University of Bergen
Department of informatics

Predicting Loss of Inference Accuracy in Bounded Tree-Width Bayesian Networks

Author: Kristian Rosland
Supervisor: Pekka Parviainen

UNIVERSITY OF BERGEN
Faculty of Mathematics and Natural Sciences

June, 2019

Abstract

A Bayesian network (BN) is a compact way to represent a joint probability distribution graphically. The BN consists of a structure in the form of a directed acyclic graph (DAG) and a set of parameters. The nodes of the DAG correspond to random variables, and the absence of an arc encodes a conditional independence between two variables. Computing conditional probabilities from a Bayesian network is known as inference and is an NP-hard problem. However, the problem is fixed-parameter tractable with respect to a property of the network called tree-width. As a consequence, learning networks of bounded tree-width is of interest. When we bound the tree-width of a BN, we may no longer be able to accurately represent the probability distribution and thus we expect some loss of inference accuracy. However, predicting how much the inference accuracy will decay is no easy task. In this thesis, we propose a solution to this problem by quantifying the strength of arcs in the network. We define a measure called dependency strength that measures how strong the dependencies in our network are. We also report results from an experiment to evaluate how well the measure performs in predicting the loss of accuracy in bounded tree-width BNs. Our findings show indications that the measure can be used to predict loss of inference accuracy, but we conclude that more experiments are needed to confirm this.

Acknowledgements

First of all, I would like to thank my wonderful supervisor, Pekka Parviainen, for supporting and encouraging my research. Thank you for always keeping an open door and making me feel welcome. I would also like to thank the entire Department of Informatics at UiB for creating an inspirational and challenging learning environment that I have truly grown in. I would like to thank Kaja Dey, Ragnhild Aalvik, the students at JAFU, and all the other students that I have worked with throughout my degree. Without you, I would never have achieved what I have today.

Lastly, I would like to thank my family for always encouraging me to pursue my dreams, and my girlfriend, Maren Aleksandersen, for all her love and support throughout a hectic final year.

Kristian Rosland
03 June, 2019

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Consider a scenario where you just started working for a medical company, and you are tasked with making a program that takes a patient's symptoms as input, and outputs the probabilities of different diagnoses. As you are a computer scientist and not a doctor, you quickly realize that you don't have the knowledge or experience required to perform medical diagnostics. Making a traditional program that encodes the relationships between all the symptoms and diagnoses is therefore out of the question. Instead, you decide to collect data about previous examinations and apply statistical methods to model the probability of a diagnosis given the symptoms.

One way to implement such a program is to construct a table of probabilities based on the collected data. This way, whenever you are provided with a patient's symptoms, you can compute the conditional probability of each disease by marginalizing out the unobserved symptoms. From a statistical point of view, this is a sound idea, but you might be disappointed with the results. As the number of rows in your table grows exponentially with the number of variables, it is likely that a lot of the entries in the table would have no observations, and the resulting probability of that row would be zero. From a programmer's point of view, this solution is not feasible because of two things: Running time and space usage. The running time of marginalizing out the unobserved symptoms would be exponential w.r.t. the number of symptoms. Representing the table would require $\Omega(k^n)$ space, where n is the number of symptoms and diagnoses, and k is the number of different values each of them can take on.

To avoid this complexity issue, you decide to combine the statistical problem with your favorite field within computer science, namely graph theory. A *probabilistic graphical model (PGM)* is a compact way of representing a joint distribution over a set of random variables by exploiting independencies between them. It models our problem domain as a graph, where the nodes correspond to variables in our data set and the edges encode independence relations between variables. A Bayesian network (BN) [30] is a type of PGM with a directed acyclic graph (DAG) structure accompanied by parameters that describe probabilities. The absence of an arc between two nodes encode a *conditional independence* between the two variables. In other words, if there is no arc from X to Y, then Y is independent of X given some set of variables. We will see an example of this in the next paragraph. The acyclicity constraint of a BN allows us to represent the distribution in a compact way by using the factorization theorem discussed in Section 2.5.1.

In our medical application example, say we have a data set containing observations of the following symptoms and diseases: headache, appetite, coughing, cold, fever, flu and malaria. Examining the data, we can identify dependencies between the variables and express these graphically as the DAG in Figure 1.1. We observe that the probability of Fever is dependent on the probability of Flu and Malaria, and denote the probability of Fever given Flu and Malaria as $P(Fever \mid Flu,\ Malaria)$. An interesting observation is that there is no arc from Flu to Coughing, even though a patient with the flu clearly has a higher probability of coughing. The absence of this arc indicates that there is a conditional independence between them, and in this case they are conditionally independent on Cold. If we know whether or not the patient has a cold, learning whether or not he has the flu does not affect the probability of Coughing. We say that Flu is only influencing Coughing through Cold, and denote the conditional independence as $(Coughing \perp Flu) \mid Cold$. A network with this structure is considered a *causal network*. In causal networks, the arcs can be read as *parent node causes child node*. Notice that many, although not all, BNs are causal.

As mentioned, this structure is accompanied by a set of parameters to make up the BN. Every node in the network stores a *conditional probability distribution (CPD)* describing the probabilities of different values of that particular variable given its parents. The parameters are what specify this distribution. For discrete variables, the CPD is a *conditional probability table (CPT)*, as seen in Figure 1.2. This CPT is specified by eight parameters in total. Notice that as the probabilities in each column always have to sum to 1, we can omit the last row without any loss of information. If all the variables in the example are binary, we need to

store 17 parameters in total for this network. As a comparison, a probability table would require $2^7 - 1 = 127$ parameters.

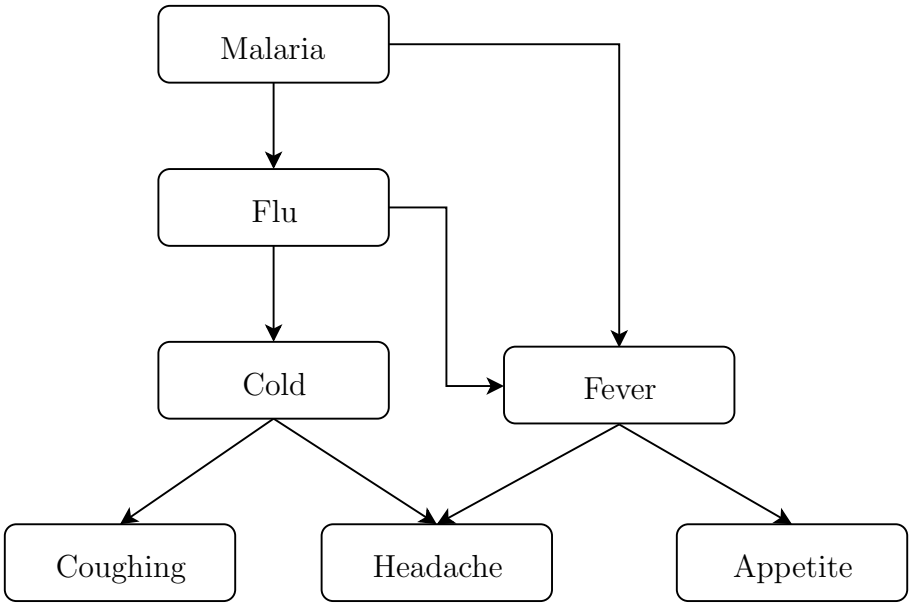


Figure 1.1: Example structure for a diagnosis network.

		(Malaria, Flu)			
		(No, No)	(No, Yes)	(Yes, No)	(Yes, Yes)
Fever	Yes	0.001	0.85	0.90	0.99
	No	0.999	0.15	0.10	0.01

Figure 1.2: Example CPT for Fever.

Constructing a BN consists of two primary tasks: deciding on a structure and learning the parameters associated with it. Deciding on a structure may in some cases be done by domain experts, e.g., a doctor in our example, that know the relationships between the variables. However, in many cases, such expert knowledge is not available, and we must learn the structure from the data. In our medical application, we could learn the relationships between symptoms and diagnoses from data about previous examinations. This process is called structure learning or structure discovery and is about finding the structure that best

fits our data. The structure discovery problem is proven to be NP-hard [9, 10]. We can do both exact and approximate structure learning. In the exact learning approach, we search for the solution that optimizes some defined scoring criterion. In the approximate learning approach, we reduce the complexity of the problem by searching for good, but not necessarily optimal solutions. In this thesis, we consider an anytime algorithm for exact structure learning, meaning that we can exit the solving at any point and get the best approximation so far. The state of the art way of solving exact structure learning is implemented by the GOBNILP-software[1] [3], which formulates the problem as an instance of *integer linear programming (ILP)*. We will use this software as a part of the experiments reported in Chapter 4. Learning the parameters of the network is called *parameter estimation*, and given the structure this is a much easier task. We can use techniques such as *Maximum Likelihood estimation* or *Bayesian estimation* to obtain the parameters of the distribution that best fits our data. These techniques are discussed in Chapter 2.

Reasoning about probabilities in our network is called *inference*. This means querying on the form *what is the probability of X, given a set of evidence* $\boldsymbol{e}$. In our example medical network, a query might be *what is the probability that a patient has malaria, given that he has a bad appetite*. Performing exact inference in a BN is an NP-hard problem [11], and even the approximate inference problem is proven to belong to the NP-hard complexity class [13]. However, both problems are fixed-parameter tractable with respect to a graph property called *tree-width* [33, 26], and thus we often need to either constrain the structure of our network to achieve a low tree-width, or resort to approximate inference. The tree-width of a graph is a measure of how closely the graph resembles a tree and is further explored in Chapter 2. In this thesis, we will consider exact inference on constrained structures. An approach for learning BNs of bounded tree-width is presented by Parviainen et al. [29] and implemented by their software TWILP[2], which is used in the experiments reported in Chapter 4

After constructing the BN, we may encounter a complexity issue. If the tree-width of the learned structure is too high, exact inference is too computationally heavy and thus not feasible. In such a case, we can learn a new, simpler network with a lower tree-width and perform exact inference on that network instead. However, bounding the tree-width does come at a cost: A simpler network structure will not be able to correctly represent the independencies in the distribution, and therefore the quality of inference will decay. The

[1] https://www.cs.york.ac.uk/aig/sw/gobnilp/
[2] https://bitbucket.org/twilp/twilp/

problem is that measuring the impact of such simplifications is hard, as we do not have access to inference results from the original network. If we decide to learn a bounded tree-width network, we cannot know how much the inference results will decay.

In this thesis, we pose a solution to this problem by formulating a property of a BN called *dependency strength*. We propose a measure of individual arcs, as well as how to combine them into the strength of a network. The ultimate goal is to provide a measure that can be used as a tool to predict the impact on inference quality when bounding the tree-width of Bayesian networks. If one decides to learn a bounded network, such a measure could be used to predict loss of inference accuracy and thus provide valuable insight for the modeler about the uncertainty of the inference results.

This is not the first attempt at quantifying the impact of simplifications of Bayesian networks. Empirical evaluations have shown [32] that roughly 20 percent of the arcs in a network can be removed with minimal effect on the classification accuracy, if the arcs are chosen wisely. One of the main approaches to identifying candidates for arc removal is to define a measure of arc strength and pick the weakest arcs. Boerlage [5] defines the strength of an arc between two binary variables as the maximum influence a parent node can have on a child node. Koiter [23] defines the measure in terms of the posterior probability distribution of the child node when fixing the value of the parent node. Nicholson and Jitnah [27] define the strength of an arc in terms of the mutual information between the parent and child.

We define our measure in terms of likelihood. The strength of an arc is defined as the difference in log-likelihood with and without the arc as part of our structure. The intuition behind this is that if there is a significant drop in likelihood, then the arc in question was important in explaining the observed data. If the drop is small, then the arc was less important in the explanation, and therefore should have less impact on inference quality if we remove it. In Chapter 3 we define a measure of dependency strength and derive the equation used to calculate the dependency strength of both individual arcs and combined networks. Even though we did not realize this before we started, we will see that we arrive at a similar definition of arc strength as Nicholson and Jitnah [27]. They introduced this measure to allow inference algorithms to focus their work on more relevant parts of the network, and thereby allocating computational resources more efficiently. As mentioned, we intend to use the measure to predict the loss of accuracy when bounding the tree-width of a BN. In Chapter 4 the results from the experiments are reported in order to evaluate our measure as a predictor of loss in inference accuracy, and Chapter 5 concludes our findings.

Chapter 2

Preliminaries

2.1 Independence Between Random Variables

An important concept of this thesis is the notion of independence between variables. A *dependent* variable X is a variable whose probability distribution depends on the value of another variable, Y. By this, we mean that observing Y changes the probability distribution of X.

Definition 1 (Independence [18])**.** *Variable X and Y are independent, denoted $(X \perp Y)$, if their joint probability is equal to the product of their marginal probabilities, $P(X, Y) = P(X)P(Y)$. Equivalently, $(X \perp Y)$ if $P(X) = P(X|Y)$.*

Definition 2 (Conditional independence)**.** *Variable X and Y are conditionally independent given Z, denoted $(X \perp Y)|Z$, if $P(X|Y, Z) = P(X|Z)$.*

From the definition of independence, we see that if the marginal probability of X is equal to the conditional probability of X given Y, then X is independent of Y. From this, it is clear that knowing the value of Y does not affect our beliefs about X. We can quantify the dependence of Y on X as the size of the change in our beliefs about X introduced by the observation of Y. *Mutual information* is an information theoretic approach to quantify the mutual dependence between two variables [34]. This measure quantifies the reduction in

uncertainty of X introduced by observing the value of Y. The mutual information between X and Y is given by

$$MI(X,Y) = \sum_{x,y} P(x,\ y) \log \frac{P(x,\ y)}{P(x)P(y)} \tag{2.1}$$

$$= \sum_{x} P(x) \sum_{y} P(y|\ x) \log \frac{P(y|\ x)}{P(y)}. \quad \text{(applying the chain rule of probability)}$$

Notice that we use the common notation x to denote a value of X and $P(x)$ to denote $P(X = x)$. The mutual information determines how similar the joint distribution of X and Y, $P(X,Y)$ is to the product of their marginal distributions, $P(X)P(Y)$. We see from Equation 2.1 that as the joint distribution approaches the product of the marginal distributions, the term $\log \frac{P(x,y)}{P(x)P(y)}$ and the mutual information approaches zero.

2.2 Graph Theory

In order to understand the structure of a BN, we need to understand basic graph theory. A graph is a data structure describing the pairwise relations between objects. An *undirected graph* is denoted $G = (N, E)$, where N is a set of nodes and E is a set of edges. An *edge* is a set of two nodes, $\{u, v\}$, describing a relation between u and v. The relationship is symmetric, meaning that $\{u, v\}$ implies $\{v, u\}$. If there exists an edge $\{u, v\}$, we say that u and v are adjacent, and they are both incident to $\{u, v\}$. Figure 2.1a shows an example of an undirected graph.

A *directed graph*, denoted $G = (N,\ A)$, is a graph where the node pairs are ordered and called *arcs*. An arc is denoted (u, v), and encodes a one-way, non-symmetric relation from u to v. The first node in the pair is called the *parent*, while the second is called the *child*. The set of parents of a node v in a directed graph is denoted A_v. A node with no parents is called a *root node*. Figure 2.1b shows an example of a directed graph. Notice that node A and E are root nodes in this example. The *skeleton* of a directed graph is an undirected graph with the exact same structure, excluding the direction of the arcs. A *v-structure* in a

directed graph (N, A) is a triplet of nodes X, Y and Z such that (X, Y), $(Z, Y) \in A$ and (X, Z), $(Z, X) \notin A$. Figure 2.1d shows an example of a v-structure. Notice that there is no arc between X and Z.

A *directed acyclic graph (DAG)* is a graph that contains no directed cycles. A directed cycle is a path following directed arcs that start and end in the same node. More formally, a cycle is defined as an arc-sequence of length k s.t. for the i-th arc, $u_i = v_{i-1}$, and $u_1 = v_k$. Figure 2.1b shows an example of a DAG, while Figure 2.1c shows an example of a cycle marked in red containing nodes B, C and D.

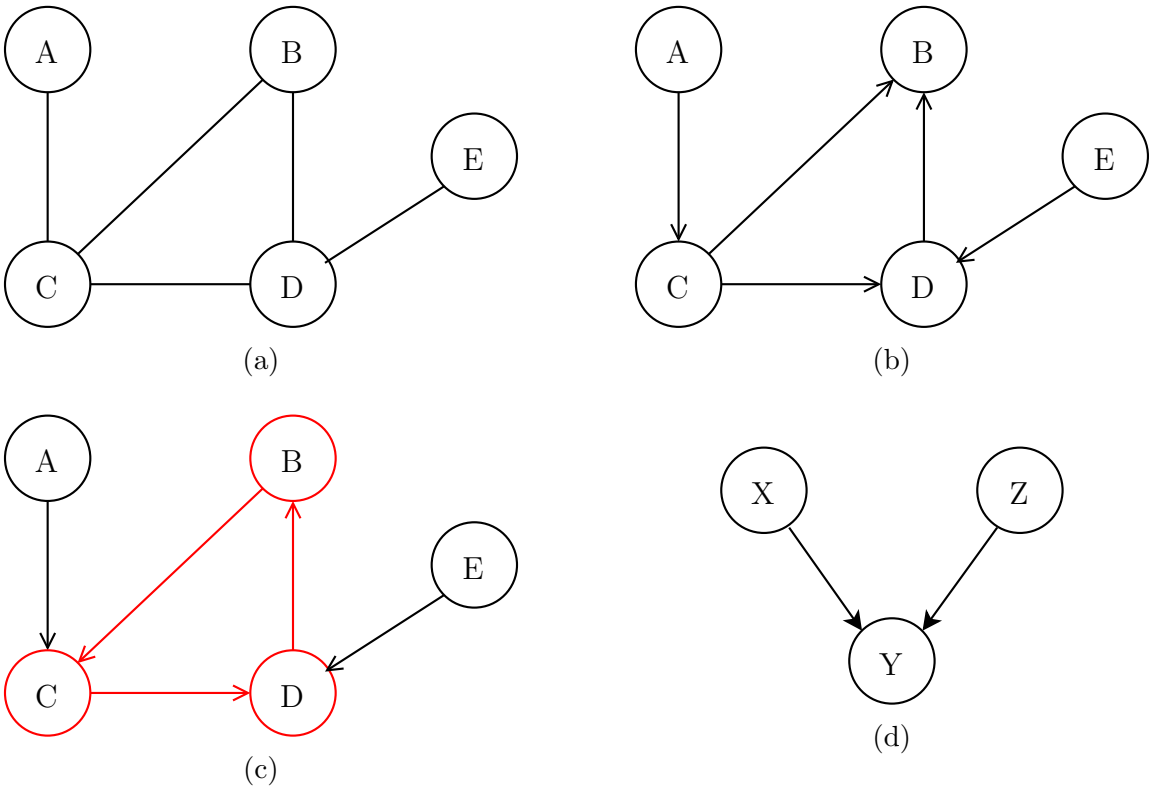


Figure 2.1: (a) An example of an undirected graph of five nodes, (b) an example of a directed (acyclic) graph of five nodes, (c) a directed graph with a cycle marked in red, and (d) an example of a v-structure.

2.3 Tree-Width

A *tree* is an undirected graph with no cycles. The *tree-width* of a graph is a measure of how closely the graph resembles a tree. If G is a tree then the tree-width of G is 1, and if G is a clique of n nodes, then the tree-width of G is $n - 1$. The property can be defined in several equivalent ways, and we start by defining it in terms of a *tree decomposition*. A tree decomposition of an undirected graph $G = (N,\ E)$ is a pair (X, T) where X is a family of subsets of N and T is a tree with nodes corresponding to the sets of X. For $(X,\ T)$ to be a tree decomposition of G, the following must hold:

- The union of all sets X_i of X contains all nodes in N: $\cup_i X_i = N$.
- For every edge $\{u, v\}$ in E, there exists a subset X_i of X such that $u \in X_i$ and $v \in X_i$.
- Every node that appears in both X_i and X_j also appears in every node X_k of X on the unique path from X_i to X_j in T.

There are many possible tree decompositions of a graph G. A trivial tree decomposition of G can be achieved by putting all nodes of N in a single node of T. The width of a tree decomposition is equal to the size of the largest set in X minus one. The tree-width of graph G is defined as the minimum width of all possible tree decompositions of G [33]. Figure 2.2 shows an example graph and a possible tree decomposition of it. This decomposition has a minimal width, and the tree-width of the graph is 2.

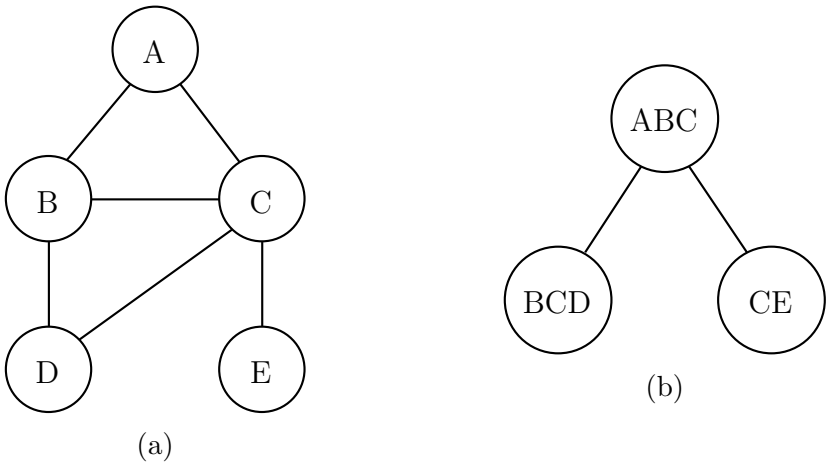


Figure 2.2: (a) An undirected graph of five nodes and (b) a possible tree decomposition of it.

The tree-width of a DAG is equal the tree-width of the corresponding *moralized graph*. The moralized graph of a DAG (N, A) is an undirected graph containing an edge $\{u, v\}$ for every $(u, v) \in A$, and an edge $\{x, y\}$ for every pair of arcs $(x, i), (y, i) \in A$. In Section 2.5.5, we will look at a definition of tree-width in terms of optimal vertex elimination order.

2.4 Likelihood

To help us formulate our measure of dependency strength in Chapter 3, we need to understand the term *likelihood*. The likelihood function, $\mathcal{L}(D : \theta)$, describes the probability that a data-generating model with parameter θ produced the observed data, D. Let D be an $M \times n$ matrix, where M is the number of data points and n is the dimension of each data point. Assuming that all observations are independent of each other, the likelihood is expressed as

$$\begin{aligned}\mathcal{L}(D : \theta) &= P(D \mid \theta) \\ &= \prod_{m=1}^{M} P(D_{m1}, D_{m2}, ..., D_{mn} : \theta).\end{aligned}$$

To get an intuition of the likelihood function, we look at an example coin toss experiment. Let us say the coin was tossed 10 times, and it landed heads up (H) 6 of them. In this example, the coin is the data generating model. Given this data, we can examine the likelihood of different parameter values. If we set the parameter to $p_H = 0.5$ and calculate the binomial probability, we get

$$\mathcal{L}(D : p_H = 0.5) = \binom{10}{6} 0.5^6 * 0.5^4 = 0.2051.$$

If we instead set the parameter value to $p_H = 0.6$, we get

$$\mathcal{L}(D : p_H = 0.6) = \binom{10}{6} 0.6^6 * 0.4^4 = 0.2508.$$

This tells us that if the parameter of the coin is $p_H = 0.5$, the probability of observing D is 0.2051, while if the parameter of the coin is $p_H = 0.6$, the probability of observing D is 0.2508.

2.5 Bayesian Networks

A Bayesian network is a compact way to represent a joint probability distribution. It consists of two parts: Structure and parameters. The structure is a DAG, (N, A), where each node in N represents a variable. The absence of an arc expresses a conditional independence between two variables. The parameters specify a set of local *conditional probability distributions (CPDs)*, θ. Every node stores a CPD which specifies the distribution of the corresponding variable given the variables corresponding to its parents. If the node is a root node, the CPD is the marginal distribution of the corresponding variable. The type of distribution may vary, as will the number of parameters required to specify it. We denote a Bayesian network by $(A,\, \theta)$.

2.5.1 Factorization Theorem

The compactness of Bayesian networks lies in the number of parameters required to represent the joint distribution. By identifying the conditional independencies and representing the distribution as a collection of conditional distributions, the parameter space is considerably reduced. We say that we are factorizing the distribution according to the structure of our network. This factorization is only valid if the joint distribution satisfies the *Markov condition*, that is, every variable is conditionally independent of its non-descendants given its parents. According to the factorization theorem of BNs, the probability distribution $P(X_1, ..., X_n)$ can be factorized as $\prod_{i=1}^{n} P(X_i \mid A_{X_i})$, where A_{X_i} is the parent set of X_i.

For the context of this thesis, we will work with discrete variables. We denote the cardinality of variable v by C_v. If we represent the joint probability distribution without factorizing it, the total number of parameters required is $\prod_{v \in N} C_v - 1$, where N is the set of nodes in the network. For a distribution of 10 binary variables, we would need 1023 parameters. For a factorized representation of the distribution, this number is significantly lower. Each local CPD consists of C_v number of rows and $\prod_{u \in A_v} C_u$ number of columns. Since the probabilities of each column must always sum to 1, we can omit the last row. The total number of parameters required is then less than $\sum_{v \in N} (C_v - 1) \prod_{u \in A_v} C_u$. If we limit the maximum number of parents for nodes in our network to for example 4, then the number of parameters required in the worst case is less than $10 * 2^4 = 160$.

2.5.2 Likelihood of Bayesian Networks

The likelihood of a BN $(A,\ \theta)$ is a measure of how well the network fits the data, D. By combining the definition of likelihood with the factorization theorem from Section 2.5.1, we arrive at the following expression for the likelihood of a BN:

$$\begin{aligned}\mathcal{L}(D|(A,\ \theta)) &= \prod_{m=1}^{M} P(D_m\ :\ (A,\ \theta)) \\ &= \prod_{m=1}^{M} \prod_{v \in N} P(D_{mv}|D_{mA_v} : (A,\ \theta_v)) \\ &= \prod_{v\ \in\ N} \prod_{m=1}^{M} P(D_{mv}|D_{mA_v}\ :\ (A,\ \theta_v)),\end{aligned}$$

where θ_v is the CPT of v and $\prod_{m=1}^{M} P(D_{mv}|D_{mA_v}\ :\ (A, \theta_v))$ is called the local likelihood of variable v.

2.5.3 Parameter Estimation

Given the structure of a Bayesian network, we use parameter estimation to specify the parameters. The input of the parameter estimation process is a structure in the form of a DAG and a data set D of observations. The output is a set of parameters specifying the conditional probability distributions (CPDs) of the network. For networks of discrete variables like the ones used in the context of this thesis, parameter estimation means filling in the CPT for each node. The two most common approaches to estimating parameters for BNs are *Maximum Likelihood Estimation (MLE)* and *Bayesian estimation.*

MLE is based on maximizing the likelihood. In this approach, we assume that θ is some fixed, unknown parameter and seek to find the θ^* that maximizes the likelihood function $\mathcal{L}(D|\ (A,\ \theta))$. The likelihood of a BN, as defined in Section 2.5.2, decomposes into local likelihoods that we can maximize separately. For discrete variables, the likelihood further decomposes into the product of the likelihood of multinomial distributions, one for each

parent configuration, which we can maximize independently. Estimating the parameters of a CPT is then done by counting the number of occurrences. As an example, suppose we have a binary variable Y, whose only parent is the binary variable X. The parameter $\theta_{Y=0|X=0}$ is then simply $\frac{M_{Y=0,X=0}}{M_{Y=0,X=0}+M_{Y=1,X=0}}$, where $M_{Y=y,X=x}$ is the number of observations where $Y = y$ and $X = x$ in the data set. One drawback with MLE is that it does not quantify uncertainty. To get an intuition of this, we revisit the coin toss example from Section 2.4. We saw that the most probable parameter in this scenario was $p_H = 0.6$. Because of our prior knowledge about coins and the fact that we only observed 10 tosses, we would likely still conclude that the coin was fair. However, if we had observed 10 000 tosses and got 6 000 heads, we would likely conclude that the coin was counterfeit. The MLE approach would estimate the same parameters in both of these cases. Another drawback with MLE is that if one value combination is never observed, the estimated probability of observing this combination is zero. This is usually not desirable, and we would rather have the estimated probability approach zero as the number of observed data points approach infinity.

Bayesian estimation is based on the Bayesian formalism, where we treat everything that we are uncertain about as a random variable. Hence, the parameter is treated as a random variable θ, with a distribution that is updated over time. In this approach, we encode our prior beliefs about the parameters and update this belief for every observation. We use the data to get the posterior probability of θ, $P(\theta|D)$, which by Bayes rule is proportional to $P(D|\theta)P(\theta)$. The $P(\theta)$ is the prior probability distribution of θ, which we assume to be a Beta distribution specified by a set of hyperparameters α. We call these hyperparameters *pseudo-counts*, and we can look at them like imaginary observations based on our prior beliefs. The sum of these pseudo-counts is called the equivalent sample size, and the larger this number is, the more confident we are in our prior beliefs. Working with distributions as parameters can be difficult. Therefore, in practice, we often resort to point estimation and estimate the parameters by adding the pseudo-counts to the observed counts. In the example from the previous paragraph, the parameter $\theta_{Y=0|X=0}$ would be calculated as $\frac{M_{Y=0,X=0}+\alpha_{Y=0,X=0}}{M_{Y=0,X=0}+M_{Y=1,X=0}+\alpha_{X=0}}$, where $\alpha_{Y=y,X=x}$ is the pseudo-counts for $X = x, Y = y$. If we set the pseudo-counts to 0, the Bayesian estimator is exactly the same as the MLE. As the number of data points in D approaches infinity, the Bayesian estimate approaches the maximum likelihood estimate.

2.5.4 Structure Learning

Structure learning, also called structure discovery, is the task of learning the structure of a Bayesian network from data. We apply this whenever the structure of the BN is unknown, and the domain expertise is either not present or not adequate. The goal of the structure learning can be to create a statistical model in order to perform inference or to discover interrelationships between variables in the data. There exist algorithms for both exact and approximate structure learning, but only the former will be considered in the context of this thesis.

The *search-and-score* approach is a common approach to solving the structure learning that turns the problem into an optimization problem [21]. In this approach, we define a scoring function that evaluates how well a given structure fits the data. We then use this scoring to search for the optimal network in the set of possible network structures. There are $2^{\Omega(n^2)}$ possible structures for a BN with n nodes, and as mentioned in Chapter 1, the problem of finding the optimal structure resides in the NP-hard complexity class [9, 10].

The scoring function takes two arguments, a candidate structure A and a data set D. We denote the score function by $Score(A,\ D)$ and proceed to discuss some of the common choices of score functions. Perhaps the simplest score function is the likelihood score in which we find the structure that maximizes the log-likelihood given the data. The *likelihood score* is given as $Score_L(A,\ D) = \ell((\hat{\theta}, A) : D)$, where $\hat{\theta}$ is the MLE of the parameters given structure A and data set D. A drawback with this score function is that it almost always favors more arcs over fewer arcs. This is caused by the fact that the mutual information between two variables in the observed data set is rarely zero, and thus the likelihood increases if we add an arc between them. In order to deal with this problem, we can introduce a complexity penalty in the scoring. The *BIC score* is similar to the likelihood score, but includes an extra term penalizing complexity. The BIC score is defined as $Score_{BIC}(A,\ D) = \ell((\hat{\theta}, A) : D) - \frac{\log M}{2} Dim[A]$, where M is the size of the data set and $Dim[A]$ is the number of independent parameters in A. The *Bayesian score* function is used to find the structure that maximizes the posterior probability $P(A|D)$ of the DAG A given data D. Applying Bayes theorem, the posterior can be expressed as:

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

where $P(D|A)$ is the marginal likelihood, $P(A)$ is the prior over graph structures that encodes our prior beliefs about the structure, and $P(D)$ is the marginal probability of the data D. As it is more convenient to work with the logarithm of the posterior, we define the score of a structure A as

$$Score_B(A,\ D) = \log P(D|A) + \log P(A).$$

In the experiments reported in Chapter 4 we used the *Bayesian Dirichlet equivalence uniform (BDeu) score*. We will not go further into details about this scoring function, and the interested reader is referred to for example Heckerman et al. [21] or Buntine [6]. As we will see in the next section, the complexity of inference is bounded by the tree-width of the network structure. As a consequence, approaches for learning BN structures of bounded tree-width have been granted some attention in recent years [24, 16, 7]. Parviainen et al. [29] introduced an approach to learn BN structures by converting the learning problem into a mixed-integer linear programming (MILP) problem and solving it using state-of-the-art integer programming solvers. Their idea is implemented in the TWILP software that was used in the experiments reported in Chapter 4.

2.5.5 Inference

Bayesian networks allow us to reason about the probabilities of the variables in our model. If we observe some set of evidence variables, we can compute the conditional probability distribution of the non-evidence variables. This process is called probabilistic inference and can be performed both exactly and approximately. Given a set of query variables $\mathbf{X}$ with corresponding values $\mathbf{x}$, a set of evidence variables $\mathbf{E}$ with corresponding values $\mathbf{e}$, and a set of remaining variables $\mathbf{Y}$, the conditional probability distribution of $\mathbf{X} = \mathbf{x}$ given $\mathbf{E} = \mathbf{e}$ is

$$P(\mathbf{X} = \mathbf{x}|\mathbf{E} = \mathbf{e}) = \frac{P(\mathbf{X} = \mathbf{x}, \mathbf{E} = \mathbf{e})}{P(\mathbf{E} = \mathbf{e})} \propto \sum_{\mathbf{y} \in \boldsymbol{\alpha}_Y} P(\mathbf{X} = \mathbf{x}, \mathbf{E} = \mathbf{e}, \mathbf{Y} = \mathbf{y}),$$

where $\boldsymbol{\alpha}_Y$ is the set of all possible assignments of the variables in $\mathbf{Y}$. Recall that the probability distribution factorizes into CPDs according to the BN structure, leading to the following expression for the conditional probability distribution:

$$P(\mathbf{X} = \mathbf{x}|\mathbf{E} = \mathbf{e}) \propto \sum_{\mathbf{y} \in \boldsymbol{\alpha}_Y} \prod_{X \in \mathbf{X}} P(X = \hat{x}|A_X = \hat{A}_X) \prod_{E \in \mathbf{E}} P(E = \hat{e}|A_E = \hat{A}_E) \prod_{Y \in \mathbf{Y}} P(Y = \hat{y}|A_Y = \hat{A}_Y),$$

where the $\hat{}$ operator denotes a value taken from $\mathbf{x}$, $\mathbf{e}$, or $\mathbf{y}$.

The inference algorithms derive $P(\mathbf{X} = \mathbf{x}|\mathbf{E} = \mathbf{e})$ by marginalizing out all the variables in $\mathbf{Y}$. In the context of this thesis, algorithms for exact inference are of most interest, and a common algorithm for exact inference is called *variable elimination (VE)*. This algorithm is used in the experiments reported in Chapter 4. Before explaining this algorithm in detail, it is necessary to define the term *factor*. A factor is simply a function, $\phi(X_1, ..., X_k)$ that takes a set of arguments $\{X_1, ..., X_k\}$ and produces a real value. The set of arguments is called the *scope* of the factor. A probability distribution over a set of variables $\mathbf{X}$ is a factor, where $\mathbf{X}$ is the scope and the probabilities for each assignment of $\mathbf{X}$ are the values. The size of a factor is equal to the number of values that it can produce. In the case of a probability distribution over the set of variables $\mathbf{X}$, the factor size is equal to $\prod_{x \in \mathbf{X}} C_x - 1$, where C_x is the cardinality of x. The VE algorithm eliminates one variable at a time in a given order. To eliminate a variable X means to find all factors in which X appears, multiplying them together to generate a new factor and then marginalizing out X.

To better understand this algorithm, we will look at an example distribution $P(X, Y, Z)$. We factorize the distribution according to the structure in Figure 2.3, resulting in $P(X)P(Y|X)P(Z|Y)$. If we want to infer $P(Z)$, we sum over all possible values of X and Y, resulting in the following expression

$$P(Z) = \sum_{x,y} P(x)\ P(y|x)\ P(Z|y)$$

Now let us look at how VE goes about eliminating variable Y. We start by identifying the factors $\phi_Y(Y, X)$ and $\phi_Z(Z, Y)$ that include Y. We calculate the factor product by multiplying the corresponding rows together, creating a factor $\phi(X, Y, Z)$. We continue by marginalizing out Y, resulting in a factor $\mathcal{T}(X, Z)$. Table 2.1 shows an example of such

marginalizing of Y. Notice that we use $\mathcal{T}$ to denote factors resulting from the elimination of a variable. Lastly, we remove the two factors of Y from the set of factors and add our new factor $\mathcal{T}(X, Z)$. By eliminating Y we have replaced the two factors that included Y with the new factor, $\mathcal{T}(X, Z)$, and our equation is now $P(Z) = \sum_x P(x)\mathcal{T}(x, Z)$.

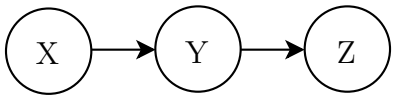


Figure 2.3: Example of a simple BN structure with variables X, Y and Z.

X	Y	Z	
x_1	y_1	z_1	0.10
x_1	y_1	z_2	0.15
x_1	y_2	z_1	0.30
x_1	y_2	z_2	0.05
x_2	y_1	z_1	0.05
x_2	y_1	z_2	0.15
x_2	y_2	z_1	0.02
x_2	y_2	z_2	0.18

(a)

X	Z	
x_1	z_1	0.40
x_1	z_2	0.20
x_2	z_1	0.07
x_2	z_2	0.33

(b)

Table 2.1: (a) An example of factor $\phi(X, Y, Z)$, and (b) the resulting factor $\mathcal{T}(X, Z)$ after Y is marginalized out.

The complexity of this algorithm depends on the order of elimination. As both taking the factor product and marginalizing out a variable are linear operations, the VE algorithm is linear in the largest factor generated during the elimination. However, the size of the largest factor is exponential in the number of variables in its scope. By picking the elimination order wisely we can minimize the number of variables in the scope of the largest factor, and thereby reduce the complexity of the algorithm. When reasoning about the VE algorithm, it can be helpful to consider the elimination operations as operations on a graph. We can create what we call the *induced graph*, which is an undirected graph corresponding to the moralization of the BN structure. This graph has an edge between all variables that share a common factor. When we eliminate a variable, we can think of it as removing the corresponding node from the induced graph. However, since we are now creating a new factor with all the neighbors of this node, we must add new edges to the graph. These edges induced by intermediate factors in the elimination are called *fill-in edges*. The number of variables in the largest

factor is then equivalent to the largest clique in the induced graph.

Let us consider the elimination of the variables in the example BN structure in Figure 2.4a, and say that we want to compute $P(A)$. The factors of this graph are $\phi_A(A)$, $\phi_B(B)$, $\phi_C(C,\ E)$, $\phi_D(D,\ E)$, and $\phi_E(A,\ B,\ E)$ as implied by the moralized graph in Figure 2.4b. We start by considering the elimination order $\omega = [E,\ D,\ C,\ B]$:

$$
\begin{aligned}
P(A) &\propto \sum_{B,C,D,E} \phi_A(A)\phi_B(B)\phi_C(C,E)\phi_D(D,E)\phi_E(A,B,E) \\
&= \sum_{B,C,D} \phi_A(A)\phi_B(B) \underbrace{\sum_{E} \phi_C(C,E)\phi_D(D,E)\phi_E(A,B,E)}_{\mathcal{T}_1(A,B,C,D)} \\
&= \sum_{B,C} \phi_A(A)\phi_B(B) \sum_{D} \mathcal{T}_1(A,B,C,D) && (E \text{ is eliminated}) \\
&= \sum_{B} \phi_A(A)\phi_B(B) \sum_{C} \mathcal{T}_2(A,B,C) && (D \text{ is eliminated}) \\
&= \phi_A(A) \sum_{B} \phi_B(B)\mathcal{T}_3(A,B) && (C \text{ is eliminated}) \\
&= \phi_A(A)\mathcal{T}_4(A) && (B \text{ is eliminated})
\end{aligned}
$$

We see that the largest intermediate factor in this elimination is $\mathcal{T}_1(A,B,C,D)$, generated by the elimination of E. This factor introduces five fill-in edges in the induced graph, as shown in orange in Figure 2.4c. If we change the order of elimination to $\omega = [B,\ C,\ D,\ E]$, the elimination steps are:

$$
\begin{aligned}
P(A) &\propto \sum_{B,C,D,E} \phi_A(A)\phi_B(B)\phi_C(C,E)\phi_D(D,E)\phi_E(A,B,E) \\
&= \sum_{C,D,E} \phi_A(A)\phi_C(C,E)\phi_D(D,E) \sum_{B} \phi_B(B)\phi_E(A,B,E) \\
&= \sum_{D,E} \phi_A(A)\phi_D(D,E)\mathcal{T}_5(A,E) \sum_{C} \phi_C(C,E) && (B \text{ is eliminated}) \\
&= \sum_{E} \phi_A(A)\mathcal{T}_5(A,E)\mathcal{T}_6(E) \sum_{D} \phi_D(D,E) && (C \text{ is eliminated}) \\
&= \phi_A(A) \sum_{E} \phi_1(A,E)\mathcal{T}_6(E)\mathcal{T}_7(E) && (D \text{ is eliminated}) \\
&= \phi_A(A)\mathcal{T}_8(A) && (E \text{ is eliminated})
\end{aligned}
$$

Notice that with this elimination order, the largest factor of the elimination is ϕ_E, and that this elimination does not introduce any fill-edges in the induced graph. From the example above, we observe that the size of the largest factor, and thus the complexity of the VE algorithm is dependent on the order of elimination. Recall that both exact and approximate inference problems belong in the NP-hard complexity class and that they are fixed-parameter tractable w.r.t. the tree-width property. In Section 2.3 we gave a definition of tree-width in terms of tree decompositions. However, we can also define the tree-width in terms of optimal vertex elimination orders. In their work on finding a best-first search algorithm for tree-width [15], Dow and Korf define the tree-width property in terms of optimal node elimination orders in a graph. *Eliminating a node* v is defined as removing v from the graph and adding an edge between all of v's neighbors that are not already adjacent. An *elimination order* is an ordering of the nodes in the graph. The *width* of a such an ordering is the maximum degree of any node at the time it is removed. The *tree-width* of the graph is defined as the minimum width over all possible elimination orders. Any order whose width is equal to the tree-width is an optimal elimination order.

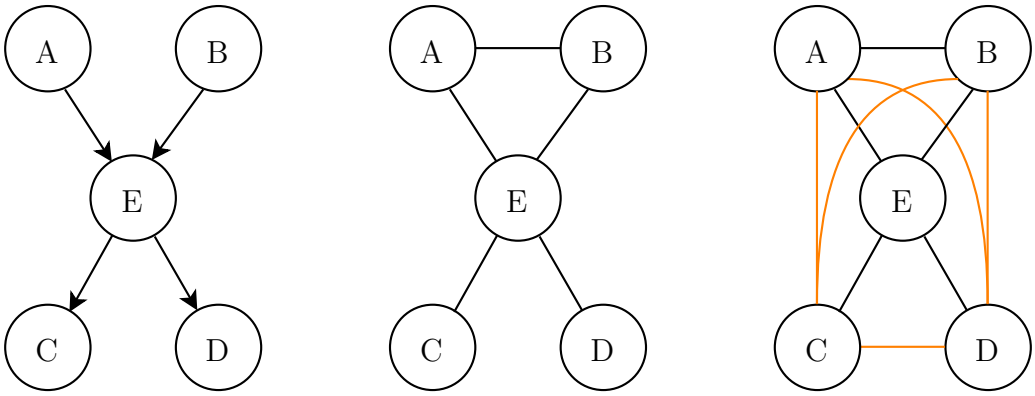


Figure 2.4: (a) The structure of an example BN, (b) a moralization of the structure and (c) the graph induced by an elimination order starting with E.

Combining this definition of tree-width with our graphical representation of variable elimination, we can see why the inference is exponential w.r.t. tree-width. The elimination process described in the tree-width definition corresponds to eliminating variables, where we add edges between all neighbors of the eliminated node. We know that the VE algorithm is linear in the size of the largest factor in the elimination and that the size of the factor is exponential in the degree of the node at the point of elimination. Since the tree-width is per definition the maximum degree of any node eliminated, it is clear that the upper bound complexity of VE is exponential w.r.t. tree-width.

2.5.6 Markov Equivalence

As mentioned earlier, a Bayesian network encodes a set of conditional independencies in the data. However, there are several network structures that encode the same conditional independencies. We say that these networks belong to the same *equivalence class*, and that they are *Markov equivalent*.

Definition 3 (Markov equivalence [8])**.** *Two networks are Markov equivalent if the set of distributions that can be represented by one of the DAGs is identical to the set of distributions that can be represented by the other one.*

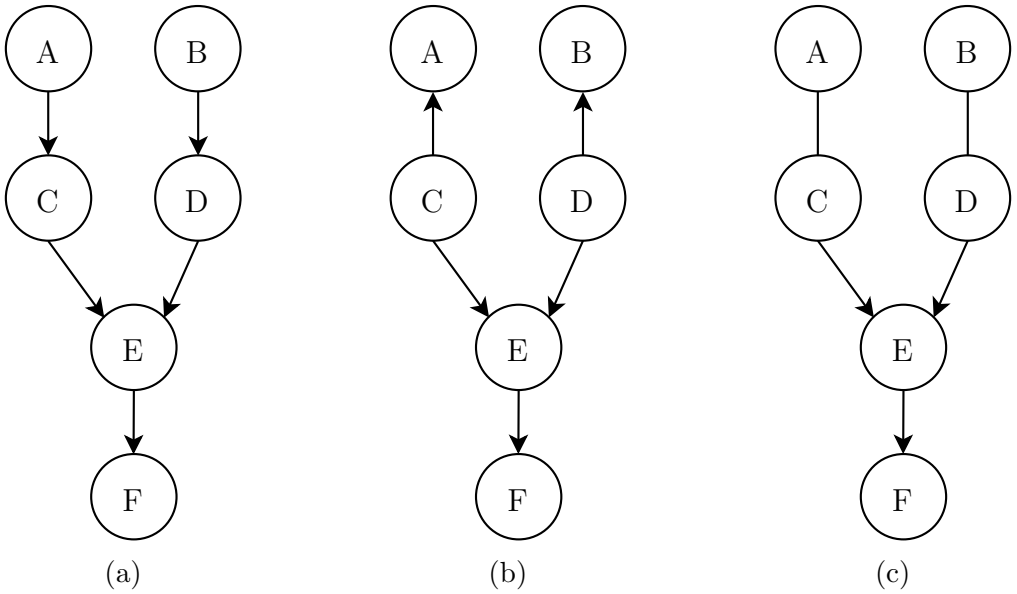


Figure 2.5: Subfigures (a) and (b) are two equivalent DAGs and (c) is the pattern describing their equivalence class.

An equivalence class can be represented by a *pattern*. A pattern is a graph corresponding to the structure of a BN where we remove the direction of all arcs except those who are members of a v-structure, and those who would introduce a new v-structure if they were reversed [37]. The directed arcs in the pattern are called *compelled* edges, and the undirected edges are called *reversible edges*. Patterns may be used when measuring the distance between two networks, as discussed in Section 2.7. An algorithm for finding the pattern of a Bayesian network is proposed by Chickering [8] and outlined in Algorithm 1. For proof of the correctness of this algorithm, the reader is referred to Chickering [8]. Two equivalent network structures and the pattern describing their equivalence class are shown in Figure 2.5. Notice that the arc from E to F in the pattern is directed because it would induce two new v-structures if it was reversed.

Algorithm 1 Routine for finding the pattern of a BN with structure A and node set N.

```
1: function OrderArcs(A, N)
2:     N_T ← topological ordering of N
3:     i ← 0
4:     while unordered arc ∈ A do
5:         y ← lowest ordered node in N_T that has an unordered arc incident into it
6:         x ← highest ordered node in N_T for which (x, y) is unordered
7:         label (x, y) with order i
8:         i ← i + 1
9:     end while
10:    return ordered arc set
11: end function

12: function FindPattern(A, N)
13:    A' ← OrderArcs(A, N)
14:    label all arcs in A' with unknown
15:    while arc labelled unknown ∈ A' do
16:        (x, y) ← lowest ordered arc in A' labelled unknown
17:        for (w, x) ∈ A' labelled compelled do
18:            if (w, y) ∉ A' then
19:                label all arcs incident into y with compelled
20:                goto line 14
21:            else
22:                label (w, y) with compelled
23:            end if
24:        end for
25:        if (z, y) ∈ A' s.t. z ≠ x and (z, x) ∉ A' then
26:            label all unknown arcs incident into y with compelled
27:        else
28:            label all unknown arcs incident into y with reversible
29:        end if
30:    end while
31:    return arcs labelled either compelled or reversible
32: end function
```

2.6 Statistical Distance

The result of an inference query is a probability distribution over the query variables. In the experiments reported in Chapter 4, we compare two query results in order to quantify the loss of inference accuracy when bounding the tree-width of Bayesian networks. Statistical distance in probability theory quantifies the distance between two probability distributions. Numerous measures for such quantification exist: *Hellinger distance* [28], *Bhattacharyya distance* [4] and *Kullback-Leibler divergence* [25] to name a few. We chose to focus our efforts on the Kullback-Leibler divergence (KL-divergence), often referred to as the *relative entropy*. KL-divergence is a measure of how one probability distribution differs from a reference distribution, and given two discrete probability distributions P and Q it is defined as

$$D_{KL}(P \parallel Q) = \sum_{x \in X} P(x) \log \left(\frac{P(x)}{Q(x)}\right)$$

for all x where $Q(x) = 0$ implies $P(x) = 0$. If $P(x) = 0$ for some x, the corresponding term in the summation is regarded as 0, because of the fact that $\lim_{x \to 0^+} x \log(x) = 0$. The KL-divergence is always non-negative, and is zero if and only if P and Q are equal. We can think of the $P(X)$ term as the weight, and the $\log \frac{P(x)}{Q(x)}$ term as the penalty. We see that as the difference between $P(x)$ and $Q(x)$ grows, the penalty increases. If $P(x)$ is high, we weight the penalty higher. Notice that the KL-divergence is not a proper metric, as it is not symmetric. By this, we mean that $D_{KL}(P \parallel Q)$ is not necessarily equal to $D_{KL}(Q \parallel P)$. We return to define a proper metric in the next section.

Absolute error and *root mean squared error (RMSE)* are two alternatives for comparing discrete probability distributions. In the absolute error measure, we simply calculate the distance between P and Q as $\sum_{x \in X} |P(x) - Q(x)|$. In the RMSE we calculate the distance between P and Q as

$$\sqrt{\frac{1}{C_X} \sum_{x \in X} \big(P(x) - Q(x)\big)^2},$$

where C_X denotes the cardinality of X. We have used KL-divergence, absolute error, and RMSE to assess inference quality in the experiments reported in Chapter 4.

2.7 Structural Hamming Distance

When bounding the tree-width of Bayesian networks, it can be interesting to measure how well the bounded networks approximate the original network. To measure the distance between two BN structures, we can use the Structural Hamming distance (SHD), a proper metric proposed by Tsamardinos et al. [36]. Versions of the SHD have been proposed by Acid and de Campos [1] and Perrier et al. [31]. To be a proper metric, a distance measure d on a set $\mathbf{X}$ must satisfy the following four properties for all x, y, and z in $\mathbf{X}$:

(i) $d(x, y) \geq 0$

(ii) $d(x, y) = 0 \Leftrightarrow x = y$

(iii) $d(x, y) = d(y, x)$

(iv) $d(x, z) \leq d(x, y) + d(y, z)$

The SHD between two networks β_1 and β_2 is defined as the sum of *added*, *missing*, and *incorrectly directed* edges. Added edges are edges that are present in β_1, and not present in β_2. Missing edges are edges that are present in β_2, and not present in β_1. Incorrectly directed edges are edges that are present in both β_1 and β_2, but with an opposite direction.

When comparing the structure of two BNs, there are typically two approaches to choose from. We may either compare two networks by their DAG structures or by their equivalence classes, represented by patterns. Empirical evaluations of the two approaches [14] have shown that their performance is similar, although the comparison of patterns produce higher results in general. This is explained by the fact that in patterns there are three types of edges ($x \rightarrow y$, $x \leftarrow y$ and $x - y$), which gives a lower probability of an edge being randomly correct.

Chapter 3

Dependency Strength

In order to predict the loss of accuracy when applying structural constraints, we introduce a measure of dependency strength. This measure will ideally tell us the strength of the dependencies in our network, and thus help us in predicting the impact of removing arcs. In this chapter, we formulate such a measure and justify why it is sensible. We look at a measure for individual arcs, as well as how to combine these individual strengths into the combined dependency strength of a BN. We start by formulating the desired properties of the measure. We then propose a measure of dependency strength in terms of likelihood and proceed to examine the properties of the measure and compare them to the desired properties.

3.1 Desired Properties

To motivate the formulation of a measure of dependency strength, we start by listing the desired properties of the measure.

1. **Stronger dependence produces a higher score**
   The score of an arc (X, Y) should be determined by how much Y is dependent on X. Recall from Section 2.1 that the dependence between two variables is quantified by their mutual information. A higher score should indicate stronger dependencies.

2. **Combination of arcs**
   If variable X is strongly dependent on a set of variables $\mathbf{S}$, then we want every arc from a variable in $\mathbf{S}$ to X to produce a high score.

3. **Independent of arcs not directed at the same node**
   The measure of an arc (X, Y) should be independent of any arc not directed at Y.

4. **Easy to compute**
   The measure should not be computationally exhaustive to compute, as it is intended to use in situations where the network structure is too complex for tractable inference.

5. **Independent of the size of the data set**
   The measure of an arc should be independent of the number of data points M in the data set, D.

6. **Non-negative**
   The measure should always produce a non-negative value, as there is no such thing as a negative dependence between two variables. We elaborate on this in Section 3.3.6.

3.2 Defining the Measure

Intuitively, the strength of an arc from X to Y w.r.t. to some network (A, θ) is *how much does knowing x help us to predict y*. To build on this intuition, we can examine the extreme cases of arc strength. If knowing x makes us certain of y, and not knowing x makes y indiscriminate, then we say that the arc (X, Y) has maximum strength. On the contrary, if knowing x does not affect our predictions of y at all, then we say that the arc (X, Y) has a minimum strength of zero.

We define the strength of an arc with help from the definition of likelihood of BNs from Section 2.5.2. Recall that the likelihood of a BN (A, θ) is a measure of how well the network fits the data. The idea is to compute and compare the likelihood, $\mathcal{L}$, of our data with and without arc (X, Y) as part of the structure.

If the likelihood is significantly larger when the arc is included, we know that the arc is important in fitting the network to the data, which in turn implies that the arc improves the

quality of our model and should be regarded as a strong arc. If the difference in likelihood is small, the arc is less important for the fit of the network and should be regarded as a weaker arc.

For convenience we use the log-likelihood, ℓ,

$$\ell(D|(A,\,\theta)) = \log \mathcal{L}(D|(A,\,\theta)) = \sum_{v\in N}\sum_{m=1}^{M} \log P(D_{mv}|D_{mA_v} : \theta_v),$$

where N is the node set of the BN. When measuring the strength of an arc $(X,\,Y)$, we consider the local likelihood of Y given data D and a BN $(A,\,\theta)$,

$$\ell_Y(D|((A,\,\theta))) = \sum_{m=1}^{M} \log P(D_{mY}|D_{mA_Y} : \theta_Y).$$

Let $(A^*,\,\theta^*)$ denote our BN without arc $(X,\,Y)$. We arrive at the following definition of the dependency strength of an arc $(X,\,Y)$:

$$\begin{aligned} Strength_{(X,\,Y)} &= \ell_Y(D|(A,\,\theta)) - \ell_Y(D|(A^*,\,\theta^*)) \\ &= \sum_{m=1}^{M} \log P(D_{mY}|D_{mA_Y} : \theta_Y) - \sum_{m=1}^{M} \log P(D_{mY}|D_{mA_Y^*} : \theta_Y^*). \end{aligned} \tag{3.1}$$

Let $\boldsymbol{\alpha}_{A_Y}$ denote all possible assignments of the variables in A_Y. Normalizing on the number of data points, M, and summing over the values of Y and all possible combinations of its parents, we get

$$Strength_{(X,\,Y)} = \sum_{a \,\in\, \boldsymbol{\alpha}_{A_Y}} \sum_{y \,\in\, Y} P(A_Y = a, Y = y) \log P(Y = y | A_Y = a) \; - \tag{3.2}$$

$$\sum_{a^* \,\in\, \boldsymbol{\alpha}_{A^*_Y}} \sum_{y \,\in\, Y} P(A^*_Y = a^*, Y = y) \log P(Y = y, A^*_Y = a^*)$$

$$= \sum_{a \,\in\, \boldsymbol{\alpha}_{A_Y}} \sum_{y \in Y} P(A_Y = a)\, \theta_{Y=y \,|\, A_Y=a} \log \theta_{Y=y|A_Y=a} \; - \tag{3.3}$$

$$\sum_{a^* \,\in\, \boldsymbol{\alpha}_{A^*_Y}} \sum_{y \,\in\, Y} P(A^*_Y = a^*, Y = y) \log P(Y = y | A^*_Y = a^*).$$

In Equation 3.3 we use $\theta_{Y=y|A_Y=a}$ to emphasize that these parameters are directly available in the parameter set of the network, θ.

3.2.1 Measuring the Strength of a Network

We express the total dependency strength of a BN as a function of local arc scores. Summing all the scores of the individual arcs would lead to larger networks producing higher scores in general. Instead, we normalize the sum by the number of variables in the network and compute the combined strength of a Bayesian network $(A,\, \theta)$ with $|N|$ nodes as

$$Strength_{\,(A,\,\theta)} = \sum_{(X,\,Y) \in A} \frac{Strength_{(X,\,Y)}}{|N|}.$$

One could experiment with alternative ways of combining local arc scores into the combined strength of a network. One idea is to do a weighted summation of arcs based on the in-degree of the child. Another idea is to play around with other normalizing factors such as the number of arcs or the average in-degree. Due to the limited time available for this thesis, this is left for future work.

3.3 Evaluating the Measure

In this section, the proposed measure is evaluated w.r.t. to the desired properties listed in Section 3.1.

3.3.1 Stronger Dependence Produces a Higher Score

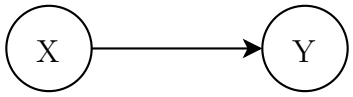


Figure 3.1: Example of a structure with two variables.

The first desired property states that a stronger dependence between variable X and Y produces a higher score. To evaluate this property, we consider the example network from Figure 3.1, and let X and Y be two binary variables. This network has four parameters: $\theta_{X=0}$, $\theta_{X=1}$, $\theta_{Y=0|X=0}$ and $\theta_{Y=0|X=1}$. Note that $\theta_{Y=1|X=0}$ and $\theta_{Y=1|X=1}$ are not needed, as they can be expressed using the other parameters. By Equation 3.2, we express the strength of arc (X, Y) as

$$
\begin{aligned}
Strength_{(X,Y)} &= \overbrace{\sum_{x=0}^{1}\sum_{y=0}^{1} \theta_x\, \theta_{y|x}\ \log\ \theta_{y|x}}^{\text{with arc } (X,Y)} - \overbrace{\sum_{y=0}^{1} P(y)\ \log\ P(y)}^{\text{without arc } (X,Y)} \\
&= \theta_{X=0}\, \theta_{Y=0|X=0}\ \log\left(\theta_{Y=0|X=0}\right) \\
&+ \theta_{X=0}\left(1-\theta_{Y=0|X=0}\right)\ \log\left(1-\theta_{Y=0|X=0}\right) \\
&+ \left(1-\theta_{X=0}\right)\theta_{Y=0|X=1}\ \log\left(\theta_{Y=0|X=1}\right) \\
&+ \left(1-\theta_{X=0}\right)\left(1-\theta_{Y=0|X=1}\right)\ \log\left(1-\theta_{Y=0|X=1}\right) \\
&- \Big(\left(\theta_{X=0}\,\theta_{Y=0|X=0}+\theta_{X=1}\,\theta_{Y=0|X=1}\right)\ \log\left(\theta_{X=0}\,\theta_{Y=0|X=0}+\theta_{X=1}\,\theta_{Y=0|X=1}\right) \\
&+ \left(1-\left(\theta_{X=0}\,\theta_{Y=0|X=0}+\theta_{X=1}\theta_{Y=0|X=1}\right)\right)\ \log\left(1-\theta_{X=0}\theta_{Y=0|X=0}+\theta_{X=1}\theta_{Y=0|X=1}\right)\Big).
\end{aligned}
$$

As analyzing the behavior of this equation is challenging, we add the simplifications listed below:

- $\theta_{X=0} = \theta_{X=1} = 0.5$
- $\theta_{Y=0|X=0} = \theta_{Y=1|X=1} = \theta$
- $\theta_{Y=1|X=0} = \theta_{Y=0|X=1} = 1 - \theta$
- $0 \leq \theta \leq 1$

The simplified expression for the strength of arc $(X,\ Y)$ becomes:

$$\begin{aligned} Strength_{(X,\,Y)} &= 0.5\,\theta\ \log\ \theta + 0.5\,(1-\theta)\ \log\,(1-\theta) \\ &\quad + 0.5\,(1-\theta)\ \log\,(1-\theta) + 0.5\,\theta\ \log\ \theta \\ &\quad - \Big(0.5\ \log\,(0.5) + 0.5\ \log\,(0.5)\Big) \\ &= \theta\ \log\ \theta + (1-\theta)\ \log\,(1-\theta) - \log\,(0.5). \end{aligned} \tag{3.4}$$

With these simplifications, it becomes clear that the value of θ determines the dependence of Y on X. By uniformly distributing X, we ensure that the log-likelihood of Y without arc $(X,\ Y)$ is $\log(0.5)$. This is desirable as it allows the strength of the arc to vary from minimum to maximum strength. Equation 3.4 is plotted in Figure 3.2. The simplification $\theta_{Y=0|X=0} = \theta_{Y=1|X=1} = \theta$ shows that as θ approaches 1, the values of X and Y are likely to be equal and as θ approaches 0 the values of X and Y are likely to be opposite. In both of these cases, we see from the plot that arc $(X,\ Y)$ approaches its maximum strength. As θ approaches 0.5, the information about Y provided by observing X approaches 0, and the strength of arc $(X,\ Y)$ approaches 0. This concurs with our expectations and argues that a stronger dependence between X and Y produces a higher score for arc $(X,\ Y)$.

Recall from Section 2.1 that the dependency between X on Y can be quantified by the mutual information between X and Y, $MI(X,\ Y) = \sum_{x,y} P(x,\ y) \log \frac{P(x,y)}{P(x)P(y)}$. In the case where X is the only parent of Y, Equation 3.2 can be manipulated to show that it is exactly the same as the mutual information between X and Y. This is a pleasing result, as the mutual information between X and Y is a quantification of dependence, and we want the

measure to produce a higher score for stronger dependencies. The strength of an arc in the case of a single parent relationship is exactly the same as Nicholson and Jitnah [27] proposed in their work, although we were not aware of this similarity when we started to work on our idea based on log-likelihood difference.

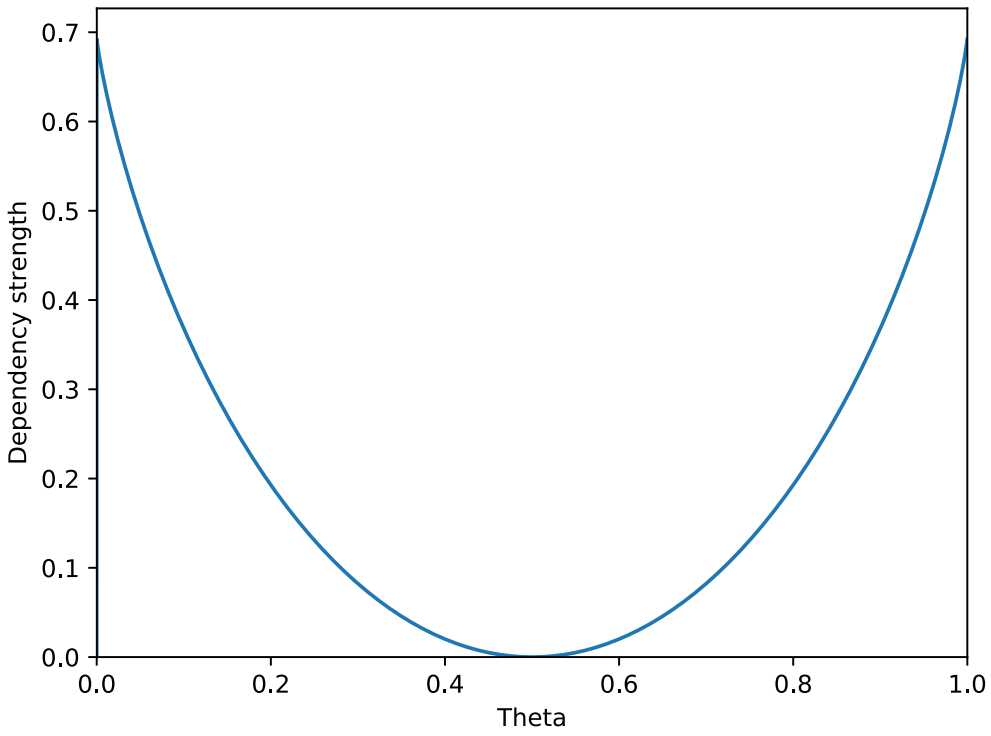


Figure 3.2: Plot of Equation 3.4.

3.3.2 Combination of arcs

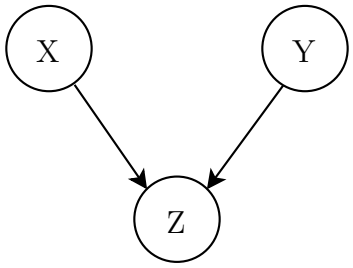


Figure 3.3: A simple Bayesian network structure of three variables.

Measuring the dependency strength of an arc, we must consider the fact that a variable V may be dependent on a set, $\mathbf{S}$, of variables. Even if V is weakly dependent on each of the individual variables in $\mathbf{S}$, we still want the measure to produce a high score for arcs from a variable in $\mathbf{S}$ to V. To evaluate if the measure holds this property, we consider an example network with three binary variables, X, Y and Z and the structure shown in Figure 3.3. As in Section 3.3.1, we add simplifications for the sake of analyzing the behavior. We say that the following holds for our BN:

- $\theta_{Z=0|Y=0} = \theta_{Z=0|Y=1} = \theta_{Z=1|Y=0} = \theta_{Z=1|Y=1} = 0.5$
- $\theta_{Z=0|X=0,Y=0} = \theta_{Z=1|X=0,Y=1} = \theta_{Z=1|X=1,Y=0} = \theta_{Z=0|X=1,Y=1} = \theta$
- $\theta_{Z=1|X=0,Y=0} = \theta_{Z=0|X=0,Y=1} = \theta_{Z=0|X=1,Y=0} = \theta_{Z=1|X=1,Y=1} = 1 - \theta$
- $0 \leq \theta \leq 1$

If we calculate the dependency strength of (X, Z) with these simplifications, we arrive at

$$
\begin{aligned}
Strength_{(X,Z)} &= \overbrace{\sum_{x=0}^{1}\sum_{z=0}^{1} P(x,z)\sum_{y=0}^{1}\theta_{z|x,y}\log\theta_{z|x,y}}^{\text{with arc (X, Z)}} - \overbrace{\left(\sum_{y=0}^{1}\sum_{z=0}^{1} P(z,y)\log P(z|y)\right)}^{\text{without arc (X, Z)}} \\
&= 4 * \frac{1}{4}\Big(\theta\ \log\ \theta + (1-\theta)\ \log\ (1-\theta)\Big) - 4 * \frac{1}{4}\log(0.5) \\
&= \theta\log\theta + (1-\theta)\ \log\ (1-\theta) - \log(0.5)
\end{aligned}
$$

Notice that this simplified result is equal to the result from Section 3.3.1. This indicates that if a variable V has a strong dependence on a set of variables $\mathbf{S}$, then all arcs from a node

in S to V will produce a high score. However, in the case of multiple parents, we are not able to make a direct connection to mutual information. The *multivariate mutual information (MMI)*, that is, the mutual information between three or more variables, is a less intuitive measure and a poorly understood concept of information theory. Timme et al. [35] defines the MMI between a variable Y and a set of variables $\mathbf{S} = \{X_1, ..., X_n\}$ as

$$MMI(Y, S) = \sum_{y \in Y,\, x_1 \in X_1, .., x_n \in X_n} P(y, x_1, ..., x_n) \log \frac{P(y, x_1, ..., x_n)}{P(y)P(x1, ..., x_n)},$$

by treating $\mathbf{S}$ as a single vector-valued variable. This is not equivalent to the measure defined in Equation 3.2 and thus, in the case where Y has multiple parents, we cannot use the mutual information argument that was used in Section 3.3.1.

The measure of arc (X, Y) where Y has multiple parents is similar to that of Nicholson and Jitnah [27], although not equivalent. In the case of multiple parents, they define the weight of an arc (X, Y) as

$$\omega(X, Y) = \sum_{z \in Z} P(z) \sum_{x \in X} P(x) \sum_{y \in Y} P(y|x, z) \log \frac{P(y|x, z)}{P(y|z)},$$

where Z is the set $A_Z \setminus \{X\}$, assuming that X and Z are independent. In our definition of arc strength (Equation 3.2) we do not consider these variables independent, and therefore we arrive at a slightly different equation.

3.3.3 Independent of Arcs not Directed at the Same Node

The third desired property states that the score of arc (X, Y) should be independent of any arc not directed at Y. This property follows directly from Equation 3.2, as the parameters of Y are independent of arcs not directed at Y and therefore cannot be affected by any other arc.

3.3.4 Easy to Compute

The fourth desired property states that the measure should not be computationally exhaustive. This is a desirable property because we intend to use the measure in situations where the structure of the original network is too complex for inference to be tractable. Unfortunately, computing the dependency strength of a network using our defined measure is a computationally heavy task. The complexity arises from the fact that some of the probabilities used are not present in the network parameters. When an arc (X, Y) is removed from the network, the probability of Y given its other parents is not available in the network. As a consequence, these probabilities must be computed through inference. As shown in Section 2.5.5, inferring conditional probabilities is very computationally exhausting.

A solution to this problem could be to calculate the missing probabilities by averaging over local CPDs. This would require far less computations, but would lead to less accurate results. Nicholson and Jitnah [27] used this strategy to overcome the problem, and achieved satisfying results. Experimenting with versions of approximate inference as a combination of the two approaches is left for future work.

3.3.5 Independent of the Size of the Data Set

The fifth desired property states that the measure should be independent of the number of data points, M, in the data set D. We can see that this property holds directly from the generalized equation of dependency strength from Equation 3.2, as M is not a factor.

3.3.6 Non-Negative

The sixth desired property states that the strength of an arc (X, Y) should always be non-negative. This is desirable as we can never become more uncertain about the value of a variable Y by observing some other variable X. The arc from X to Y can never negatively impact our predictions. In the worst case, observing X does not provide any valuable information for predicting Y, and the strength of the arc (X, Y) should be zero. Following the same logic, the log-likelihood of a Bayesian network can never be increased by removing an arc and thus Equation 3.1 will always produce a non-negative result.

Chapter 4

Predicting Loss of Inference Accuracy

One of the main challenges with Bayesian networks is the complexity of inference. If the structure of the network is too complex, that is, if its tree-width is too large, inference is not tractable. Recall from Chapter 1 that the exact inference problem resides in the NP-hard complexity class and that it is fixed-parameter tractable w.r.t. the tree-width of the BN structure. If we learn an unbounded BN only to realize that inference is intractable, we must either resort to approximate inference or to learning a new, bounded network. If we decide to bound the tree-width of our network, we can assume that these simplifications will have a negative impact on the inference quality, as we may no longer be able to accurately represent the probability distribution. However, predicting how much the inference results will decay is no easy task. Since we do not have access to inference results from the unbounded network, we cannot compare the results directly. Our proposed solution to this problem is to use the measure defined in Chapter 3 as a predictor of loss of inference accuracy. Our hypothesis is that if we bound the tree-width of a Bayesian network with a high dependency strength, then we are more likely to remove important arcs, and the simplifications will have a larger impact on the inference quality. More specifically, our hypothesis is that there will be a positive correlation between the dependency strength of the original network and the decay in inference accuracy when bounding the tree-width of the network.

In this chapter, we evaluate the dependency strength measure as a predictor of loss of accuracy when learning bounded tree-width networks. Results from an experiment on five networks of different dependency strengths are reported. We start by giving an overview and

describing the implementation details. We proceed to present the results of the experiments. Finally, we discuss the weaknesses of the experiment and present some slightly modified results.

4.1 Experiment Design

In this section, we outline the design of the conducted experiment. The purpose of the experiment was to evaluate how well the dependency strength of a network predicts the loss of accuracy when learning bounded tree-width networks. An overview of the experiment setup is given in Figure 4.1 and we will now give a brief introduction to each step.

Five networks from the bnlearn repository[1] were used, which we will refer to as the *data-generating networks*. From each of these networks we generated 10 000 random samples. These samples were used to produce local scores with GOBNILP. The scores were provided as input to the TWILP-software in order to learn four bounded tree-width structures ranging from a tree-width of 1 to a tree-width of 4. The Bayesian estimator discussed in Section 2.5.3 was then used to learn the parameters of the bounded networks. We proceeded to run 2500 queries on the data-generating network and each of the bounded networks, and the resulting distributions were compared using the KL-divergence described in Section 2.6. Finally, we plotted the KL-divergence against the dependency strength of the original data-generating network.

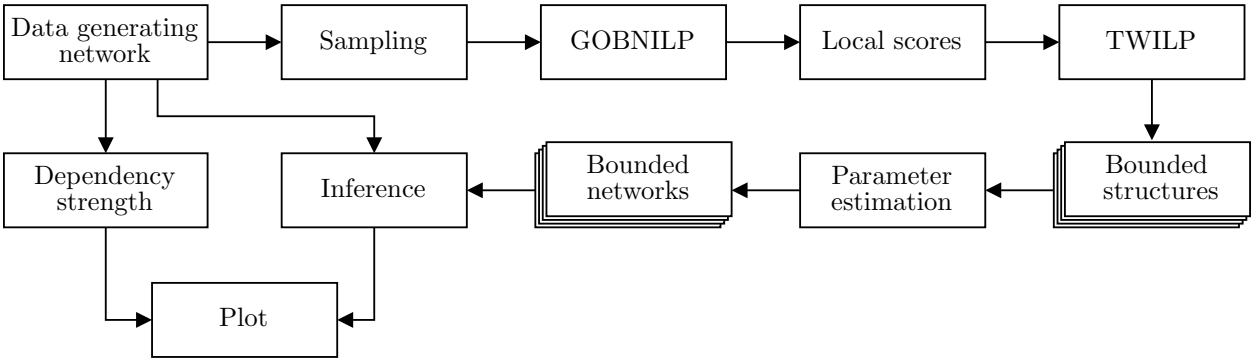


Figure 4.1: Overview of the experiment setup.

[1] http://www.bnlearn.com/bnrepository/

4.2 Implementation Details

Hardware

All the tests were run on a dual CPU (Intel(R) Xeon(R) CPU E5-2699 2.30GHz) SYS-2028 GR-TR with 256Gb RAM.

Software Dependencies

Globally Optimal Bayesian Network learning using Integer Linear Programming (GOBNILP)[2] is a C-program for exact structure learning of Bayesian networks written by Cussens and Bartlett [3]. Given a complete data set of discrete random variables GOBNILP can be used to find the optimal structure of a BN. It can also be used to produce *local scores*, which we address later in this section. A local score is the score of one possible parent set of a node. There are several scoring functions that may be used to produce these scores, as discussed in Section 2.5.4. The optimization problem that GOBNILP solves is finding the best combination of parent sets with the constraint that they must form a DAG. It does so by converting the problem to an ILP-problem and solving it using SCIP[3] [19, 20]. We used GOBNILP version 1.6.2 along with SCIP Optimization suite version 3.2.0.

TWILP[4] is a python software developed by Parviainen and Farahani [29] for learning bounded Bayesian networks. It can be used to bound both the vertex cover number and the tree-width of a BN, but we will only use it for the latter. TWILP solves the bounding problem by reducing it to an ILP-problem and solving it using the IBM ILOG CPLEX optimizer (CPLEX)[5]. We used TWILP version 1.1 with CPLEX version 12.8.

pgmpy[6] is an open source python library for working with PGMs, developed by Ankan and Panda [2]. It provides a wide range of functionality for working with Bayesian networks, including structure learning, parameter estimation, sampling, and inference. We used pgmpy version 0.1.2 for sampling, parameter estimation, and inference.

[2]https://www.cs.york.ac.uk/aig/sw/gobnilp/
[3]https://scip.zib.de/
[4]https://bitbucket.org/twilp/twilp/
[5]https://www.ibm.com/analytics/cplex-optimizer
[6]https://github.com/pgmpy/pgmpy/

Data-generating Networks

The data-generating networks were downloaded from the bnlearn repository in the *BIF-format*[7] [22]. The five networks that were used are listed in Table 4.1, along with the number of nodes, arcs, and parameters for each network. These networks were selected as they are small and simple enough that exact inference can be performed on the data-generating network in a reasonable amount of time.

Network	Nodes	Arcs	Parameters
Child	20	25	230
Alarm	37	46	509
Insurance	27	52	984
Win95pts	76	112	574
Hepar II	70	123	1453

Table 4.1: The five networks that were used in the experiment and their properties.

Sampling

We used pgmpy to generate samples using the forward sampling strategy. This sampling strategy is implemented in the following steps:

(i) Topologically order the nodes of the network.

(ii) In topological order, sample the value x_i from the distribution $P(X_i|A_i)$. The topological ordering will ensure that values for the variables in A_i are always accessible in $\{x_0, \dots, x_{i-1}\}$.

(iii) Repeat step *ii* for each sample.

The samples were stored as a *.dat*-file, a format that is accepted by GOBNILP. The first line of this file contains a space-separated list of the names of the n variables in the data set. The second line contains n space-separated integers, the i-th integer denoting the cardinality of the i-th variable in the first line. Then follows one line for each data point containing a space-separated list of n values, the i-th value denoting the value of the i-th variable. These values can be both strings and integers.

[7] http://www.cs.washington.edu/dm/vfml/appendixes/bif.htm

Generating Local Scores

The generated samples were given as input to GOBNILP to generate local scores. This was done by running the shell command *$ gobnilp -x samples.dat*. The $-x$ flag tells GOBNILP to exit before solving the optimization problem and output the local scores. We set the maximum parent set size to 3 by setting the parameter *gobnilp/scoring/palim* in GOBNILPs parameter file called gobnilp.set. For GOBNILP to output the local scores, the parameter *gobnilp/outputfile/scores* must also be set. For more information about how to run GOBNILP, the reader is referred to the manual [12].

The local scores were stored in the following format: The first line contains an integer, n, the number of variables in our BN. Then follow n sections, one per variable. Each section starts with a line containing the name of the variable and the number of candidate parent sets, p, for this variable. The remaining p lines in the section describe the candidate parent sets: First a real-valued score of the set, then the size of the parent set, and finally a space-separated list of the parents in the set.

Learning Bounded Tree-Width Structures

The local scores were provided as input to the TWILP-software. TWILP was run with the following command: *$ python twilp/twilp.py -f parent_set.scores -o output/path/ -t tree_width -p max_parents -r 43200 -s 300 -m 1*. The $-f$ flag specifies the path to the file containing local scores, the $-o$ flag specifies the output path of the resulting bounded structures, the $-t$ and $-p$ flags specify the tree-width bound and maximum number of parents per node, the $-r$ flag specifies the total amount of time in seconds that TWILP is allowed to run. The $-s$ flag specifies the amount of time TWILP is allowed for each sub-IP. The $-m$ flag determines the graph parameter to use as a bound, where 1 corresponds to tree-width.

TWILP produces three types of output. The files *y.gml and *z.gml contain the learned y-graph and z-graph, respectively. The *.result file contains information about the solution and the elimination order. For the purpose of this experiment, we only used the z-graph output, which contains the learned bounded structure.

Parameter Estimation

The learned structure and the generated samples were provided to pgmpy in order to learn the parameters. A simple wrapper was developed to utilize the Bayesian estimator from pgmpy. We used a BDeu prior type as discussed in Section 2.5.4, with an equivalent sample size of 5. With the parameters in place, the bounded tree-width networks were complete and they were written to files in the BIF-format.

Inference

In the next step of the experiment, exact inference queries were performed on both the data-generating and the bounded tree-width networks. Recall that performing an inference query means calculating the conditional probability distribution of some query variable X given a set of evidence $\mathbf{E}$ with fixed values $\mathbf{e}$, or more formally, specifying the probability distribution $P(X \mid \mathbf{E} = \mathbf{e})$.

We started by generating the queries, $(X, \mathbf{E}, \mathbf{e})$, dividing them into five categories of 500 queries each: Random queries, marginal queries, parent-child queries, weighted maximum distance queries and weighted minimum distance queries. The following strategies were used to generate queries for the respective categories:

- **Random queries** were generated by picking a random node q as the query variable, then selecting from 1 to 5 evidence variables by using the routine outlined in Algorithm 2.
- **Marginal queries** were generated by picking a random node q as the query variable, and using the empty set as evidence.
- **Parent-child queries** were generated by picking a random non-root node c, and then randomly selecting one node from its parents as evidence variable. The evidence variable was then given a random value drawn from its set of possible values.
- **Weighted maximum distance queries** were generated by picking a random node q as the query variable, and then selecting evidence variables based on the distance from q in the skeleton of the structure. Recall that the skeleton of a DAG is an undirected graph with the same structure, where the direction of arcs are omitted. Algorithm 3

outlines the algorithm used to randomly draw evidence variables weighted by their shortest path from q, favoring nodes further away. The probability of drawing variable v is $\frac{d_v^2}{\sum_{u \in N} d_u^2}$, where d_v is the shortest path from q to v.

- **Weighted minimum distance queries** were generated using the same procedure as the weighted maximum queries, but favoring nodes closer to q. The probability of drawing variable v is $\frac{(d_{max}+1-d_v)^2}{\sum_{u \in N} d_u^2}$, where d_{max} is the longest shortest path from q to any other variable in the skeleton graph of the structure.

All queries were performed on each of the bounded networks and on the data-generating network. We performed exact inference with algorithms provided by the pgmpy library. More specifically, we used the Variable elimination algorithm described in Section 2.5.5. Recall that a crucial part of this algorithm is the order of elimination of variables. In this experiment, the elimination order was decided using the *weighted min fill* heuristic. This heuristic tries to minimize the number of fill edges induced by the elimination order. Empirical results provided by Fishelson and Geiger [17] show that this heuristic outperforms other greedy approaches in selecting the best order. An algorithm for selecting the elimination order based on the weighted min fill heuristic is implemented by the pgmpy library.

Algorithm 2 Routine for selecting random evidence.

```
1: function RandomEvidence(N, q)
2:     E ← empty set
3:     n ← random(1, 5)                          ▷ random number from 1 to 5

4:     while size(E) < n do
5:         e ← random(N)                         ▷ random variable from N
6:         if e ∉ E and e ≠ q then
7:             e' ← random Val(E)                ▷ random value from values of E
8:             E ← E ∪ (e, e')
9:         end if
10:    end while
11:    return E
12: end function
```

Algorithm 3 Routine for randomly selecting evidence variables weighted on maximum shortest distance from q.

```
1: function WeightedRandomMaxDistanceEvidence(N, A, q)
2:     E ← empty set
3:     n ← random(1, 5)                          ▷ random number from 1 to 5
4:     A' ← skeleton(A)                          ▷ skeleton of the structure A
5:     sp ← shortest_paths(A', q)                ▷ length of shortest path from q to all nodes in A'
6:     arr ← []

7:     for v ∈ N do
8:         arr ← arr ∪ repeat(v, sp[v]^2)        ▷ add v to arr sp[v]^2 times
9:     end for

10:    while size(E) < n do
11:        e ← random variable from arr
12:        if e ∉ E then
13:            e' ← random Val(e)                ▷ select random from values of e
14:            E ← E ∪ (e, e')
15:        end if
16:    end while
17:    return E
18: end function
```

Dependency Strength

The next step of the experiment was to determine the dependency strength of the data-generating networks. Recall that we define the strength of an arc $(X,\, Y)$ as the difference in the log-likelihood of the BN with and without arc $(X,\, Y)$,

$$\begin{aligned} Strength_{(X,\,Y)} = & \sum_{a \,\in\, \boldsymbol{\alpha}_{A_Y}} \sum_{y \,\in\, Y} P(A_Y = a, Y = y) \log P(Y = y | A_Y = a) \; - \\ & \sum_{a^* \,\in\, \boldsymbol{\alpha}_{A^*_Y}} \sum_{y \,\in\, Y} P(A^*_Y = a^*, Y = y) \log P(Y = y, A^*_Y = a^*) \\ = & \sum_{a \,\in\, \boldsymbol{\alpha}_{A_Y}} \sum_{y \in Y} P(A_Y = a)\, \theta_{Y=y \,|\, A_Y=a} \log \theta_{Y=y|A_Y=a} \; - \\ & \sum_{a^* \,\in\, \boldsymbol{\alpha}_{A^*_Y}} \sum_{y \,\in\, Y} P(A^*_Y = a^*, Y = y) \log P(Y = y | A^*_Y = a^*), \end{aligned}$$

where A_Y is the parent set of Y, A^* is the structure of BN without arc $(X,\, Y)$, $\boldsymbol{\alpha}_{A_Y}$ is the set of all assignments of the variables in A_Y and θ is the set of parameters for the BN.

The general strategy for calculating the dependency strength of a network is outlined in Algorithm 4. We start by iterating over all nodes v in the node set N. For every node, we calculate the local log-likelihood l_v of that node given parameters θ and structure A. We then iterate over all arcs $\{u,\, v\}$ directed at v, and create an arc-set A^* by removing $\{u,\, v\}$ from A. We proceed to calculate the log-likelihood l^*_v of v given parameters θ and structure A^*, and finally add the difference between l_v and l^*_v to the total dependency strength. In the sub-routine for calculating likelihood, we start by iterating over all possible assignments of v and its parents A_v. For every assignment a, if the probability $P(a)$ is in the parameter set θ, we add $\theta_a \log \theta_a$ to the total likelihood result. If $P(a)$ is not in the parameter set θ, we must perform an inference query $P(v|A_v : \theta)$ to obtain the parameter before adding $\theta_a \log \theta_a$ to the total likelihood result.

Algorithm 4 Routine for calculating the dependency strength of a network.

```
1: function CalculateDependencyStrength(N, A, θ)
2:     result ← 0
3:     for v ∈ N do
4:         l_v ← CalculateLogLikelihood(v, A, θ)
5:         for {u, v} ∈ A do
6:             A* ← A \ {u, v}
7:             l*_v ← CalculateLogLikelihood(v, A*, θ)
8:             result ← result + (l_v − l*_v)
9:         end for
10:    end for
11:    return result
12: end function

13: function CalculateLogLikelihood(v, A, θ)
14:    l ← 0
15:    for a ∈ α_{A_v} do          ▷ iterate over all possible assignments of parents of v
16:        if P(a) ∈ θ then          ▷ if parameter exists in parameter set θ
17:            l ← l + (θ_a log θ_a)
18:        else
19:            θ_a ← query(v, A_v, θ)          ▷ perform inference to obtain parameter θ_a
20:            l ← l + (θ_a log θ_a)
21:        end if
22:    end for
23:    return l
24: end function
```

4.3 Results

In this section, we describe the results of the conducted experiment. The purpose of the experiment was to lay ground for discussions about whether or not the dependency strength measure from Chapter 3 is a good predictor of loss of accuracy when bounding the tree-width of Bayesian networks. More specifically, we wanted to examine the correlation between the dependency strength of a network and the KL-divergence between the inference results from the unbounded and bounded networks. We start by presenting the results of the inference queries performed on the data-generating and bounded networks. We proceed to examine the correlations and discuss the weaknesses of the experiment. Finally, we present modified results where the most complex network of the experiment is excluded.

The results from the experiment are visualized in Figure 4.2 by plotting dependency strength versus the KL-divergence between the query results. Notice that all points on the x-axis have been evenly spread out to prevent the Child and Alarm networks to appear too close to each other and cause confusion. The blue line shows the KL-divergence between inference results on the data-generating networks and the bounded networks of tree-width 1. Similarly, the orange, green, and red lines show the KL-divergence between inference results on the data-generating networks and bounded networks of tree-width 2, 3, and 4, respectively. The purple line shows the KL-divergence between inference results from the data-generating network and the unbounded network learned by GOBNILP. All the unbounded networks were learned with a maximum parent set size of 3, except for Win95pts which had to be bounded to a maximum parent set size of 2 due to its complexity. The size of the markers on each data point represent the Structural Hamming distance (SHD) between the learned network and the data-generating network, obtained by comparing patterns.

For the network with the weakest dependencies, Hepar II, the KL-divergence is close to zero for all tree-width bounds, even though the SHD is at a considerable size. For the second weakest network, Win95pts, there is a significant decay in inference results for both the bounded and the unbounded networks. We also observe a considerable SHD for these networks. For the Child network, all the bounded networks except for that of tree-width 1 resemble the data-generating network perfectly (the SHD is zero). From this we can conclude that the original Child network has a tree-width of 2. If the structures are identical, the only difference between the data-generating and bounded network is the parameters. As a

consequence, the inference results are close to equal and the KL-divergence is close to zero. The dependency strength of the Alarm network is close to the strength of the Child network, but the decay in inference results is much higher. There is also a considerable SHD between the data-generating and the bounded tree-width Alarm networks. The Insurance network has the strongest dependencies according to our measure, and the bounded network of tree-width 1 has the highest KL-divergence in the experiment. There is also a large spread in KL-divergence for the different Insurance networks.

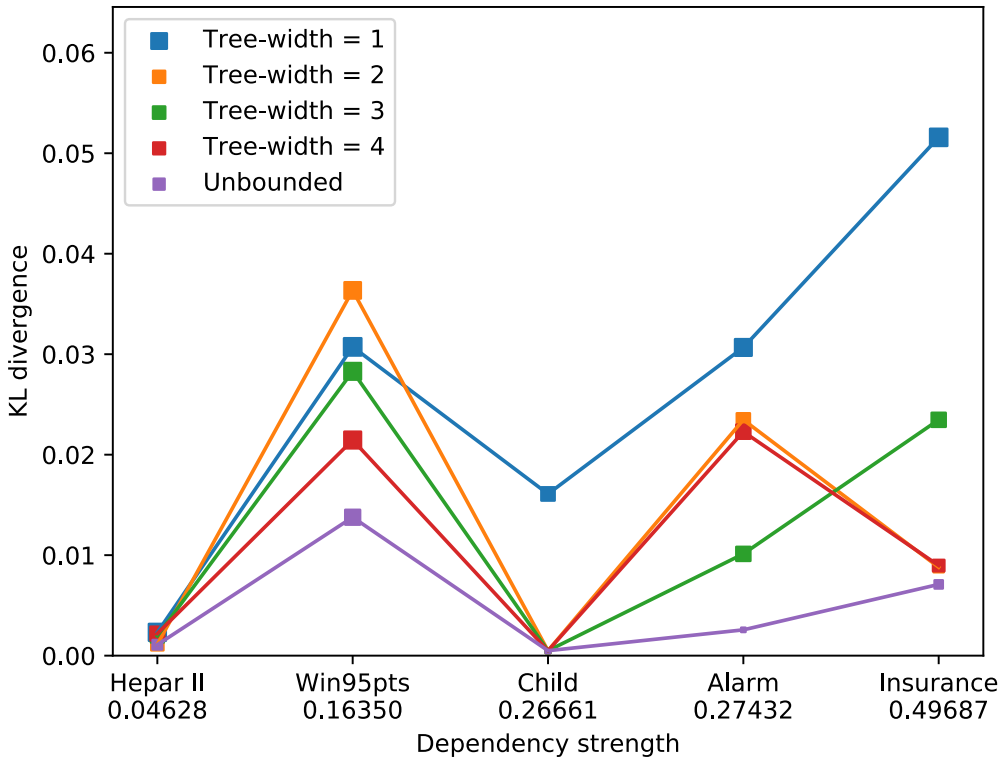


Figure 4.2: Plot of dependency strength versus KL-divergence.

An interesting observation from Figure 4.2 is that for some of the networks, a network of a lower tree-width bound outperforms one of a higher tree-width bound. An example of this is the Win95pts network: We observe that the bounded network of tree-width 1 outperforms the bounded network of tree-width 2. This should not be possible, as the network of higher tree-width bound can always replicate the lower bounded network to achieve equal inference results. These observations suggest that TWILP did not converge on an optimal solution. As

the search space grows with the tree-width bound, TWILP managed to find a more optimal network within the time constraint when the bound was lower. Because of this, we can lower the KL-divergence result of queries performed on bounded networks that were outperformed by lower networks to the result of the lower network. We have plotted the results with these adjustments in Figure 4.3. As we consider this a more accurate plot, we will use it as a base for the discussions to follow.

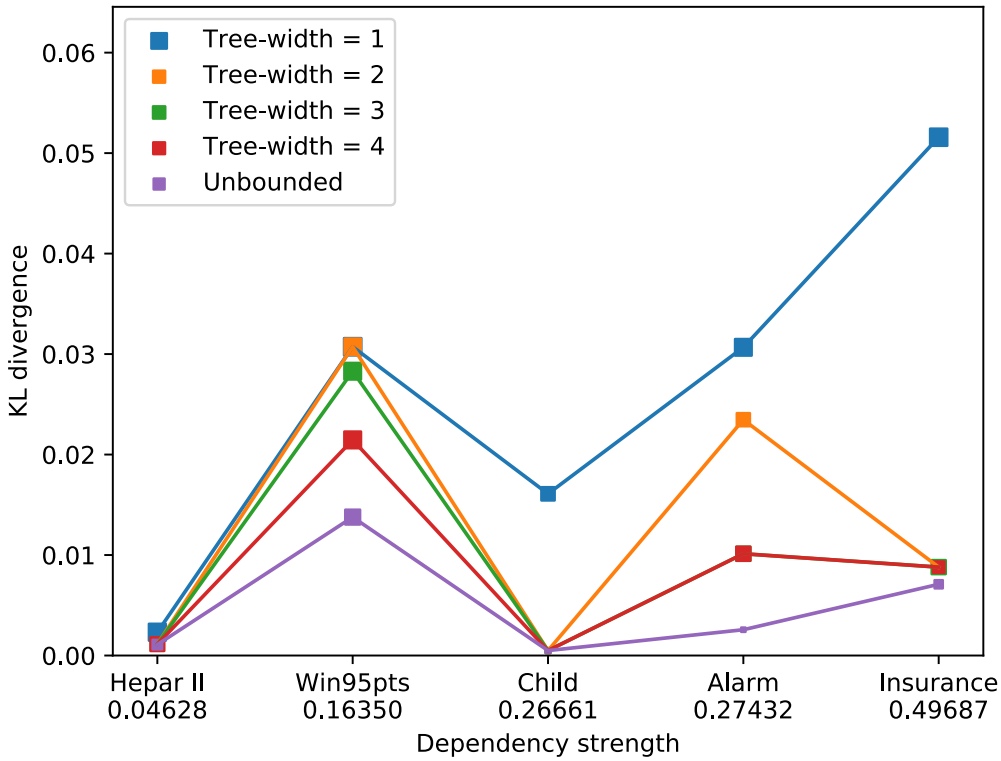
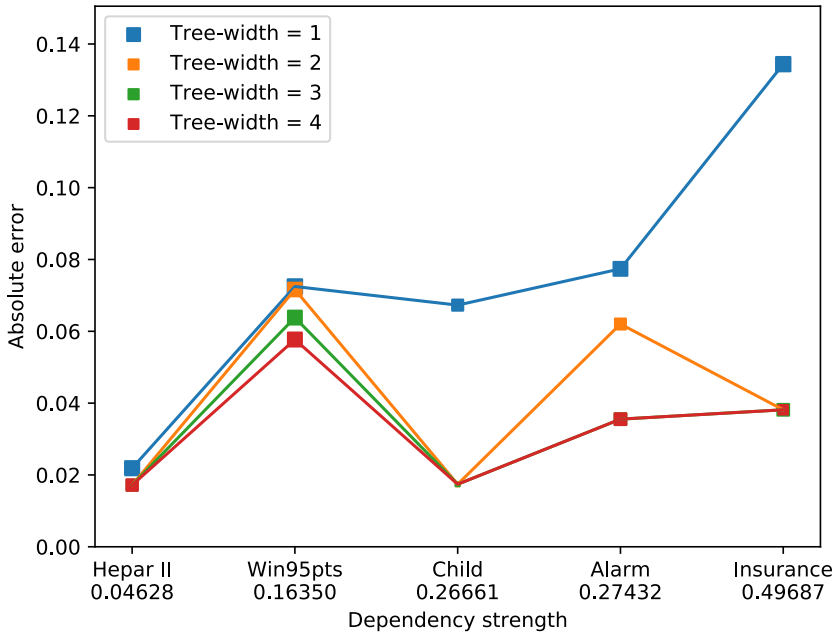
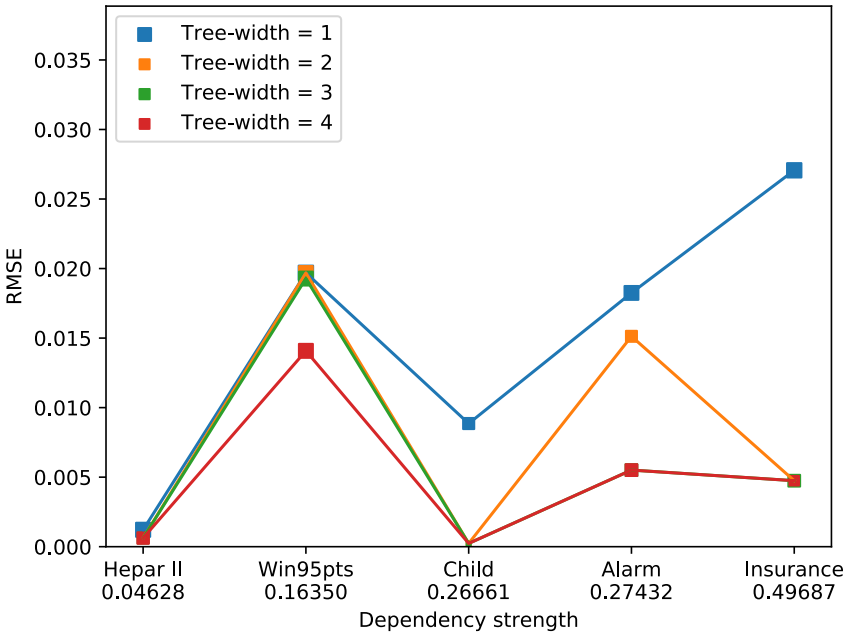


Figure 4.3: Adjusted plot of dependency strength versus KL-divergence.

To examine if the results are different when using the absolute error or RMSE from Section 2.6, we have plotted both of these versus the dependency strength in Figure 4.4. Both measures provide trends similar to those achieved by comparing KL-divergence. Therefore, in further discussions we will focus on the results form the KL-divergence plot.



(a)



(b)

Figure 4.4: (a) Plot of dependency strength versus absolute error and (b) plot of dependency strength versus root mean squared error.

Returning to our initial hypothesis, we examine the correlation between the dependency strength of the data-generating network and the KL-divergence between the inference results. The correlation coefficients between the dependency strength of the network and the KL-divergence for each of the different tree-width bounds are listed in Table 4.2. For the bounded networks of tree-width 1, the correlation coefficient is 0.813, which is generally considered a strong positive correlation. For the bounded networks of tree-width 2 and 4, the correlation is close to zero. For the bounded networks of tree-width 3, there is a weak positive correlation. The combined correlation for all the data points shows a weak positive correlation.

Tree-width	1	2	3	4	Combined
Correlation	0.813	-0.024	0.283	0.121	0.303

Table 4.2: Correlation between the tree-width bound and the KL-divergence of inference results.

An obvious weakness in this experiment is the number of networks used in the tests. With only five networks evaluated, the conclusions must be seen in light of the sample size. A small number of data points also make the results more sensitive to outliers or inaccuracies in the test. We were not able to deal with this problem in the limited time of this thesis, as there were problems with computational resources and complications with software dependencies. Some of the networks were too complex to either produce local scores from or to perform exact inference on. Some of the networks caused the software dependencies to crash, and we were not able to identify the root causes.

Another weakness in this experiment lies in the usage of TWILP. TWILP requires a lot of computational resources to converge on the optimal solution for the bounded tree-width networks. As both time and computational resources were limited, we had to set a maximum time limit for the learning algorithm. Since TWILP uses an anytime algorithm we could still get the best result so far. As mentioned in Section 4.2, TWILP was given 12 hours of computation time for each problem. For the Child network and all of the networks with a bounded tree-width of 1, TWILP was able to find the provably optimal network given the constraints. For most of the networks, however, TWILP was not able to converge on the optimal solution. This can be seen in the original results, as there are several examples where a lower tree-width bounded network outperforms a higher tree-width bounded network. It is practically impossible to say how much of the decay in inference results is caused by

TWILP not converging towards an optimal solution, and how much is caused by the tree-width bound. A related weakness is that since the networks are of different complexity, some networks will be further away from an optimal solution than others. As a result, the comparison between networks of different complexity is not fair.

The Win95pts network was the most complex network used in the test. When running GOBNILP with a maximum parent set size of 3 on a data set of 10 000 entries, it did not finish in 7 days. The scoring process was aborted and had to be re-run with a lower maximum parent set size of 2 in order to reduce the search space. With the updated parent size limit, the algorithm ran for only a short amount of time, as with the other data-generating networks. This indicates that the Win95pts is a much more complex network and we can speculate whether or not TWILP was able to approximate the bounded tree-width versions of this network as accurate as the rest. Under the assumption that the Win95pts network is too complex to provide a fair comparison with the rest of the networks used in the experiment, we exclude it from the test and inspect the results. The results without including the Win95pts network are plotted in Figure 4.5. From this plot, it appears that there is a stronger correlation between the dependency strength and the KL-divergence. The correlation coefficients are listed in Table 4.3 and we can see that tree-width 1 has a strong positive correlation, that tree-width 3 and 4 have moderate positive correlations and that tree-width 2 has a weak positive correlation. The combined correlation is now moderate with a correlation coefficient of 0.54. Naturally one should be very careful discussing correlations with only four data points for each plot, but we can use this as an indication of a positive correlation.

Tree-width	1	2	3	4	Combined
Correlation	0.909	0.397	0.632	0.632	0.476

Table 4.3: Correlation between the tree-width bound and the KL-divergence of inference results, excluding the Win95pts network.

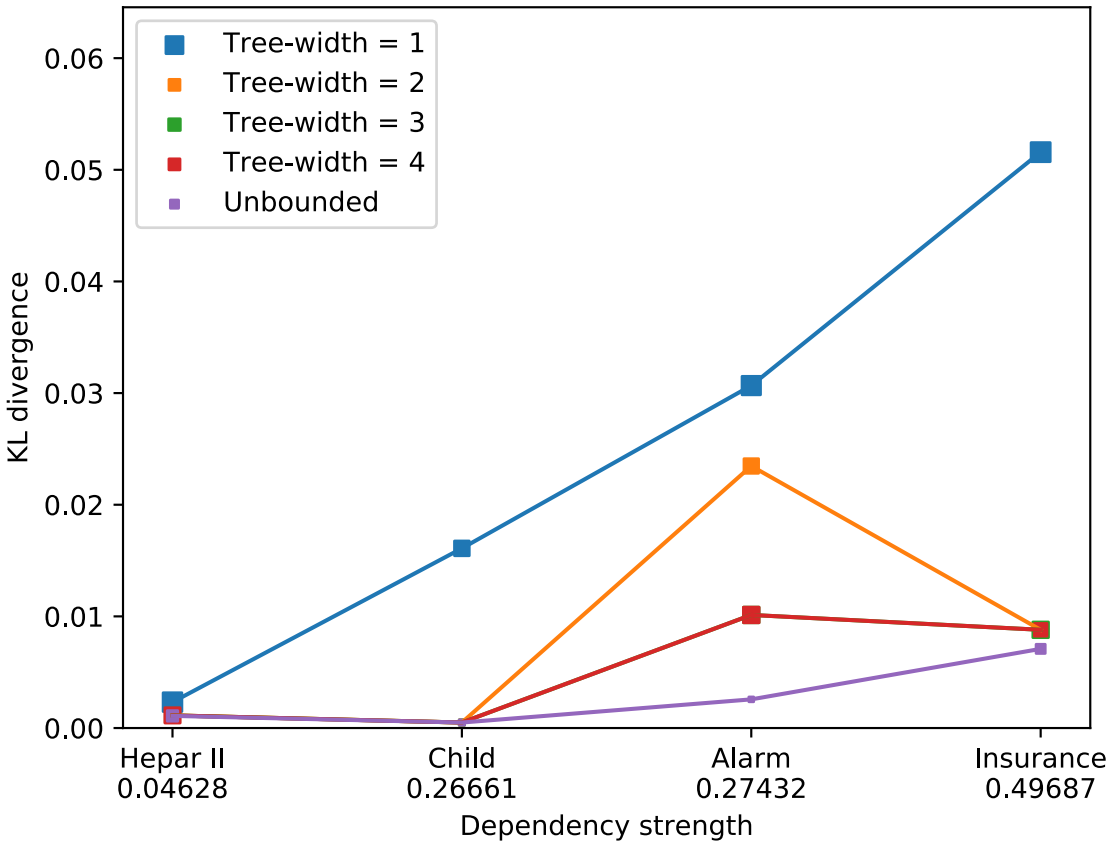


Figure 4.5: Plot of dependency strength versus KL-divergence, excluding the Win95pts network.

Chapter 5

Conclusion

The two main goals of this thesis were to (1) define a measure for dependency strength of Bayesian networks and (2) evaluate how this measure can be used to predict the loss in inference accuracy when bounding the tree-width of Bayesian networks. In this chapter, we conclude by discussing to what extent we have reached these goals.

We defined the dependency strength of an arc as the difference in local log-likelihood with and without the arc as part of the BN structure. We showed that in a single parent relationship the measure of an arc is equivalent to the mutual information between parent and child, which is a theoretic argument that we are in fact measuring the strength of a dependency relationship. In the case of multiple parents of a node, we showed that a simplified example was equivalent to the single parent case, and used this as an argument that our measure generalizes to multiple parents. We showed that our measure is equivalent to that of Nicholson and Jitnah [27] in the single parent case, and similar in the multiple parents case. Two main challenges with the measure are left for future work: The issue of complexity and the issue of combining local arc strengths into the combined strength of a network. The former issue is discussed in Section 3.3.4, suggesting that future work focus on approximating the probabilities not available in the network parameters. This could be done either by following the averaging strategy of Nicholson and Jitnah or by performing some variation of approximate inference. The latter issue is discussed in Section 3.2.1, where we suggested that future work investigate different possibilities of combining local arc strengths.

In Chapter 4 we reported the results from an experiment designed to investigate the correlation between the dependency strength of a network and the loss of inference accuracy when bounding the tree-width of the network. We discussed two weaknesses with the experiment. The first one relates to the small number of networks used in the tests. Measuring the correlation with only five data points, the risk of random correlations must be considered and we argue that the results obtained should only be used as indications. The second weakness that was discussed relates to the usage of TWILP. We observed from the results that TWILP did not always manage to find better approximations as the tree-width bound increased, and thus we can assume that some of the decay in inference results is to blame on poor approximations by TWILP. As it is hard to distinguish between decay caused by the tree-width constraint and decay caused by poor approximations, we cannot draw any strong conclusions regarding the correlation between dependency strength and loss of inference quality when bounding the tree-width of networks.

We did observe some indications of positive correlation. For the bounded networks of tree-width 1, there was a strong correlation of 0.813. One can argue that since for all the networks of tree-width 1 we found the optimal solution, bad approximations by TWILP is not a factor contributing to decay in these networks, and thus we should pay more attention to these networks. For the bounded networks of tree-width 2 and 4, there was close to no correlation, and for the bounded networks of tree-width 3 there was a small positive correlation. There was also a small combined correlation for all the bounded tree-width networks. When the Win95pts network was removed from the experiment due to its complexity, we observed stronger correlations. All of the tree-width bounds showed an increase in the correlation coefficient, and the combined correlation for all bounds increased to 0.476. With these observations, we conclude that there are some indications of a positive correlation between the dependency strength and the loss of inference accuracy when bounding the tree-width of a network, but more and better tests are needed to strengthen the results. As an extension of the work done in this thesis one should perform extended experiments in an attempt to strengthen the empirical results. Performing tests on a larger amount of networks would provide more reliable empirical results. One should also investigate the possibilities of generating networks, and in that way control the complexity of the network to ensure a fair comparison between the approximated networks.

List of Acronyms

BN	Bayesian network.
CPD	conditional probability distribution.
CPT	conditional probability table.
DAG	directed acyclic graph.
GOBNILP	Globally Optimal Bayesian Network learning using Integer Linear Programming.
ILP	integer linear programming.
MILP	mixed-integer linear programming.
MLE	Maximum Likelihood Estimation.
MMI	multivariate mutual information.
PGM	probabilistic graphical model.
RMSE	root mean squared error.
SHD	Structural Hamming distance.
VE	variable elimination.

Bibliography

[1] Silvia Acid and Luis M. de Campos. Searching for Bayesian network structures in the space of restricted acyclic partially directed graphs. *Journal of Artificial Intelligence Research*, 18:445–490, 2003.

[2] Ankur Ankan and Abinash Panda. pgmpy: Probabilistic graphical models using python. In *Proceedings of the 14th Python in Science Conference (SCIPY 2015)*. Citeseer, 2015.

[3] Mark Bartlett and James Cussens. Integer Linear Programming for the Bayesian network structure learning problem. *Artificial Intelligence*, 244:258–271, 2017.

[4] Anil K. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99–109, 1943.

[5] Brent Boerlage. Link strength in Bayesian networks. Master's thesis, University of British Columbia, 1992.

[6] Wray Buntine. Theory Refinement on Bayesian Networks. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, UAI'91, pages 52–60, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[7] Anton Chechetka and Carlos Guestrin. Efficient principled learning of thin junction trees. In *Advances in Neural Information Processing Systems*, pages 273–280, 2008.

[8] David M. Chickering. A transformational characterization of equivalent Bayesian network structures. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 87–98. Morgan Kaufmann Publishers Inc., 1995.

[9] David M. Chickering. Learning Bayesian Networks is NP-Complete. In Doug Fisher and Hans J. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer New York, 1996.

[10] David M. Chickering, David Heckerman, and Christopher Meek. Large-Sample Learning of Bayesian Networks is NP-Hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.

[11] Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2):393–405, 1990.

[12] James Cussens and Mark Bartlett. GOBNILP 1.6.2 User/Developer Manual. *University of York*, 2015.
URL: `https://www.cs.york.ac.uk/aig/sw/gobnilp/manual.pdf`. [Accessed June 3rd, 2019].

[13] Paul Dagum and Michael Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993.

[14] Martijn de Jongh and Marek J. Druzdzel. A comparison of structural distance measures for causal Bayesian network models. *Recent Advances in Intelligent Information Systems, Challenging Problems of Science, Computer Science series*, pages 443–456, 2009.

[15] Alex P. Dow and Richard E. Korf. Best-first Search for Treewidth. In *Proceedings of the 22Nd National Conference on Artificial Intelligence*, volume 2 of *AAAI'07*, pages 1146–1151. AAAI Press, 2007.

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

[17] Maáyan Fishelson and Dan Geiger. Optimizing exact genetic linkage computations. *Journal of Computational Biology*, 11(2-3):263–275, 2004.

[18] Ionut Florescu. *Probability and stochastic processes*. John Wiley & Sons, 2014.

[19] Gerald Gamrath, Tobias Fischer, Tristan Gally, Ambros M. Gleixner, Gregor Hendel, Thorsten Koch, Stephen J. Maher, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Sebastian Schenker, Robert Schwarz, Felipe Serrano, Yuji Shinano, Stefan Vigerske, Dieter Weninger, Michael Winkler, Jonas T.

Witt, and Jakob Witzig. The SCIP Optimization Suite 3.2. Technical report, Optimization Online, 2016.
URL: http://www.optimization-online.org/DB_HTML/2016/03/5360.html. [Accessed June 3rd, 2019].

[20] Gerald Gamrath, Tobias Fischer, Tristan Gally, Ambros M. Gleixner, Gregor Hendel, Thorsten Koch, Stephen J. Maher, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Sebastian Schenker, Robert Schwarz, Felipe Serrano, Yuji Shinano, Stefan Vigerske, Dieter Weninger, Michael Winkler, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 3.2. ZIB-Report 15-60, Zuse Institute Berlin, 2016.
URL: http://nbn-resolving.de/urn:nbn:de:0297-zib-57675. [Accessed June 3rd, 2019].

[21] David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning*, 20(3):197–243, 1995.

[22] Geoff Hulten and Pedro Domingos. VFML – a toolkit for mining high-speed time-changing data streams. Unpublished, [Accessed June 3rd, 2019].
URL: http://www.cs.washington.edu/dm/vfml/.

[23] Joost R. Koiter. Visualizing inference in Bayesian networks. Master's thesis, Delft University of Technology, 2006.

[24] Janne Korhonen and Pekka Parviainen. Exact learning of bounded tree-width Bayesian networks. In *Artificial Intelligence and Statistics*, pages 370–378, 2013.

[25] Solomon Kullback and Richard A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[26] Johan Kwisthout, Hans L. Bodlaender, and Linda C. van der Gaag. The Necessity of Bounded Treewidth for Efficient Inference in Bayesian Networks. In *ECAI*, volume 215, pages 237–242, 2010.

[27] Ann E. Nicholson and Nathalie Jitnah. Using mutual information to determine relevance in Bayesian networks. In Hing-Yan Lee and Hiroshi Motoda, editors, *PRICAI'98: Topics in Artificial Intelligence*, pages 399–410. Springer Berlin Heidelberg, 1998.

[28] Mikhail S. Nikulin. Hellinger distance. *Encyclopedia of mathematics*, 78, 2001. **URL:** `http://www.encyclopediaofmath.org/index.php?title=Hellinger_distance&oldid=16453`. [Accessed June 3rd, 2019].

[29] Pekka Parviainen, Hossein S. Farahani, and Jens Lagergren. Learning bounded tree-width Bayesian networks using integer linear programming. In *Artificial Intelligence and Statistics*, pages 751–759, 2014.

[30] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.

[31] Eric Perrier, Seiya Imoto, and Satoru Miyano. Finding optimal Bayesian network given a super-structure. *Journal of Machine Learning Research*, 9:2251–2286, 2008.

[32] Parot Ratnapinda and Marek J. Druzdzel. An Empirical Evaluation of Costs and Benefits of Simplifying Bayesian Networks by Removing Weak Arcs. *Proceedings of the 27th International Florida Artificial Intelligence Research Society Conference*, pages 508—-511, 2014.

[33] Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

[34] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[35] Nicholas Timme, Wesley Alford, Benjamin Flecker, and John M. Beggs. Multivariate information measures: an experimentalist's perspective. *Journal of Computational Neuroscience*, 36(2):119–140, 2011.

[36] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.

[37] Thomas Verma and Judea Pearl. Equivalence and synthesis of causal models. *UAI '90: Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, pages 255–270, 1991.