughly validated once more using 5-fold cross-validation, each fold having the same ratio of spam to ham.

The metric used for evaluating classification capabilities of our models will be accuracy which is computed from the test set as follows:

$$accuracy = \frac{\text{\# of correctly classified samples}}{\text{\# of all samples}}$$

In context of spam filtering there is the fact that penalties for false positives and false negatives are asymmetric because placing a ham email to the spam folder (or even automatically deleting it) is considered very undesirable. Therefore metrics such as precision, recall or their combination is often used. To ensure higher precision, and thereby to reduce the amount of false positives, a threshold τ is employed. Semantics of the threshold τ is that an email is classified as ham unless our model predicts $P(Spam = true \mid email) \geq \tau$. A study [1] inspecting the impact of selection of τ on the overall performance suggests that τ merely regulates the trade-off between precision and recall and that accuracy can be used as a metric for evaluating quality of classifiers just as correctly. That is why I use accuracy as the main and only metric and set $\tau = 0.5$.

Crucial step in answering the question whether a general Bayesian network can beat the Naïve Bayes model is to explore the space of possible parameter settings of parameter

[8] Spambase is a freely available spam dataset accessible at the UCI Machine Learning Repository at http://archive.ics.uci.edu/ml/datasets/Spambase.

learning and of structure learning. Let α_S denote equivalent sample size used for structure learning using the Bayesian score and let α_P denote equivalent sample size used for parameter learning using the Bayesian estimation. I have manually inspected all combinations $(\alpha_S, \alpha_P) \in \{1, 2, 5, 7.5, 10, 12.5, 15, 20, 30\} \times \{0.5, 1, 1.5, 2, 3, 5, 7.5, 10, 20, 30, 40, 50, 60\}$; unparameterized methods (maximum likelihood estimation for parameters and the BIC score for structure) were also considered but outperformed in each case. Results of these experiments is shown in the Figure 5.10. As we can see, the best performance was achieved around the point $(\alpha_S, \alpha_P) = (10, 12.5)$, so this point and its close neighborhood will, together with the Naïve Bayes model, be subjects to thorough 5-fold cross-validation tests.

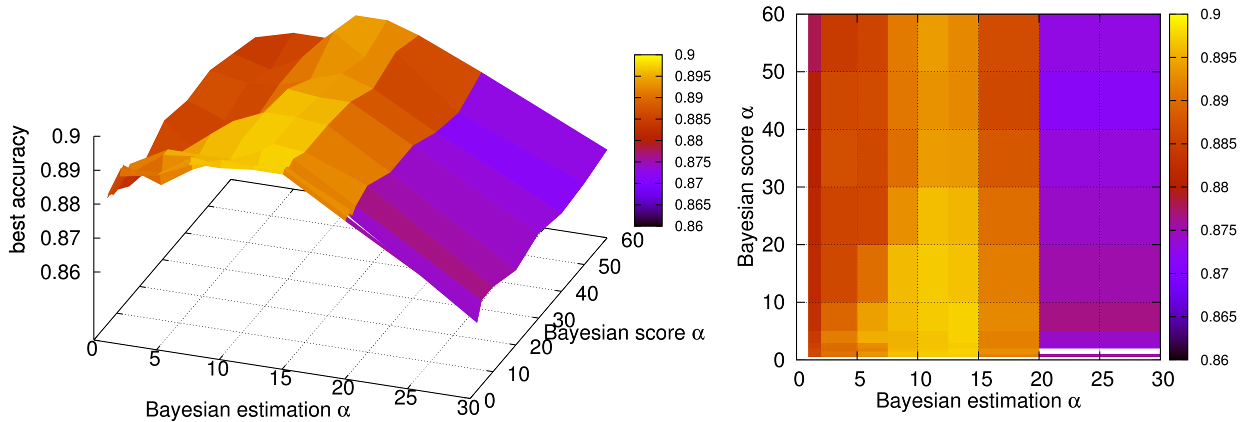


Figure 5.10: Accuracy of a Bayesian network model trained with different values of equivalent sample size α_S for structure learning (with the Bayesian score) and for different values of α_P for parameter estimation (with Bayesian estimation).

Results of the 5-fold cross-validation are the following. Naïve Bayes model (parameters estimated via MLE) has the average accuracy 85.3 % which is by 3.7 % less than was reported for the Spambase dataset and Naïve Bayes in [14] (reported 89 %). I believe this drop in accuracy is caused by discretization of the dataset and, in fact, it is almost surprising that the loss of variance isn't greater. Performance of the more general Bayesian network model has been inspected for parameter settings $(\alpha_S, \alpha_P) \in \{7.5, 10, 12.5, 15\} \times \{7.5, 10, 12.5, 15\}$. The best average accuracy from five-fold cross-validation has been 87.7 % with parameters $(\alpha_S, \alpha_P) \in \{(15, 12.5), (15, 15)\}$ which is by 2.4 % better than the accuracy of Naïve Bayes model. This result is comparable with the one achieved in [14] through augmenting the Naïve Bayes model with a SVD-like preprocessing (reported increase of accuracy by 4 %).

Considering that the Spambase dataset is relatively small in size and provides only 57 features, it is very likely that better performance can be achieved with a bigger dataset and with a richer and carefully chosen set of features (the learnt Bayesian networks contained isolated variables suggesting that some of the features aren't correlated with the spam variable nor with any other and therefore are effectively useless).

Other authors have also shown that performance of probabilistic classifiers can be greatly improved by introducing a broader set of features including, for example, sender's domain, reverse DNS lookups, presence of HTML code in the body (also considering coloring, fonts), punctuation, currency characters, run-length of capital letters etc. [16]

Parameters of the structure learning procedure: 200 runs, 3 000 iterations per run, 60 random restart steps, 0.15 relative size of tabu-list, maximum of three parents per variable, no incoming edges into the *spam* variable.

5.3.4 Conclusion

In this section I have demonstrated that a Bayesian network can perform better than a Naïve Bayes model in the field of spam detection even with a relatively small training set. Because both these models are very close in their nature and because Naïve Bayes model is so powerful that many real-world spam filters use it, I believe that with a large scale dataset a Bayesian network can achieve very high success rates. This is because the scoring functions for structure learning of Bayesian networks inherently make a bias-variance trade-off depending on the size of our dataset and because we can further relax structural restrictions to increase possible variance of the Bayesian network model (in my experiments I disallowed the *spam* variable to be a child and allowed no more than three parents per node). Both these properties of Bayesian networks suggest that their true power in the field of spam detection lies with large datasets having rich sets of features, whereas the Naïve Bayes model has been repeatedly proved to perform excellently with small datasets which are common in practice.

Chapter 6

Conclusion

This master's thesis explained the fundamentals of Bayesian networks, described various inference methods and methods of model learning. To get better hold of the theoretical concepts and to gain some insight I supplied commented formal proofs or derivations of multiple statements, some of the proofs were constructed by myself. All techniques have been studied and presented in such a detail that is fully sufficient in order to implement them and to understand their limitations.

The practical part of this thesis is divided into three areas. The first area is a study of effects of different parameter settings of presented model learning methods for datasets and networks of various sizes. This study provides the reader with some intuition and hints on how to proceed when learning a network whose correct structure and/or CPDs are unknown.

The second practical application explores the possibility of using Bayesian networks as a data-mining tool in order to discover internal structure of a studied domain. Concretely, three models for the Communities and crime dataset have been learnt and we have identified recurring clusters of tightly connected variables as well as overall impact of these variables on criminality. Based on these networks we were able to make propositions regarding suppression of criminality through justified interventions.

The third practical application examines the possibilities of using a Bayesian network in the field of spam detection and shows that a Bayesian network can outperform the very successful Naïve Bayes model, even with a relatively modest training set.

I believe that the most promising continuation of this thesis would be to thoroughly inspect the possibility of using Bayesian networks as spam detectors. In order to achieve better results a bigger dataset would be needed and extending the Bayesian network from discrete to continuous would also bring additional improvement.

Bibliography

[1] Ion Androutsopoulos, John Koutsias, Konstantinos Chandrinos, George Paliouras, and Constantine Spyropoulos. An evaluation of naive bayesian anti-spam filtering. In *Proceedings of the workshop on Machine Learning in the New Information Age*, 2000.

[2] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999. ISBN 0-13-301615-3.

[3] Enrico Blanzieri and Anton Bryl. A survey of learning-based techniques of email spam filtering. *Artificial Intelligence Review*, 29(1):63–92, March 2008.

[4] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, second edition, 2001. ISNB 0-07-013151-1.

[5] Karen Sachs et al. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308:523–529, 2005.

[6] Eric Freeman, Elisabeth Robson, Bert Bates, and Kathy Sierra. *Head First Design Patterns*. O'Reilly Media, 2009. ISBN 978-0-596-00712-6.

[7] David Halliday, Robert Resnick, and Jearl Walker. *Fyzika: Vysokoškolská učebnice obecné fyziky*. VUTIUM, 2000. ISBN 80-214-1869-9.

[8] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, second edition, 2006. ISBN 978-1-55860-901-3.

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

[10] Eric Horvitz, Johnson Apacible, Raman Sarin, and Lin Liao. Prediction, expectation, and surprise: Methods, designs, and study of a deployed traffic forecasting service. In *Conference on Uncertainty and Artificial Intelligence 2005*, pages 275–283, 2005.

[11] B. Issac, W. J. Jap, and J. H. Sutanto. Improved bayesian anti-spam filter implementation and analysis on independent spam corpuses. In *International Conference on Computer Engineering and Technology (ICCET '09)*, volume 2, pages 326–330, 2009.

[12] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009. ISBN 978-0-262-01319-2.

[13] P. Mirowski, H. Steck, P. Whiting, R. Palaniappan, M. MacDonald, and Tin Kam Ho. Kl-divergence kernel regression for non-gaussian fingerprint based localization. In *2011 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–10, 2011.

[14] D. Karthika Renuka, T. Hamsapriya, M. Raja Chakkaravarthi, and P. Lakshmi Surya. Spam classification based on supervised learning using machine learning techniques. In *2011 International Conference on Process Automation, Control and Computing (PACC)*, pages 1–7, 2011.

[15] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003. ISBN 0-13-080302-2.

[16] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail. Technical report, AAAI Workshop on Learning for Text Categorization, 1998.

[17] A. Zagorecki and M. J. Druzdzel. Knowledge engineering for bayesian networks: How common are noisy-max distributions in practice? *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(1):186–195, 2013.

Appendix A

Notation overview

This chapter presents listing of the mathematical notation used in this thesis:

X	Capital letter denotes a random variable.
x	Lowercase letter denotes a concrete instantiation of the variable X.
$\mathbf{X}$	Capital bold letter denotes a set of random variables.
$\mathbf{x}$	Lowercase bold letter denotes an instantiation of all variables in the set $\mathbf{X}$.
$val(\mathbf{X})$	Set of all possible instantiations of variables $\mathbf{X}$.
$Parents(X)$	Set of parent variables of variable X in a Bayesian network.
$parents(X)$	Instantiation of parent variables of variable X in a Bayesian network.
$Children(X)$	Set of child variables of variable X in a Bayesian network.
$children(X)$	Instantiation of child variables of variable X in a Bayesian network.
N_x	Number of samples of a dataset for which variable X has the value x.
$N_{x,\mathbf{pa}}$	Number of samples of a dataset for which X has the value x and variables $Parents(X)$ have the value $\mathbf{pa}$.
$\sum_{\mathbf{X}}(\dots)$	Summation over instantiations $\mathbf{x}$ of the variables $\mathbf{X}$.
$P(\mathbf{X})$	Probability distribution over variables $\mathbf{X}$.
$P(\mathbf{x})$	Probability of variables $\mathbf{X}$ having the concrete instantiation $\mathbf{x}$.

Appendix B

CD Content

Content of the enclosed CD is organized into the following directories:

- `1-benchmarks/`: Network `.net` files, datasets and measurements related to the first practical application.
- `2-crime/`: Vector images of the final three networks that resulted from the analysis of criminality. Also contains the original dataset, its discretized version, networks learnt during the feature elimination process and records of analysis of the crime impact factor.
- `3-spam/`: Datasets (the original dataset, its discretized version, train and test sets for holdout testing and for 5-fold cross-validation). Also contains measurement records and figures.
- `program/`: Program source codes compilable using the *ant* build tool and also runnable compiled version in `program/bin-precompiled`.
- `text/`: Electronic version of this thesis and LaTeX sources, including all figures.

Appendix C

Crime analysis – final networks

This chapter contains the three networks that are result of the crime analysis performed in Section 5.2. Because of their size, each network is on its own A3 page. If you have trouble reading variable names in the printed version please see the electronic version of this thesis or the original figures on the encolsed CD within the directory `2-crime/final networks/`.

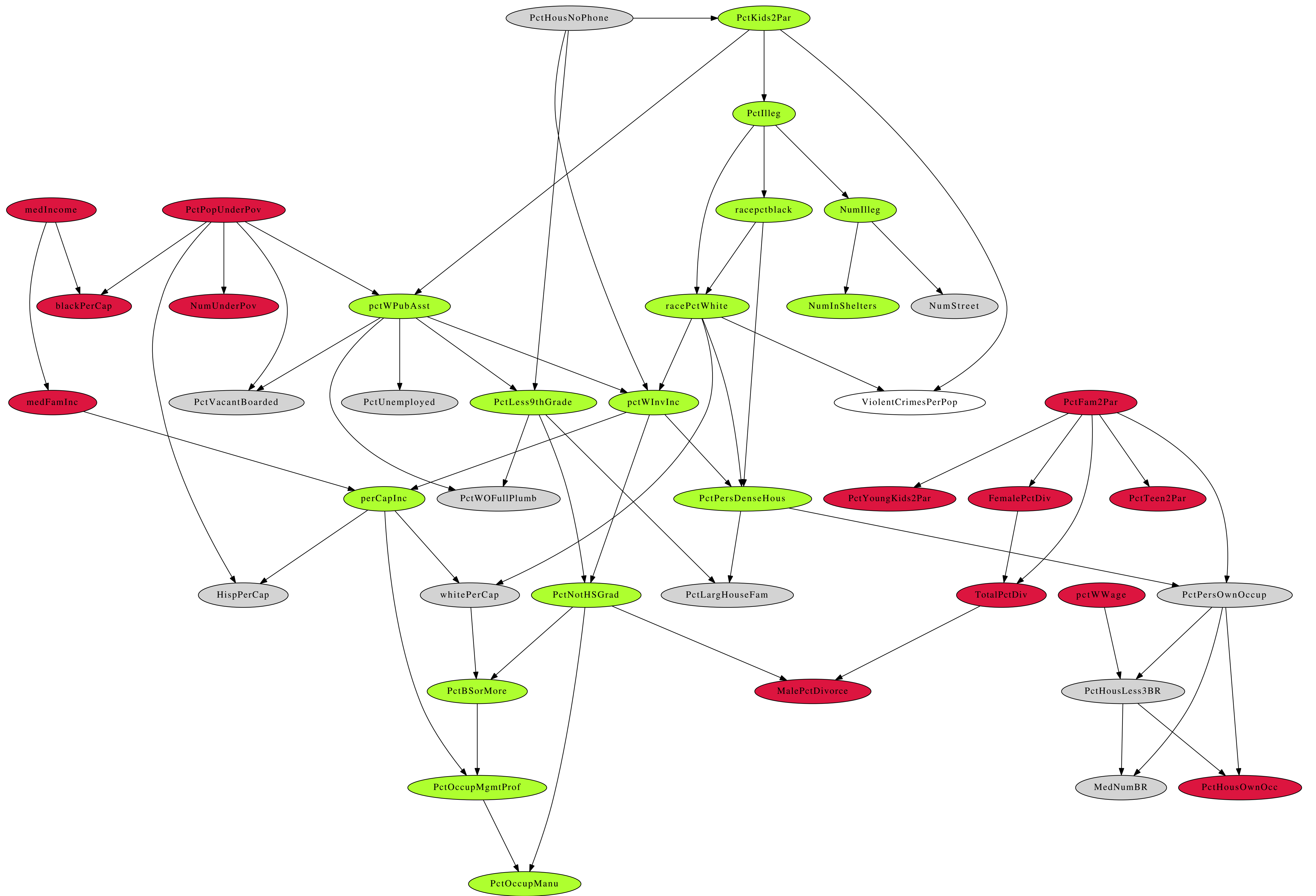


Figure C.1: Bayesian network for crime containing top 40 attributes with the highest crime impact factor.
Legend: Green variables influence the $ViolentCrimesPerPop$ the same way as in the network with all 100 features. Influence of red variables is significantly different and therefore these variables shouldn't be considered.

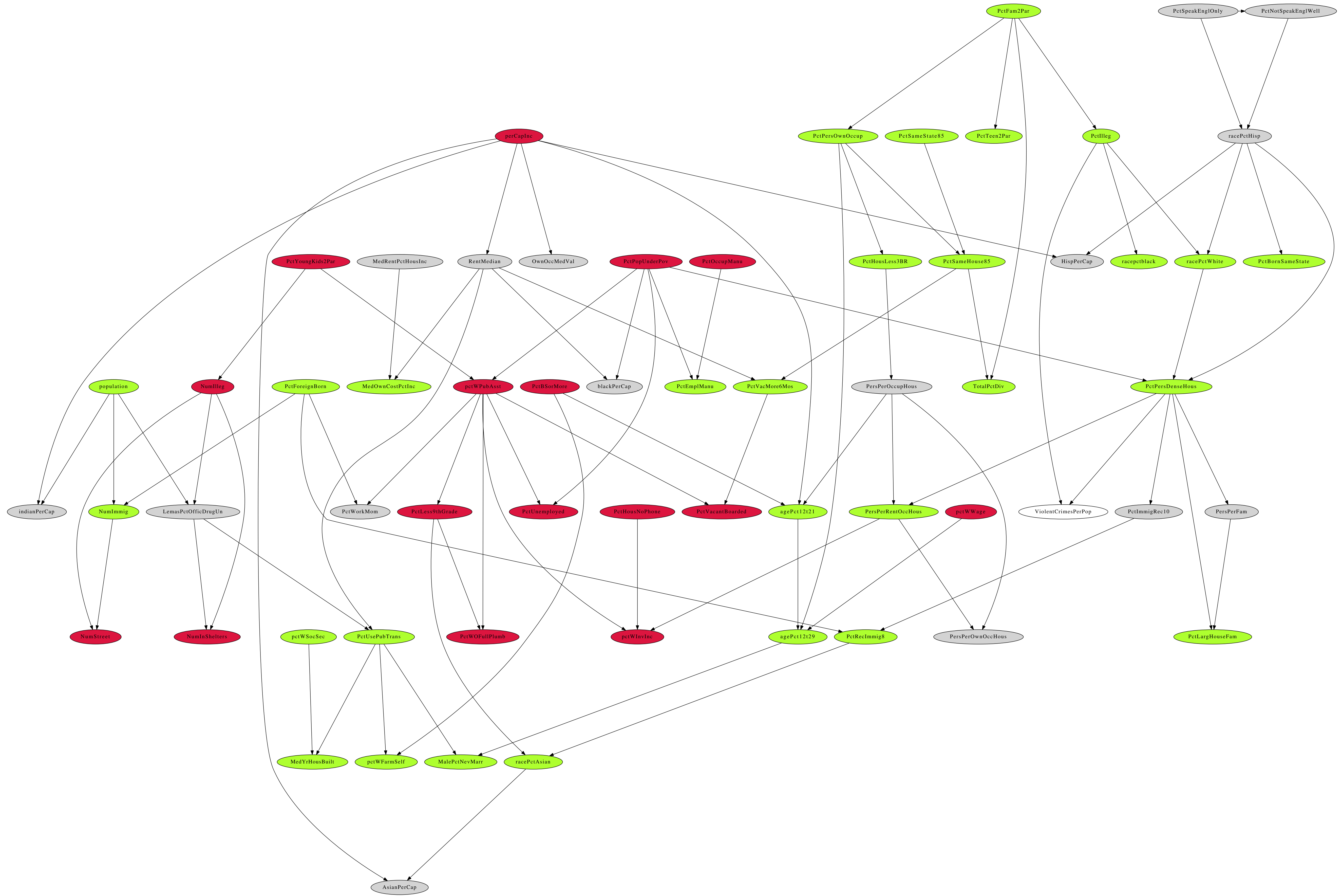


Figure C.2: Final Bayesian network for crime after three rounds of iterative elimination of similar variables.
Legend: Green variables influence the $ViolentCrimesPerPop$ the same way as in the network with all 100 features. Influence of red variables is significantly different and therefore these variables shouldn't be considered.

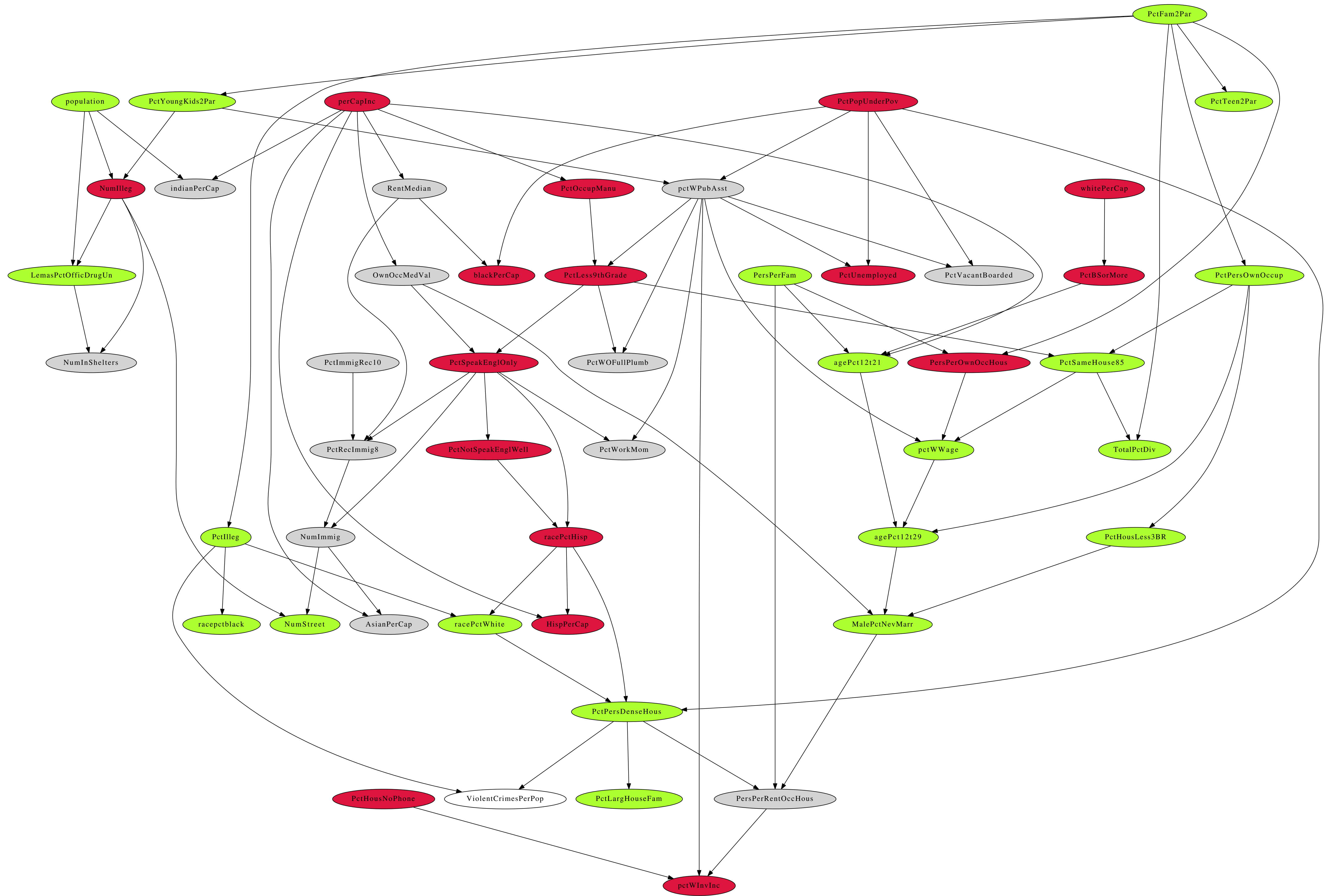


Figure C.3: Final Bayesian network for crime after three rounds of iterative elimination of similar variables and additional removal of variables whose crime impact factor is less than 0.07.
Legend: Green variables influence the $ViolentCrimesPerPop$ the same way as in the network with all 100 features. Influence of red variables is significantly different and therefore these variables shouldn't be considered.