MASTER THESIS JURRIAAN PARIE

PERFORMANCE OF SAMPLING METHODS ON DETERMINISTIC BAYESIAN NETWORKS

MSC MATHEMATICAL SCIENCES

UTRECHT UNIVERSITY

SUPERVISED BY:

PROF. DR. GERARD BARKEMA	DR. FRANK PHILLIPSON
DEPARTMENT INFORMATION AND COMPUTING SCIENCES	DEPARTMENT CYBER SECURITY AND ROBUSTNESS
UTRECHT UNIVERSITY	TNO

DECEMBER 22, 2019

Abstract

The performance of the recently introduced *prune sampling* algorithm [1] is characterised for various types of Bayesian networks and is compared to conventional sampling methods like Gibbs sampling, backward sampling, likelihood weighting, and SampleSearch. A procedure is devised to obtain the performance of sampling methods in the limit of infinite simulation time, extrapolated from relatively short simulations. This approach was used to conduct an experimental analysis to compare the accuracy, rate of convergence and the time consumption of the sampling methods. It is shown that Markov chains created by *prune sampling* always converge to the desired posterior distribution, also for networks where conventional Gibbs sampling fails. In addition, it is demonstrated that *prune sampling* outperforms Gibbs sampling – arguably the most widely used MCMC inference technique – at least for one class of BNs. Though, this tempting feature comes at a price. In the first implementation of *prune sampling*, the procedure to assign an initial configuration to a BN is rather time intensive. A solution to mitigate this drawback is implemented and reviewed.

In total 72 experiments are conducted on 12 different BNs. Our conclusion is that, being devised specifically to deal with determinism, *prune sampling* thwarts its expectations. Particularly in its accuracy on the deterministic class of Grid BNs. *Prune sampling* shows consistently fast performance on all types of small BNs. However, on this type of BNs its accuracy is seriously inadequate. On all other BNs, *prune sampling* shows serious shortcomings in terms of all three performance indicators. Hence, overall it needs to be concluded that *prune sampling* is not a competitive sampling method in comparison to established (MCMC) sampling methods.

Part of this thesis is preliminary published on ArXiv as *Prune sampling: a MCMC inference technique for discrete and deterministic Bayesian networks* [1]

Contents

1 Introduction **4**
- 1.1 Bayesian networks and inference . . . 4
- 1.2 Goals and approach of this research project . . . 4
- 1.3 Overview of the thesis . . . 5

2 Bayesian network inference **6**
- 2.1 Bayesian networks . . . 6
- 2.2 Inference . . . 8

3 Approximate inference methods **9**
- 3.1 Sampling . . . 9
- 3.2 Approximate inference . . . 10
- 3.3 Forward and backward sampling . . . 10
- 3.4 Likelihood weighting . . . 13
- 3.5 Sample search . . . 15
- 3.6 Markov chain Monte Carlo simulation . . . 16
  - 3.6.1 Theoretical motivation . . . 16
  - 3.6.2 Metropolis sampling . . . 17
  - 3.6.3 Gibbs sampling . . . 19

4 Prune Sampling **22**
- 4.1 Background: MC-SAT algorithm . . . 22
- 4.2 Notation and definition . . . 22
  - 4.2.1 Regularity and reversibility . . . 26
- 4.3 Practical implementation . . . 28
  - 4.3.1 Generate initial states . . . 28
  - 4.3.2 Sampling from the pruned network . . . 29

5 Performance indicators **31**
- 5.1 Notation . . . 31
- 5.2 Accuracy: the average Hellinger distance . . . 32
- 5.3 Rate of convergence . . . 33
- 5.4 Time consumption . . . 37

6 Results **39**
- 6.1 Experiments . . . 39
- 6.2 Pitfalls of Gibbs sampling . . . 39
  - 6.2.1 Simple deterministic network . . . 40
  - 6.2.2 Block shaped network . . . 41

6.3 Performance on Benchmark Bayesian networks 43
6.3.1 Accuracy on real world Bayesian networks with 0% available evidence 44
6.3.2 Accuracy on real world Bayesian networks with 25% available evidence 45
6.3.3 Accuracy on Grid Bayesian networks 46
6.4 Rate of Convergence 50
6.5 Time consumption 51

7 Improving prune sampling 54
7.1 Exhaustive listing of all feasible states in a pruned network 54
7.2 Hybrid forward sampling 56

8 Conclusion 60
8.1 Accuracy 60
8.2 Rate of convergence 61
8.3 Time consumption 61

A Appendix 62

Chapter 1 **Introduction**

This thesis is about Bayesian network inference. In particular, it is about the performance of various approximate inference sampling methods on deterministic Bayesian networks. Below, a brief overview of the conducted research is given.

1.1 Bayesian networks and inference

A Bayesian network (BN) is a probabilistic model that represents a set of random variables and their conditional dependencies. One could represent a BN graphically by considering a directed acyclic graph where the set of nodes is induced by the set of random variables and where the set of edges is given by the conditional dependencies between these random variables. Assuming that instances fall into one of a number of mutually exclusive and exhaustive classes, discrete BNs are used to model probabilistic relationships. As an illustration, BNs model genetic linkage [2], causal reasoning [3] and defence systems [4]. For all of these models BNs are used to answer probabilistic queries about variables and their relationships. The BN framework entail the tempting feature to make use of updated knowledge of the state of the network's variables. The task of computing the posterior distribution of the BN, given certain evidence is called *inference*. Conducting brute force exact inference on BNs is often too computationally intensive. On the other hand, approximate inference methods have difficulties to guarantee an adequate level of convergence and often perform poorly in the presence of deterministic relations in the BN [5, 6, 7]. In order to deal with those challenges a plethora of different inference strategies have been developed [3, 5, 8, 9].

1.2 Goals and approach of this research project

In order to improve the reliability of approximate inference methods, at TNO a new Markov Chain Monte Carlo (MCMC) approximate inference method named *prune sampling* was created. In this research project, the performance of the first implemented version of *prune sampling* on discrete and deterministic BNs is characterised. In addition, a procedure is devised to obtain the performance of MCMC sampling methods in the limit of infinite simulation time, extrapolated from relatively short simulations. This approach is used to conduct a study to compare the accuracy, rate of convergence and the time consumption of *prune sampling* with four conventional sampling methods: Gibbs sampling, backward sampling, likelihood weighting and SampleSearch. In addition, drawbacks of *prune sampling* are discussed and suggestions are made how those can be resolved. Note that results of this study are summarized in the paper *Prune sampling: a MCMC inference technique for discrete and deterministic Bayesian networks* [1].

1.3 Overview of the thesis

In chapter 2, the theoretical framework of BNs is introduced and the task of BN inference is described. Chapter 3 discusses in detail multiple popular approximate inference techniques and their limitations. In chapter 4, *prune sampling* is introduced and its theoretical justification and implementation is elaborated on. Consecutively in chapter 5, the performance indicators which are used to characterise the sampling methods are presented. In chapter 6, the experimental test results of *prune sampling* in comparison with the four other inference methods are reported and interpreted. In chapter 7, one significant drawback of *prune sampling* is discussed in greater detail and a solution to this shortcoming is implemented and reviewed. Chapter 8 concludes this thesis and proposes suggestions for further research.

Chapter 2 Bayesian network inference

The theoretical framework of this study consists of two main concepts: the Bayesian network (BNs) model and the task of doing BN inference. In this chapter those two main concepts are introduced.

2.1 Bayesian networks

First, the BN representation and its corresponding probabilistic methodology is defined.

Definition 2.1 (Bayesian network)**.** A Bayesian network (BN) structure $\mathcal{G}$ is a Directed Acyclic Graph (DAG) whose nodes represent random variables $\mathcal{X} = (X_1, \ldots, X_n)$. Let Pa_{X_i} denote the direct parents of X_i in $\mathcal{G}$ and ND_{X_i} denote the variables in the graph that are non-descendants of X_i. Then, $\mathcal{G}$ encodes the following set of conditional independence assumptions, called the local independencies, which is denoted by $\mathcal{I}_l(\mathcal{G})$:

$$\text{for each variable } X_i\text{: } (X_i \perp\!\!\!\perp \text{ND}_{X_i} | \text{Pa}_{X_i}).$$

In other words, the local independencies state that each node X_i is conditionally independent of its non-descendants given its parents [5, p. 57].

Consider a state spaces composed solely of discrete-valued random variables. For all variables X_i in the BN, a state $x_i \in \text{Val}(X_i)$ can be assigned. Here, $\text{Val}(X_i)$ denotes the set of values that a random variable X_i can take. The conditional probability distribution $P(X_i|\text{Pa}_{X_i})$ is displayed in a Conditional Probability Table (CPT), where

$$\sum_{i \in \{1,\ldots,n\}} P(x_i|\text{Pa}_{X_i}) = 1.$$

So, a BN exists of a graph with a collection of local probability distributions given in CPTs. Together, these local probability distributions give the joint probability distribution on the BN. The symbol $\mathbf{X} \subseteq \mathcal{X}$ is used to denote the set of all random variables in a BN, while $\mathbf{X}$ denotes an assignment of values to the variables in this set. For convenience, a state (or configuration) of a BN is denoted as $\mathbf{x} = (x_1, \ldots, x_n)$ and $P(\mathbf{x})$ denotes the probability of the BN having this state $\mathbf{x}$. Next, the event called *determinism* is defined [5, p. 158].

Definition 2.2 (Deterministic relation)**.** There exists a function $f : \text{Val}(\text{Pa}_{X_i}) \rightarrow \text{Val}(X_i)$, such that

$$P(x_i|\text{Pa}_{X_i}) = \begin{cases} 1 & x_i = f(\text{Pa}_{X_i}) \\ 0 & \text{otherwise,} \end{cases}$$

i.e. the CPT contains one of more zeros.

A state $\mathbf{x}$ is considered to be feasible if each unique CPT-entry that corresponds to this state is strictly positive.

Definition 2.3 (Feasible state)**.** A feasible state of the BN is a state $\mathbf{x}$ such that $P(\mathbf{x}) > 0$.

The *sample space* Ω is defined as the set containing all possible (not necessarily feasible) states of a BN. All of the above introduced concepts come together in the following example.

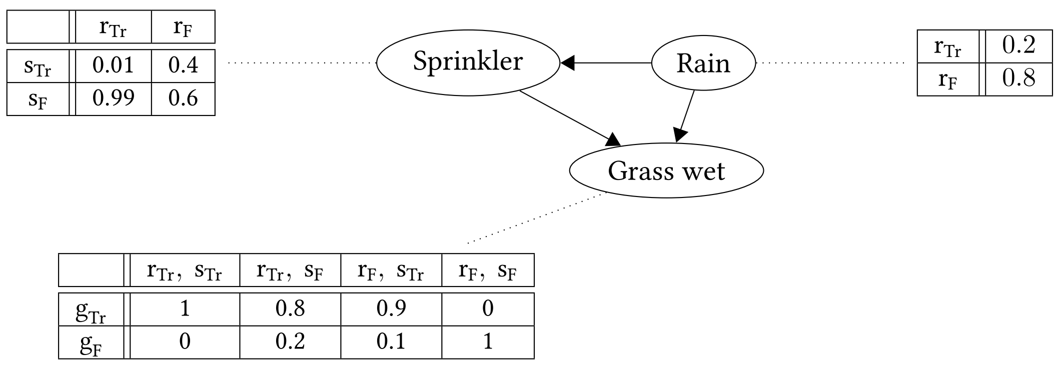
Example 2.4. Consider the *Rain-Sprinkler* BN in Figure 2.1. In this model, one could consider the event of grass (g) being wet as a results of two causes: a sprinkler (s) or rain (r). It is supposed that the rain has a direct effect on the usage of the sprinkler. All three variables have two possible values, Tr (for true or on) and F (for false or off). From the CPTs it becomes clear that when it rains, the sprinkler is usually not turned on, i.e. $P(\mathrm{s_{Tr}}|\mathrm{r_{Tr}}) = 0.01$. The counter intuitive state of the BN, that the grass is wet given it is not raining and the sprinkler is off: $\mathbf{x} = (\mathrm{r_F}, \mathrm{s_F}, \mathrm{g_{Tr}})$, is excluded by the deterministic relation $P(\mathrm{g_{Tr}}|\mathrm{r_F}, \mathrm{s_F}) = 0$. Once considering the fixed (topological) ordering of the variables (R, S, G), the joint probability function is given by

$$P(R, S, G) = P(G|R, S)P(S|R)P(R). \tag{2.1}$$

To the Rain-Sprinkler BN queries can be given like: '*given the grass is wet, what is the probability that it is raining?*'. This probability can be computed by using *Bayes' rule*, i.e.

$$P(R = \mathrm{r_{Tr}}|G = \mathrm{g_{Tr}}) = \frac{P(R = \mathrm{r_{Tr}}, G = \mathrm{g_{Tr}})}{P(G = \mathrm{r_{Tr}})} = \frac{\sum_{S\in\{T,F\}} P(R = \mathrm{r_{Tr}}, S, G = \mathrm{g_{Tr}})}{\sum_{R,S\in\{T,F\}} P(R, S, G = \mathrm{g_{Tr}})}. \tag{2.2}$$

More background concerning the mathematical representation of BNs and its methodology can be found in [5], chapter 3; [9], chapter 2; [3], chapter 2.



	$\mathrm{r_{Tr}}$	$\mathrm{r_F}$
$\mathrm{s_{Tr}}$	0.01	0.4
$\mathrm{s_F}$	0.99	0.6

$\mathrm{r_{Tr}}$	0.2
$\mathrm{r_F}$	0.8

	$\mathrm{r_{Tr}}$, $\mathrm{s_{Tr}}$	$\mathrm{r_{Tr}}$, $\mathrm{s_F}$	$\mathrm{r_F}$, $\mathrm{s_{Tr}}$	$\mathrm{r_F}$, $\mathrm{s_F}$
$\mathrm{g_{Tr}}$	1	0.8	0.9	0
$\mathrm{g_F}$	0	0.2	0.1	1

Figure 2.1: BN structure and corresponding CPTs are used to model the event of grass being wet (g) as a results of two causes: either a sprinkler (s) is on or it is raining (r).

2.2 Inference

The task to answer '*what is $P(R = r_{Tr}|G = g_{Tr})$?*' is called *inference*. Two approaches exist to compute this probability: exact or approximate methods. It is described below how one could answer such queries by computing the exact probability.

When values of variables are known, or are *given*, this set $\mathbf{E} \subset \mathcal{X}$ is called evidence. Now, one can formulate the main goal of inference mathematically as: given a set of evidence $\mathbf{E} = \mathbf{e}$ and nodes of interest $\mathbf{X} \subset \mathcal{X}$, such that $\mathbf{E} \cap \mathbf{X} = \emptyset$, what is the probability distribution $P(\mathbf{X}|\mathbf{E} = \mathbf{e})$? Answering this type of questions is called *inferring unobserved variables.* One can write P as the posterior probability distribution of interest, with reduced CPTs according to the evidence nodes. In the next example, the *Rain-Sprinkler* BN is revisited to illustrate this type of inference.

Example 2.5. Again, consider the BN from Figure 2.1. It is shown how the question in Example 2.4 – *what is $P(R = r_{Tr}|G = g_{Tr})$?* – could be computed explicitly. Using the expansion for the joint probability function from Equation 2.1 and the conditional probabilities from the CPTs, one could evaluate each term in the numerator and denominator of Equation 2.2. As such, one can evaluate for example

$$\begin{aligned}
&P(R = \mathrm{r_{Tr}}, S = \mathrm{s_F}, G = \mathrm{g_{Tr}}) \\
&= P(G = \mathrm{g_{Tr}}|R = \mathrm{r_{Tr}}, S = \mathrm{s_F})P(S = \mathrm{s_F}|R = \mathrm{r_{Tr}})P(R = \mathrm{r_{Tr}}) \\
&= 0.8 \cdot 0.99 \cdot 0.2 \\
&= 0.1584.
\end{aligned}$$

All those calculations together yield

$$\begin{aligned}
&P(R = \mathrm{r_{Tr}}|G = \mathrm{g_{Tr}}) \\
&= \frac{0.00198_{\mathrm{r_{Tr},s_{Tr},g_{Tr}}} + 0.1584_{\mathrm{r_{Tr},s_F,g_{Tr}}}}{0.00198_{\mathrm{r_{Tr},s_{Tr},g_{Tr}}} + 0.288_{\mathrm{r_F,s_{Tr},g_{Tr}}} + 0.1584_{\mathrm{r_{Tr},s_F,g_{Tr}}} + 0.0_{\mathrm{r_F,s_F,g_{Tr}}}} \\
&= \frac{891}{2491} \approx 0.3577.
\end{aligned}$$

Hence, the probability that it is raining, given the grass is wet is approximately 36%.

The procedure described in Example 2.5 is called *exact marginalization.* It illustrates that in order to do inference, there is not always need for an explicit joint distribution. On more complex BNs, *exact marginalization* is vulnerable for the exponentially blow up of the number of computations that need to be executed. In general, all exact inference methods – variable elimination, clique tree propagation, recursive conditioning et cetera – have to perform a number of computations that are exponential in the network's tree-width (a measure for graph complexity). To dwell on those concepts is out of the scope of this thesis. More details can be found in [5], chapter 9; [9], chapter 4.
One way to avoid the exponential blow up is to consider approximate inference methods. In the next chapter, it is discussed how sampling techniques are used to approximate probability distributions on a BN.

Chapter 3 Approximate inference methods

Due to its combinatorial nature exact inference is NP-complete [10]. For this reason, academics have been working on finding efficient approximate inference methods. Most of the developed methods rely on simulations: repeatedly sample values are drawn from BN nodes to generate representative samples of the variables in the Bayesian network. Then, based on relative frequencies of the sample values probabilities of interest are estimated [11, 12, 13, 14, 15]. In doing so, researchers try to find methods that converge quickly to the exact result and to characterize the convergence properties of the simulation algorithms. In this chapter, various sampling and simulation techniques from all classes and their limitations on (deterministic) Bayesian networks are discussed.

3.1 Sampling

In general, sampling aims to approximate the probability distribution as defined by the CPTs. As discussed in chapter 2, reproducing the entire state space Ω is generally not applicable due to computational limitations. Instead, a set of BN configurations $\mathcal{S} \subset \Omega$, called *samples*, is generated. The probability distribution $\widetilde{P}$ on $\mathcal{S}$ is an approximation of the probability distribution P on Ω. The aim of approximate inference sampling methods is to mimic Ω as good as possible by S, such that $\widetilde{P}$ is similar to P. When a set of samples has exactly the same distribution as the real state space, this set of sample is called a *complete uniform.*

As discussed in section 2.1, our main interest is to sample from a finite discrete distribution which is defined below.

Definition 3.1 (Sampling). It is known that: given the state $\mathbf{x}$ of its parents, the distribution of node X_i on a BN is denoted by

$$P(X_i|\text{Pa}(X_i) = \mathbf{x}).$$

Now, suppose $|\text{Val}(X_i)| = M$. Then, with probability $p_1, \dots, p_M$ one of the M possible states is assigned to variable X_i. Note that $\sum_{m=1}^{M} p_m = 1$ and that these probabilities p_m are the column CPT-entries of variable X_i. Sampling from such a distribution can be done by partitioning the interval $[0, 1)$ in M smaller intervals of the form

$$[0, p_1), [p_1, p_1 + p_2), [p_1 + p_2, p_1 + p_2 + p_3), \dots, [1 - p_M, 1).$$

Subsequently, a random number generator could be used to generate a random number r_0 in the interval $[0, 1)$. The unique smaller interval in which r_0 is found yields the sampled state.

This procedure (to assign states to nodes in a BN) is the basis of all sampling methods in this thesis. Before discussing techniques how to generate samples, it is first explained how inference can be conducted on a given collection of generated samples.

3.2 Approximate inference

Suppose one has a collection of 6 Independent and Identically Distributed (IID)) samples of the Rain-Sprinkler BN. Let Rain (R) be our variable of interest and consider the topological ordering (R, S, G). The collection is given by

$$\begin{aligned}
\mathbf{x}^{(1)} &= (R^{(1)} = \mathrm{r_F}, S^{(1)} = \mathrm{s_{Tr}}, G^{(1)} = \mathrm{g_{Tr}}) \\
\mathbf{x}^{(2)} &= (R^{(2)} = \mathrm{r_{Tr}}, S^{(2)} = \mathrm{s_F}, G^{(2)} = \mathrm{g_{Tr}}) \\
\mathbf{x}^{(3)} &= (R^{(3)} = \mathrm{r_F}, S^{(3)} = \mathrm{s_{Tr}}, G^{(3)} = \mathrm{g_{Tr}}) \\
\mathbf{x}^{(4)} &= (R^{(4)} = \mathrm{r_F}, S^{(4)} = \mathrm{s_{Tr}}, G^{(4)} = \mathrm{g_F}) \\
\mathbf{x}^{(5)} &= (R^{(5)} = \mathrm{r_{Tr}}, S^{(5)} = \mathrm{s_{Tr}}, G^{(5)} = \mathrm{g_{Tr}}) \\
\mathbf{x}^{(6)} &= (R^{(6)} = \mathrm{r_F}, S^{(6)} = \mathrm{s_F}, G^{(6)} = \mathrm{g_F}).
\end{aligned}$$

Then, approximate inference is based on the relative frequencies (counting) of the sample values. So, the probability of $R = \mathrm{r_{Tr}}$ can be estimated as

$$\mathbb{E}[\mathbb{1}_{R=\mathrm{r_{Tr}}}] = P(R = \mathrm{r_{Tr}}) \approx \widetilde{P}_6(R = \mathrm{r_{Tr}}) = \frac{1}{6}\sum_{t=1}^{6} \mathbb{1}_{R^{(t)}=\mathrm{r_{Tr}}} = \frac{2}{6},$$

where $\mathbb{1}_{R=\mathrm{r_{Tr}}}$ is the indicator function of the event $R = \mathrm{r_{Tr}}$. In general, for a variable of interest X with $T \in \mathbb{N}$ (total number of i.i.d. samples), the probability of $X = x$ can be approximated by

$$\mathbb{E}[\mathbb{1}_{X=x}] \approx \widetilde{P}_T(X = x) = \frac{1}{T}\sum_{t=1}^{T} \mathbb{1}_{X^{(t)}=x}.$$

The above introduced indicator function $\mathbb{1}$ is a function that indicates if an event of interest occurs or not. Note that the 6 samples from example 3.2 is an approximation of the entire state space and therefore does not share the same distribution, i.e. $P(R = \mathrm{r_{Tr}}) = 0.2 \neq 2/6$.

A straight-forward technique to generate samples is *forward sampling* and is discussed in the next section.

3.3 Forward and backward sampling

Forward sampling is defined by the following procedure.

Algorithm 1 Forward sampling

```
function FORWARD SAMPLING( BN )
    Let $\mathcal{N} = (X_1, \ldots, X_n)$ be a topological ordering of $\mathcal{X}$.

    for $i = 1, \ldots, n$ do
        $X_i \leftarrow$ sample $x_i$ from $P(x_i|\mathrm{Pa}_{X_i})$        ▷ See Definition 3.1
    end for

    return $\mathbf{x} = (x_1, \ldots, x_n)$
end function
```

The above forward sampling algorithm is applied on the Rain-Sprinkler BN in the following example.

Example 3.2. Consider the BN in Figure 2.1 with (topological) ordering (R, S, G). Forward sampling starts with sampling a state for node R according to the unconditioned probability distribution (CPT). Suppose it turns out that $R = \mathrm{r_F}$. Then, node S is sampled from the conditional probability $P(S|R = \mathrm{r_F})$. Suppose this turns out to be $\mathrm{s_{Tr}}$. Then, node G is sampled according to the conditional probability $P(G|R = \mathrm{r_F}, S = \mathrm{s_{Tr}})$. Let's say this state becomes $\mathrm{g_{Tr}}$. In this manner, forward sampling returns the sampled state $\mathbf{x} = (\mathrm{r_F}, \mathrm{s_{Tr}}, \mathrm{g_{Tr}})$.

A drawback of forward sampling is that any available evidence is not taken into account. Suppose that the following evidence is given: $S = \mathrm{s_F}$. Then, the generated sample in Example 3.2 turns out to be infeasible. One could come up with a quick fix by simply removing all infeasible BN states from the generated samples. This is called *rejection sampling*. However, a problem arises when evidence occurs with low probability. In this case, a huge amount of samples needs to be generated in order to reflect the true BN probability distribution, since a large amount of samples are rejected.

Backward sampling deals with available evidence in a more sophisticated way. Instead of sampling according to a fixed topological ordering, backward sampling starts sampling back from given evidence nodes. This is repeated until a state is assigned to all parent nodes of the evidence nodes. Any remaining nodes are sampled according to forward sampling. This procedure is illustrated in the following example.

Example 3.3. Consider the binary BN in Figure 3.1. Suppose the value of node D is observed to be $\mathrm{D_F}$ and consider the backward sampling order to be (D, B) and the forward sampling order to be (E). Joint values of state B and C are denoted as $\mathrm{B_{Tr}},\ \mathrm{C_{Tr}}$; $\mathrm{B_{Tr}},\ \mathrm{C_F}$; $\mathrm{B_F},\ \mathrm{C_{Tr}}$ and $\mathrm{B_F},\ \mathrm{C_F}$. The procedure of backward sampling starts at the evidence node. Since $D = \mathrm{D_F}$, the state of parent nodes B and C is sampled from the second row in D's CPT according to the normalized sampling distribution displayed in Table 3.1. The constant

$$c_1 = \mathrm{d_{Tr,Tr,F}} + \mathrm{d_{Tr,F,F}} + \mathrm{d_{F,\,Tr,F}} + \mathrm{d_{F,F,F}}$$

normalizes the probabilities to sum to 1. Suppose the sampling steps chooses joint state $\mathrm{B_F},\ \mathrm{C_{Tr}}$ and sets the states of those nodes to those values. Next, node A is sampled

$P(B, C \mid D = \mathrm{D_F})$	$\mathrm{B_{Tr}},\ \mathrm{C_{Tr}}$	$\mathrm{B_{Tr}},\ \mathrm{C_F}$	$\mathrm{B_F},\ \mathrm{C_{Tr}}$	$\mathrm{B_F},\ \mathrm{C_F}$
$\mathrm{D_F}$	$\frac{\mathrm{d_{Tr,Tr,F}}}{c_1}$	$\frac{\mathrm{d_{Tr,F,F}}}{c_1}$	$\frac{\mathrm{d_{F,Tr,F}}}{c_1}$	$\frac{\mathrm{d_{F,F,F}}}{c_1}$

Table 3.1: Sampling distribution normalized by constant $c_1 = \mathrm{d_{Tr,Tr,F}} + \mathrm{d_{Tr,F,F}} + \mathrm{d_{F,\ Tr,F}} + \mathrm{d_{F,F,F}}$

given that $B = \mathrm{B_F}$. The normalized sampling distribution is given in Table 3.2. Suppose

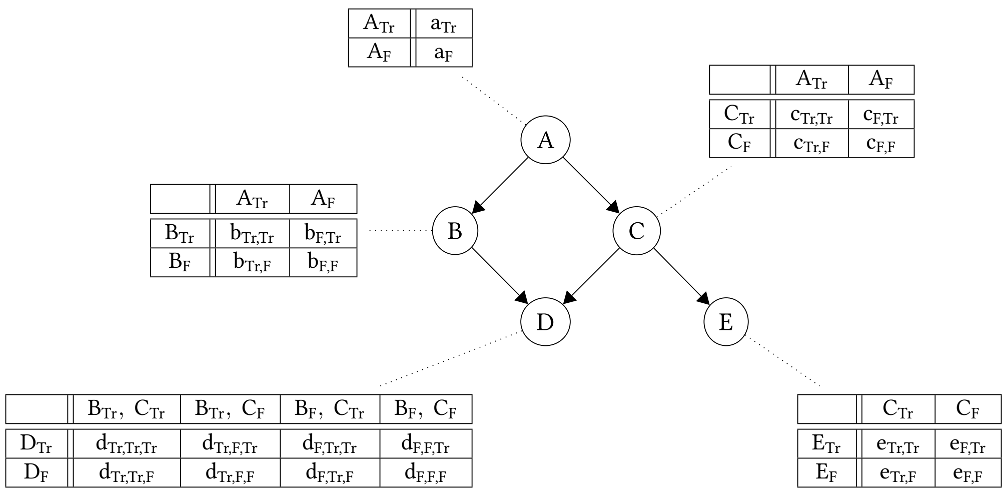
$P(A \mid B = \mathrm{B_F})$	$\mathrm{A_{Tr}}$	$\mathrm{A_F}$
$\mathrm{B_F}$	$\frac{\mathrm{a_{Tr,F}}}{c_2}$	$\frac{\mathrm{a_{F,F}}}{c_2}$

Table 3.2: Sampling distribution normalized by constant $c_2 = \mathrm{a_{Tr,F}} + \mathrm{a_{F,F}}$

that the sampling procedure picks out $A = \mathrm{A_{Tr}}$. Finally, forward sampling is used to assign a state to node E taking into account that $C = \mathrm{C_{Tr}}$. Note that forward sampling does not normalize the sampling distribution since the CPT entries do already sum up to 1. The distribution is displayed in Table 3.3 and suppose $\mathrm{E_{Tr}}$ is selected [16].

$P(E \mid C = \mathrm{C_{Tr}})$	$\mathrm{E_{Tr}}$	$\mathrm{E_F}$
$\mathrm{C_{Tr}}$	$\mathrm{e_{Tr,Tr}}$	$\mathrm{e_{Tr,F}}$

Table 3.3: Sampling distribution for forward sampling



$\mathrm{A_{Tr}}$	$\mathrm{a_{Tr}}$
$\mathrm{A_F}$	$\mathrm{a_F}$

	$\mathrm{A_{Tr}}$	$\mathrm{A_F}$
$\mathrm{B_{Tr}}$	$\mathrm{b_{Tr,Tr}}$	$\mathrm{b_{F,Tr}}$
$\mathrm{B_F}$	$\mathrm{b_{Tr,F}}$	$\mathrm{b_{F,F}}$

	$\mathrm{A_{Tr}}$	$\mathrm{A_F}$
$\mathrm{C_{Tr}}$	$\mathrm{c_{Tr,Tr}}$	$\mathrm{c_{F,Tr}}$
$\mathrm{C_F}$	$\mathrm{c_{Tr,F}}$	$\mathrm{c_{F,F}}$

	$\mathrm{B_{Tr}},\ \mathrm{C_{Tr}}$	$\mathrm{B_{Tr}},\ \mathrm{C_F}$	$\mathrm{B_F},\ \mathrm{C_{Tr}}$	$\mathrm{B_F},\ \mathrm{C_F}$
$\mathrm{D_{Tr}}$	$\mathrm{d_{Tr,Tr,Tr}}$	$\mathrm{d_{Tr,F,Tr}}$	$\mathrm{d_{F,Tr,Tr}}$	$\mathrm{d_{F,F,Tr}}$
$\mathrm{D_F}$	$\mathrm{d_{Tr,Tr,F}}$	$\mathrm{d_{Tr,F,F}}$	$\mathrm{d_{F,Tr,F}}$	$\mathrm{d_{F,F,F}}$

	$\mathrm{C_{Tr}}$	$\mathrm{C_F}$
$\mathrm{E_{Tr}}$	$\mathrm{e_{Tr,Tr}}$	$\mathrm{e_{F,Tr}}$
$\mathrm{E_F}$	$\mathrm{e_{Tr,F}}$	$\mathrm{e_{F,F}}$

Figure 3.1: BN with available evidence: $D = \mathrm{D_F}$

In Example 3.1, backward sampling generates the sample $(\mathrm{A_{Tr}}, \mathrm{B_F}, \mathrm{C_{Tr}}, \mathrm{D_F}, \mathrm{E_{Tr}})$. For general purpose, the pseudo code of backward sampling is given in Algorithm 2 .
Even though backward sampling mitigates the rejection problem, it could still be difficult to generate enough feasible samples within a reasonable amount of time. A different way of making use of available evidence is discussed in the next section.

Algorithm 2 Backward sampling

```
function BACKWARD SAMPLING( BN, $\mathbf{E}$ )
    Let $\mathcal{N}_\text{B} \subseteq \mathcal{X}$ be a ordering of the backward sampling nodes
    Let $\mathcal{N}_\text{F} \subseteq \mathcal{X}$ be a ordering of the forward sampling nodes.

    for $X_i \in \mathcal{N}_\text{B}$ do
        Set $X_i = x_i$ according to evidence
        $X_i \leftarrow$ sample $x_i$ from $\frac{P(x_i|\text{Pa}_{X_i})}{c}$          ▷ See Definition 3.1
                                                    ▷ $c$ is a normalization constant (see Example 3.3)
    end for

    for $X_i \in \mathcal{N}_\text{F}$ do
        $X_i \leftarrow$ sample $x_i$ from $P(x_i|\text{Pa}_{X_i})$
    end for

    return $\mathbf{x} = (x_1, \ldots, x_n)$
end function
```

3.4 Likelihood weighting

The idea of rejection sampling is cumbersome: numerous samples conflicting with the given evidence are generated and are subsequently rejected without contributing to the estimator. In contrary, *likelihood weighting* (LW) sampling makes – like backward sampling – use of available evidence in a more ingenious way.

The basic idea of likelihood weighting is to assign weights to node configurations that correspond to the likelihood of the evidence accumulated throughout the sampling process. This idea is illustrated in the following example in greater detail.

Example 3.4. Consider again the Rain-Sprinkler BN in Figure 2.1 with (topological) ordering (R, S, G) and assume that the following evidence is available: $S = \text{s}_\text{Tr}$. A naive sampling technique would set $S = \text{s}_\text{Tr}$ and would sample R and G from the initial CPTs, the so-called the *prior distribution.* In doing so, the expected number of samples with $R = \text{r}_\text{Tr}$ would be 0.2. This conflicts with the *posterior distribution* that takes into consideration the available evidence and implies that the expected number of samples with $R = \text{r}_\text{Tr}$ conditioned on $S = \text{s}_\text{Tr}$ is 0.01. Hence, this approach falls short to identify that posterior probability of $R = \text{r}_\text{Tr}$ is lower when one observes $S = \text{s}_\text{Tr}$.

One way to resolve this miscalculation, is to assign weights to the generated samples. Consider the sample $\mathbf{x} = (R = \text{r}_\text{Tr}, S = \text{s}_\text{Tr}, G = \text{g}_\text{Tr})$ is generated. Then, the weight $w_\mathbf{x}$ is assigned to the likelihood that the sample would occur given the available evidence $S = \text{s}_\text{Tr}$. Which equals the the product of the CPT entries

$$\begin{aligned} w_\mathbf{x} &= P(S = \text{s}_\text{Tr}|R = \text{r}_\text{Tr}) && (3.1) \\ &= 0.01. && (3.2) \end{aligned}$$

Accordingly, the sample $\mathbf{x}' = (\text{r}_\text{F}, \text{s}_\text{Tr}, \text{g}_\text{Tr})$ yields $w_{\mathbf{x}'} = 0.4$. This process generates *weighted particles* and the conditional probability that $R = \text{r}_\text{Tr}$ given $S = \text{s}_\text{Tr}$ is given by

$$P(S = \mathrm{s}_{\mathrm{Tr}} | R = \mathrm{r}_{\mathrm{Tr}}) \approx \frac{0.01}{0.01 + 0.4} \approx 0.024.$$

Algorithm 3 describes the procedure of LW. In general, the conditional probability of event A given B can be estimated using a collection of samples $\mathcal{S}$

$$\begin{aligned} P(A|B) &\approx \frac{\sum_{\mathbf{x} \in \mathcal{S}} w_{\mathbf{x}} \cdot \mathbb{1}_{A,B}(\mathbf{x})}{\sum_{\mathbf{x} \in \mathcal{S}} w_{\mathbf{x}} \cdot \mathbb{1}_{B}(\mathbf{x})} \\ &= \frac{\sum_{\mathbf{x} \in \mathcal{S}} w_{\mathbf{x}} \cdot \mathbb{1}_{B}(\mathbf{x})}{\sum_{\mathbf{x} \in \mathcal{S}} w_{\mathbf{x}}}, \end{aligned}$$

where the latter equality follows from the fact that event B (the evidence) is fixed. Note that without available evidence likelihood weighting equals forward sampling. Moreover, likelihood weighting is an improvement of forward sampling since LW does not necessarily reject samples (unless its weight is zero).

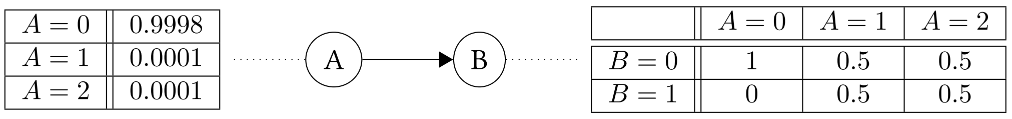
Algorithm 3 Likelihood weighting sampling

```
function LW-SAMPLING( BN, E )
    Let $\mathcal{N} = (X_1, \dots, X_n)$ be a topological ordering of $\mathcal{X}$.

    $w \leftarrow 1$
    for $i = 1, \dots, n$ do
        if $X_i \notin \mathbf{E}$ then
            $X_i \leftarrow$ sample $x_i$ from $P(x_i|\mathrm{Pa}_{X_i})$        ▷ See Definition 3.1
        else
            Set $X_i = x_i$ according to evidence
            $w \leftarrow w \cdot P(x_i|\mathrm{Pa}_{X_i})$
        end if
    end for
    return $\mathbf{x} = (x_1, \dots, x_n), w$
end function
```

A drawback of LW sampling is the occurrence of a near deterministic relation. This is illustrated in the following example.

Example 3.5. Consider the BN in Figure 3.2. Suppose that available evidence is given: $B = 1$. Hence, it holds that $A = 1$ or $A = 2$ both with probability 0.5. If LW sampling is applied to determine the conditional probability $P(A|B = 1)$, due to the prior distribution of A numerous samples will be created with $A = 0$. Since these samples are incompatible with the evidence zero weights are assigned to those samples, i.e. rejection. On average only $2/10.000$ samples are generated either with $A = 1$ or $A = 2$. This drawback could make LW on this type of BNs inefficient and time-consuming to find an accurate estimate of the true posterior distribution.



$A = 0$	0.9998
$A = 1$	0.0001
$A = 2$	0.0001

	$A = 0$	$A = 1$	$A = 2$
$B = 0$	1	0.5	0.5
$B = 1$	0	0.5	0.5

Figure 3.2: BN with a near deterministic relation poses difficulties for likelihood sampling

Example 3.5 demonstrates a major drawback of sampling methods in general: if the proposal distribution is not close enough to the true posterior distribution, the sampling processes is not working well in terms of accuracy and efficiency. An approach to deal with this challenge is so-called Markov chain Monte Carlo simulations, which is discussed in section 3.6.1. But first, a method is introduced that deals with the inabilities of both backward sampling and likelihood weighting sampling.

3.5 Sample search

Gogate and Dechter devised SampleSearch to prevent samples to be rejected by zero weights in deterministic BNs [17]. Algorithm 4 gives the pseudo code of SampleSearch. Rather than rejecting a sample as soon as its weight becomes zero and restarting the sample process, SampleSearch goes back and reassigns variables until one finds a state that has a non-zero weight. Basically, SampleSearch starts sampling according to a certain topological ordering. If SampleSearch visits an evidence node where the probability of the evidence is 0 given the values previously assigned to the variable's parents, SampleSearch backtracks the previous variable in the topological ordering, excludes the value that was previously assigned, renormalizes the conditional distribution and chooses a different value. What if all the values within a domain are excluded? SampleSearch backtracks further, until all variables have been assigned a value that is compatible with the evidence [18]. To put it briefly, SampleSearch guarantees that all generated samples have non-zero weights.

Even more interesting for the scope of this thesis, the SampleSearch algorithm is devised to deal with BNs with a substantial amount of deterministic relations. However, the SampleSearch approach – performing a search when a sample is not compatible with the evidence – has two problems:

1. search results in bias
2. search is computational intensive.

This first drawback can be addressed by an adjusted weighting. It can be shown that an alternative weighting scheme can be realized that guarantees asymptotically unbiased samples [17]. However, this comes with the price of extra computational time.

In existing literature, various sampling techniques are combined with SampleSearch, for example Backward SampleSearch (BSS) and BSS with backjumping, which jumps

Algorithm 4 SampleSearch sampling

```
function SampleSearch sampling( BN, $\mathbf{E}$ )
    Let $\mathcal{N} = (X_1, \dots, X_n)$ be a topological ordering of $\mathcal{X}$.

    for $i = 1, \dots, n$ do
        $X_i \leftarrow$ sample $x_i$ from $P(x_i|\text{Pa}_{X_i})$                ▷ See Definition 3.1
        if $(x_1, \dots, x_i)$ violates $\mathbf{E}$ then
            $X_i \leftarrow$ sample $x_i$ from $P(x_i|\text{Pa}_{X_i})$
        else
            Continue
        end if
    end for
    return $\mathbf{x} = (x_1, \dots, x_n)$
end function
```

back to the node with the highest index in the sampling order that has bearing on the conflict that was encountered [18]. However, a plethora of different methods have been developed to generate representative samples. Giving a complete overview of all developed methods is out the scope of this thesis. For an overview of a wide selection of approximate sampling techniques and their limitations one can turn to [5], chapter 12; [9] chapter 4. In the next section, a last method is discussed that proves to be effective to get close to the true posterior distribution during the sampling process.

3.6 Markov chain Monte Carlo simulation

Markov chain Monte Carlo (MCMC) sampling methods are a different class of methods than discussed in the previous sections. MCMC sampling methods are characterized by the generation of a sequence of samples. This sequence is created such that subsequent sequences are generated from distributions that provably get increasingly closer to the desired posterior distribution. In this section, some theoretical background is given about MCMC sampling methods and advantages and disadvantages of MCMC inference techniques are discussed.

3.6.1 Theoretical motivation

Consider a collection of samples as in Section 3.2. Suppose that sample $\mathbf{x}^{(t)}$ with $t \geq 1$ is not created step-by-step but that sample $\mathbf{x}^{(t+1)}$ is constructed by modifying sample $\mathbf{x}^{(t)}$. Repeating this modification process yields a Markov chain $(\mathbf{x}^{(t)})_{t \in \mathbb{N}_0}$, i.e. in the context of BNs a Markov chain could be seen as a random walk over the states of variables in the BN. Hence, repeating this process many times is therefore called a *Markov chain Monte Carlo* (MCMC) simulation. Due to the clever way of modifying samples, MCMC methods construct a Markov chain such that, although the first sample may be generated from the prior distribution, successive samples are generated from a distribution that provably gets closer and closer to the desired posterior distribution.

It could be shown [5, p. 517] that $\widetilde{P}_T \to P$ as $T \to \infty$. In order to use this tempting feature of MCMC methods, it needs to be guaranteed that a limiting process of the Markov chain exists and is unique. Since only Markov chains on finite state spaces are considered, from the theory of Markov chains it is known that if a Markov chain is regular and reversible with respect to a distribution π, then π is a unique stationary distribution. These notions are defined below more formally.

Definition 3.6 (Regular or ergodic Markov chain)**.** A Markov chain is said to be regular if there exists some number $H \in \mathbb{N}$ such that for every state $\mathbf{x}, \mathbf{x}'$ the probability of getting from $\mathbf{x}$ to $\mathbf{x}' \in \text{Val}(\mathcal{X})$, denoted as $(\mathbf{x} \to \mathbf{x}')$, in exactly H steps is > 0.

Definition 3.7 (Transition model)**.** A Markov chain is defined through the state space $\text{Val}(\mathcal{X})$ and a transition model $\mathcal{T}$ that defines for all $\mathbf{x} \in \text{Val}(\mathcal{X})$ a next-state distribution over $\text{Val}(\mathcal{X})$. A transition model specifies for each pair $\mathbf{x}, \mathbf{x}'$ the probability $\mathcal{T}(\mathbf{x} \to \mathbf{x}')$.

Definition 3.8 (Reversible Markov chain)**.** A finite-state Markov chain $\mathcal{T}$ is called reversible if there exists a unique distribution π such that for all states $\mathbf{x}$ and $\mathbf{x}'$

$$\pi(\mathbf{x})\mathcal{T}(\mathbf{x} \to \mathbf{x}') = \pi(\mathbf{x}')\mathcal{T}(\mathbf{x}' \to \mathbf{x}). \tag{3.3}$$

Equation 3.3 is known as the *detailed balance equation.*

Definition 3.9 (Stationary distribution)**.** A distribution π is a stationary distribution for a Markov chain $\mathcal{T}$ if

$$\pi(\mathbf{x}') = \sum_{\mathbf{x} \in \text{Val}(\mathbf{X})} \pi(\mathbf{x})\mathcal{T}(\mathbf{x} \to \mathbf{x}'). \tag{3.4}$$

Not all Markov chains satisfy the reversibility condition. A method that guarantees the construction of a reversible Markov chain with a particular stationary distribution is *Metropolis sampling.*

3.6.2 Metropolis sampling

Metropolis sampling samples from a proposal distribution $\mathcal{T}^Q$ that defines a transition model over the state space. In contrary to the sampling methods discussed in section 3.4-3.5, Metropolis sampling does not keep track of weights. Instead, randomly a proposed transition is accepted or rejected along with a probability that corrects for the resulting error.

In the context of BNs, Metropolis sampling follows the procedure below.

Definition 3.10 (Metropolis sampling)**.**

- Select an initial state $\mathbf{x}^{(0)}$ for the BN generated by, for example, forward sampling;
- for each iteration $0 < t \leq T$: the transition model $\mathcal{T}^Q$ defines a distribution over possible successor states of $\mathbf{x}$ in $\text{Val}(\mathbf{X})$ from which one candidate sample $\mathbf{x}'$ is selected randomly. The choice of the proposal distribution can be arbitrary as long as it results in a regular Markov chain;

- the proposed transition $(\mathbf{x} \to \mathbf{x}')$ is either accepted or rejected. The *acceptance probability* is denoted by $\mathcal{A}(\mathbf{x} \to \mathbf{x}')$. So, the transitional model of the Markov chain is given by

$$\begin{aligned}
\mathcal{T}(\mathbf{x} \to \mathbf{x}') &= \mathcal{T}^Q(\mathbf{x} \to \mathbf{x}')\mathcal{A}(\mathbf{x} \to \mathbf{x}'), \qquad \text{where } \mathbf{x} \neq \mathbf{x}' \\
\mathcal{T}(\mathbf{x} \to \mathbf{x}) &= \mathcal{T}^Q(\mathbf{x} \to \mathbf{x}) + \sum_{\mathbf{x}' \neq \mathbf{x}} \mathcal{T}^Q(\mathbf{x} \to \mathbf{x}')(1 - \mathcal{A}(\mathbf{x} \to \mathbf{x}'));
\end{aligned}$$

- given a proposal distribution $\mathcal{T}^Q$, the detailed balance equation from (3.3) can be used to determine the acceptance probability γ. Since,

$$\pi(\mathbf{x})\mathcal{T}^Q(\mathbf{x} \to \mathbf{x}')\mathcal{A}(\mathbf{x} \to \mathbf{x}') = \pi(\mathbf{x}')\mathcal{T}^Q(\mathbf{x}' \to \mathbf{x})\mathcal{A}(\mathbf{x}' \to \mathbf{x})$$

it follows that

$$\gamma = \mathcal{A}(\mathbf{x} \to \mathbf{x}') = \min\left[1, \frac{\pi(\mathbf{x}')\mathcal{T}^Q(\mathbf{x}' \to \mathbf{x})}{\pi(\mathbf{x})\mathcal{T}^Q(\mathbf{x} \to \mathbf{x}')}\mathcal{A}(\mathbf{x}' \to \mathbf{x})\right];$$

- generate a uniform random number $u \in [0, 1]$
  - if $u \leq \gamma$ then $\mathbf{x}^{(t)} = \mathbf{x}'$;
  - if $u > \gamma$ then $\mathbf{x}^{(t)} = \mathbf{x}^{(t-1)}$.

The above construction allows us to produce a Markov chain for an arbitrary stationary distribution. Though, it does not guarantee convergence to the desired distribution, since the constructed Markov chain is not necessarily regular. This property does not follow directly from Metropolis sampling. An example in which Metropolis sampling constructs a non-regular Markov chain is given in [5, p. 514].

In the next example, Metropolis sampling is applied on the *Rain-Sprinkler* network.

Example 3.11. Reconsider the BN in Figure 2.1. Suppose that no evidence is available and that the initial state is

$$\mathbf{x}^{(0)} = (R^{(0)} = \mathrm{r_{Tr}}, S^{(0)} = \mathrm{s_F}, G^{(0)} = \mathrm{g_{Tr}}).$$

Then, $R^{(1)}$ is selected according to the distribution $P(R^{(1)} = \mathrm{r_{Tr}}) = P(R^{(1)} = \mathrm{r_F}) = 0.5$. Repeating this for S and G, one could find a candidate state

$$\mathbf{x}' = (R' = \mathrm{r_{Tr}}, S' = \mathrm{s_F}, G' = \mathrm{g_F}).$$

The acceptance probability $\gamma = \mathcal{A}(\mathbf{x}^{(0)} \to \mathbf{x}')$ is computed by

$$\gamma = \frac{P(R' = \mathrm{r_{Tr}}, S' = \mathrm{s_F}, G' = \mathrm{g_F})}{P(R^{(0)} = \mathrm{r_{Tr}}, S^{(0)} = \mathrm{s_F}, G^{(0)} = \mathrm{g_{Tr}})} = \frac{0.0396}{0.1584} = 0.25.$$

Then, $\mathbf{x}^{(1)} = \mathbf{x}'$ is accepted if the uniform random number $u \in [0, 1]$ is smaller than 0.25, otherwise $\mathbf{x}^{(1)} = \mathbf{x}^{(0)}$.

By repeating this procedure a collection of T samples of the BN can be generated. On this collection of samples approximate inference techniques, as described in section 3.2, can be applied. Metropolis sampling with a particular choice of proposal distribution is widely used and is discussed in the next subsection.

3.6.3 Gibbs sampling

Gibbs sampling [19] is one of the most popular MCMC methods to date. In this subsection, the connection between Metropolis and Gibbs sampling is discussed and it is shown how regularity and reversibility of a Markov chain could break down due to the appearance of deterministic relations in the BN.

Definition 3.12 (Gibbs sampling)**.** As does Metropolis sampling, Gibbs sampling starts with an initial assignment $\mathbf{x}^{(0)}$ to all variables in the BN generated by, for example, forward sampling. Starting from $\mathbf{x}^{(0)}$ it performs $0 \leq t \leq T$ so-called *Gibbs iterations*, $T \in \mathbb{N}$. One single Gibbs iteration resamples all $i = 1, \dots, n$ variables in the BN by fixing all but one variable. Resampling the i-th variable in the t-th Gibbs iteration is done by sampling from the distribution $P(X_i|\mathbf{x}_{-i}^{(t)})$, where $\mathbf{x}_{-i}^{(t)} = (x_1^{(t)}, \dots, x_{i-1}^{(t)}, x_{i+1}^{(t-1)}, \dots, x_n^{(t-1)})$ for $i = 2, \dots, n-1$. After T samples are generated according to this procedure, all one-variable marginals can be estimated by the following quantity

$$\widetilde{P}_T(x_i) = \frac{1}{T} \sum_{t=1}^{T} P(x_i|\mathbf{x}_{-i}^{(t)}). \tag{3.5}$$

It could be shown, that in the limit of infinite samples, $\widetilde{P}_T(x_i)$ will converge to $P(x_i)$ if the underlying Markov chain is regular and reversible [20].

The procedure for Gibbs sampling is given in Algorithm 5.

Algorithm 5 Gibbs sampling

```
function GIBBS SAMPLING( BN, $\mathbf{E}$, $\mathbf{x}^{(0)}$)
    Let $\mathcal{N} = (X_1, \dots, X_n)$ be a topological ordering of $\mathcal{X}$.

    for $t = 1, \dots, T$ do
        $\mathbf{x}^{(t)} \leftarrow \mathbf{x}^{(t-1)}$
        for each $X_i \in \mathbf{X}$ do
            Sample $x_i^{(t)}$ from $P(X_i|\mathbf{x}_{-i}^{(t)})$                ▷ See Definition 3.1
        end for
    end for
    return $\mathbf{x}^{(0)}, \dots, \mathbf{x}^{(T)}$
end function
```

In the next example, Gibbs sampling is applied on the *Rain-Sprinkler* network.

Example 3.13. Reconsider the BN in Figure 2.1. Suppose no evidence is available and the following initial state is given

$$\mathbf{x}^{(0)} = (R^{(0)} = \mathrm{r_{Tr}}, S^{(0)} = \mathrm{s_F}, G^{(0)} = \mathrm{g_{Tr}}).$$

For each Gibbs iteration, all unobserved variables are resampled, one at a time, in a predetermined order, say (R, S, G). So, first $R^{(1)}$ is sampled from the distribution $P(R|S^{(0)} = \mathrm{s_F}, G^{(0)} = \mathrm{g_{Tr}})$, which equals

$$\begin{aligned}
&P(R = \mathrm{r_{Tr}}|S^{(0)} = \mathrm{s_F}, G^{(0)} = \mathrm{g_{Tr}}) \\
&= \frac{P(\mathrm{r_{Tr}}, \mathrm{s_F}, \mathrm{g_{Tr}})}{P(\mathrm{s_F}, \mathrm{g_{Tr}})} \\
&= \frac{P(\mathrm{r_{Tr}}, \mathrm{s_F}, \mathrm{g_{Tr}})}{P(\mathrm{s_F}, \mathrm{g_{Tr}}|\mathrm{r_{Tr}}) + P(\mathrm{s_F}, \mathrm{g_{Tr}}|\mathrm{r_F})} \\
&= \frac{0.1584}{0.00198 + 0.288} \\
&\approx 0.5462.
\end{aligned}$$

Hence, $P(R = \mathrm{r_F}|S^{(0)} = \mathrm{s_F}, G^{(0)} = \mathrm{g_{Tr}}) \approx 0.4538$. Suppose $R^{(1)} = \mathrm{r_F}$ is sampled, Gibbs sampling continues with resampling $S^{(1)}$ from the distribution $P(S|R^{(1)} = \mathrm{r_F}, G^{(0)} = \mathrm{g_{Tr}})$, obtaining for example $S^{(1)} = \mathrm{s_{Tr}}$. Finally, sampling $G^{(1)}$ from $P(G|R^{(1)} = \mathrm{r_F}, S^{(1)} = \mathrm{s_{Tr}})$ could result in $\mathrm{g_{Tr}}$. Then, the result of the first Gibbs iteration is the sample $\mathbf{x}^{(1)} = (R^{(1)} = \mathrm{r_F}, S^{(1)} = \mathrm{s_{Tr}}, G^{(1)} = \mathrm{g_{Tr}})$. This process can be repeated to generate a collection of samples.

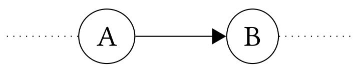
As mentioned before, Gibbs sampling could perform poorly in the presence of deterministic relations [5, 6, 7]. In the next example, this phenomenon is illustrated.

Example 3.14. Consider the BN in Figure 3.3. Suppose the initial configuration is given by $\mathbf{x}^{(0)} = (A^{(0)} = 0,\ B^{(0)} = 0)$ – shortened $(0, 0)$ – and suppose no evidence is available. In the first Gibbs iteration, one resamples both unobserved variables, one at a time, in the order A, B. So, one first samples $A^{(1)}$ from the distribution $P(A|B^{(0)} = 0)$. According to the CPT, with probability 1 this turns out to be $A = 0$. Consecutively, one samples $B^{(1)}$ from the distribution $P(B|A^{(1)} = 0)$. Which always returns $B = 0$. As a consequence, the Markov chain created by Gibbs sampling behaves like

$$(0, 0) \to (0, 0) \to (0, 0) \to \ldots,$$

yielding that $\mathbf{x}^{(i)} = (0, 0)$ for all $i \geq 0$. Hence, inference based on Gibbs sampling returns $P(A = 0) = 1$. However, the true distribution for A equals $P(A = 0) = P(A = 1) = 0.5$.

$A = 0$	0.5
$A = 1$	0.5



	$A = 0$	$A = 1$
$B = 0$	1	0
$B = 1$	0	1

Figure 3.3: BN with a deterministic relation. The state of B equals the state of A with probability 1.

When no deterministic relations are present in a BN, it could be shown [6] that Gibbs sampling generates a regular and reversible Markov chain (and therefore converges to the desired posterior distribution). Though, as illustrated in Example 3.14, when deterministic dependencies are present in a BN, regularity and reversibility could break down and the estimation given in Equation (3.5) does not always converge to $P(x_i)$.

Many solutions have been proposed in the past to address this problem [6, 20]. So do we: at TNO a MCMC method is devised that *always* converges to the desired posterior distribution, especially in the presence of deterministic relations. This method is called *prune sampling*.

Chapter 4 **Prune Sampling**

Prune sampling is a MCMC sampling method that always converges to the correct posterior distribution, even when deterministic relations are present in the BN. This chapter starts with the source of inspiration for *prune sampling* : the more generic MC-SAT algorithm. Then, mathematical notation and definitions to describe *prune sampling* are introduced. Next, a theoretical motivation and proof are presented why *prune* technique always generates a regular and reversible Markov chain with respect to the desired distribution. This chapter concludes with a discussion about the practical implementation of the *prune sampling* algorithm.

4.1 Background: MC-SAT algorithm

As chapter 3 concluded, many solutions have been proposed in the past to address the problem of trapped Gibbs samples in a subset of the state space. Notable examples are Sample Search, GiSS [20] and slice sampling methods [21, 22, 23]. The MC-SAT algorithm [6] is a special case of slice sampling and exploits the strategy of auxiliary variables in the more general framework of Markov logic networks (MLNs). Poon and Domingos show that (on MLNs) MC-SAT is a sound MCMC algorithm, meaning it generates a Markov chain which is regular and reversible, even in the presence of deterministic relations.

To use MC-SAT for BN inference, a BN needs to be converted to a so-called weighted Satisfiability Problem (SAT). This has two drawbacks. In the first place, an explicit translation of a BN to a weighted SAT problem is memory intensive. Secondly, the graphical dependencies of BNs are lost when the BN structure is translated to a SAT problem. In order not to suffer from these drawbacks, key strengths of the MC-SAT algorithm – the construction of a random sample space – is brought to the field of BN inference. In doing so, one preserves the compact and graphical structure of BNs. To explain these ideas best, first some mathematical notation and definitions about *prune sampling* need to be introduced.

4.2 Notation and definition

The key idea of *prune sampling* is straight-forward: in order to be able to select a sample of the state space uniformly, the exhaustive listing of all feasible states of the original BN is impossible (due to excessive memory and time consumption, as described in Section 2.2). Though, the exhaustive listing of all solutions of a randomly pruned BN is possible and still assures a sample to be chosen uniformly from the entire state space. In order to characterise this concept and to explain how *prune sampling* generates samples completely uniform, the following notation is introduced.

Given a BN structure $\mathcal{G}$ with n variables and corresponding CPTs. For $i = 1, \dots, n$, let l_i be the index of the labels of the CPT-entries corresponding to variable X_i, so $1 \leq l_i \leq |\mathrm{Val}(X_i)| \cdot |\mathrm{Val}(\mathrm{Pa}_{X_i})|$. Then,

$$\mathcal{C} := \{k(l_i) : c_{k(l_i)} \text{ is an entry in the CPT of variable } X_i, \text{ for } 1 \leq i \leq n\} \tag{4.1}$$

denotes the collection of all CPT-labels of $\mathcal{G}$. Here, k is an abbreviation of the name of variable X_i. The next example contributes to a better understanding of the slightly complex notation.

Example 4.1. The collection of CPT-labels $\mathcal{C}$ of the BN in Figure 3.3 contains 6 labels:

- 2 labels for the CPT of node A:
  - $A(1) = P(A = 0)$
  - $A(2) = P(A = 1)$
- 4 labels for the CPT of node B
  - $B(1) = P(B = 0|A = 0)$
  - $B(2) = P(B = 0|A = 1)$
  - $B(3) = P(B = 1|A = 0)$
  - $B(4) = P(B = 1|A = 1)$.

For completeness, the set with all CPT-labels in the BN is given by

$$\mathcal{C} = \{A(1), A(2), B(1), B(2), B(3), B(4)\}.$$

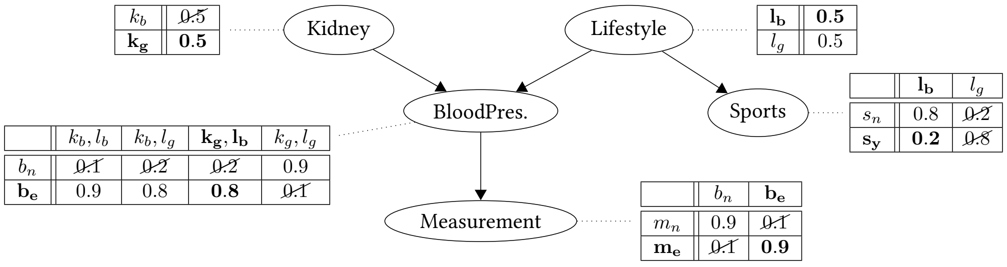
So, a state $\mathbf{x}$ of the BN corresponds to a unique collection of n CPT-entries. The collection of CPT-labels $k(l_i)$ corresponding to such a state is denoted by $\mathcal{C}_{\mathbf{x}}$. Accordingly, state $\mathbf{x} = (A = 0, B = 0)$ of the BN presented in Figure 3.3 corresponds to $\mathcal{C}_{\mathbf{x}} = \{A(1), B(1)\}$. In general, one should observe that for the probability the BN takes a specific state is the product of the CPT entries, i.e. for $i = 1, \dots, n$:

$$P(\mathbf{x}) = \prod_{k(l_i) \in \mathcal{C}_{\mathbf{x}}} c_{k(l_i)}.$$

Let C denote an arbitrary collection of CPT-labels. Then, the set of possible (not necessarily feasible) states that correspond to these CPT-labels is given by S_C. The above introduced notation and definitions are the foundations for the concept *pruning*.

Definition 4.2 (Pruning around state $\mathbf{x}$)**.** Let $\mathcal{C}_{\mathbf{x}}^{\mathrm{p}}$ be the subset of $\mathcal{C}$ that is constructed by adding each CPT-label $k(l_i) \in \mathcal{C} \setminus \mathcal{C}_{\mathbf{x}}$ with probability $1 - c_{k(l_i)}$ to the set $\mathcal{C}_{\mathbf{x}}^{\mathrm{p}}$ and with probability $c_{k(l_i)}$ not. The collection $\mathcal{C}_{\mathbf{x}}^{\mathrm{p}}$ contains the *pruned* CPT-labels.

The next example illustrates the *pruning* technique on the BloodPressure BN.



k_b	~~0.5~~
$\mathbf{k_g}$	**0.5**

$\mathbf{l_b}$	**0.5**
l_g	0.5

	k_b, l_b	k_b, l_g	$\mathbf{k_g}, \mathbf{l_b}$	k_g, l_g
b_n	~~0.1~~	~~0.2~~	~~0.2~~	0.9
$\mathbf{b_e}$	0.9	0.8	**0.8**	~~0.1~~

	$\mathbf{l_b}$	l_g
s_n	0.8	~~0.2~~
$\mathbf{s_y}$	**0.2**	~~0.8~~

	b_n	$\mathbf{b_e}$
m_n	0.9	~~0.1~~
$\mathbf{m_e}$	~~0.1~~	**0.9**

Figure 4.1: Pruned version of the BloodPressure network around the boldfaced initial state $\mathbf{x}^{(0)} = (k_g, l_b, b_e, s_y, m_e)$. Note that the lower the value of the CPT-entry, the higher the probability that the index gets pruned. One should observe that $S_{\mathcal{C}^{\text{np}}_{\mathbf{x}^{(0)}}}$ contains two feasible states: $(k_g, l_b, b_e, s_y, m_e)$ and $(k_g, l_b, b_e, s_n, m_e)$.

Example 4.3. Consider the BN in Figure 4.1. Pruning around the boldfaced initial state $\mathbf{x}^{(0)} = (k_g, l_b, b_e, s_y, m_e)$ could yield the non-crossed indices

$$\mathcal{C}^{\text{p}}_{\mathbf{x}^{(0)}} = \{K(1), L(2), BP(4), BP(5), BP(6), S(1), M(1)\}.$$

For the sake of completeness, in this example the above labels correspond to:

- $K(2) = P(K = k_g)$
- $L(2) = P(L = l_g)$
- $BP(4) = P(BP = b_n | K = k_g, L = l_g)$
- $BP(5) = P(BP = b_e | K = k_b, L = l_b)$
- $BP(6) = P(BP = b_e | K = k_b, L = l_g)$
- $S(1) = P(S = s_n | L = l_b)$
- $M(1) = P(M = m_n | B = b_n)$

Note that the lower the value of the CPT-entry, the higher the probability that the label gets pruned. The collection of CPT-labels that do not get pruned is given by $\mathcal{C}^{\text{np}}_{\mathbf{x}} := \mathcal{C} \setminus \mathcal{C}^{\text{p}}_{\mathbf{x}}$. In the situation of Example 4.3, $S_{\mathcal{C}^{\text{np}}_{\mathbf{x}^{(0)}}}$ exist of two feasible states:

- $(k_g, l_b, b_e, s_y, m_e)$;
- $(k_g, l_b, b_e, s_n, m_e)$.

Having introduced these concepts, one should note three things

1. $\mathcal{C}^{\text{p}}_{\mathbf{x}}$ is a random set;
2. $\mathcal{C}_{\mathbf{x}} \subset \mathcal{C}^{\text{np}}_{\mathbf{x}}$ and $\mathbf{x} \in S_{\mathcal{C}^{\text{np}}_{\mathbf{x}}}$;
3. the probability of generating $\mathcal{C}^{\text{p}}_{\mathbf{x}}$ and $\mathcal{C}^{\text{np}}_{\mathbf{x}}$ is given by

$$\prod_{k(l_i) \in \mathcal{C}^{\text{p}}_{\mathbf{x}}} (1 - c_{k(l_i)}) \cdot \prod_{k(l_i) \in \mathcal{C}^{\text{np}}_{\mathbf{x}} \setminus \mathcal{C}_{\mathbf{x}}} c_{k(l_i)}.$$

Due to pruning the CPT-labels, the number of feasible states in the pruned BN is much smaller in comparison to the number of feasible states in the original BN. This significant decrease of the number of feasible states makes *prune sampling* practically applicable. Assuming that sufficient memory is available, a breath first search approach can be used to list all feasible states of the pruned BN. From this collection one can easily draw a state uniformly to select the next sample.

Definition 4.4 (Uniform sampling over a set of states)**.** As defined before, $S_{\mathcal{C}^{\text{np}}_{\mathbf{x}}}$ is the set of (feasible) states corresponding to the CPT-labels which are not pruned. A uniform distribution over a set is defined as $\mathcal{U}(\cdot)$. In doing so, the uniform distribution over the states in $S_{\mathcal{C}^{\text{np}}_{\mathbf{x}}}$ is denoted as

$$\mathcal{U}(S_{\mathcal{C}^{\text{np}}_{\mathbf{x}}})(\mathbf{y}) = \frac{1}{|S_{\mathcal{C}^{\text{np}}_{\mathbf{x}}}|},$$

which represents the probability of uniformly sampling state $\mathbf{y}$ from $S_{\mathcal{C}^{\text{np}}_{\mathbf{x}}}$.

Conducting these steps – pruning, sampling uniformly, selecting a new sample – generates a sequence of samples that is able to visit the entire state space. This process is called *prune sampling* of which the pseudo-code is provided in Algorithm 6. The algorithm takes as input a BN structure $\mathcal{G}$ with corresponding CPTs, an initial configuration $\mathbf{x}^{(0)}$ and an integer T for the number of samples to be generated. The algorithm starts from the initial sample $\mathbf{x}^{(0)}$ and for $t = 1, \dots, T$ it prunes around $\mathbf{x}^{(t-1)}$ to obtain $\mathcal{C}^{\text{np}}_{\mathbf{x}^{(t-1)}}$. Then, the next sample $\mathbf{x}^{(t)}$ is chosen from $\mathcal{U}\big(S_{\mathcal{C}^{\text{np}}_{\mathbf{x}^{(t-1)}}}\big)$. Finally, the algorithm adds the new sample $\mathbf{x}^{(t)}$ to the set $\mathcal{S}$. This process yields a collection of T samples of the BN on which approximate inference techniques can be performed.

Algorithm 6 Prune sampling algorithm

```
function PruneSampling(BN, $\mathbf{x}^{(0)}$, $T$)
    $\mathcal{S} \leftarrow \{\mathbf{x}^{(0)}\}$
    for $t \leftarrow 1$ to T do
        $\mathcal{C}^{\text{p}}_{\mathbf{x}^{(t-1)}} \leftarrow$ Prune around $\mathbf{x}^{(t-1)}$
                                                        ▷ See Definition 4.2
        $\mathcal{C}^{\text{np}}_{\mathbf{x}^{(t-1)}} \leftarrow \mathcal{C} \setminus \mathcal{C}^{\text{p}}_{\mathbf{x}}$
        $\mathbf{x}^{(t)} \sim \mathcal{U}(S_{\mathcal{C}^{\text{np}}_{\mathbf{x}}})$
        $\mathcal{S} \leftarrow \mathcal{S} \cup \mathbf{x}^{(t)}$
    end for
    return $\mathcal{S}$
end function
```

Note that with strict positive probability $\mathcal{C}^{\text{np}}_{\mathbf{x}^{(t-1)}}$ contains all non-zero indices in $\mathcal{C}$. This implies that $S_{\mathcal{C}^{\text{np}}_{\mathbf{x}^{(t-1)}}}$ contains all feasible states of the BN. Hence, with positive probability a transition is possible from an arbitrary feasible state $\mathbf{x}$ to another arbitrary feasible state $\mathbf{y}$, i.e. that *prune sampling* generates a regular Markov chain. A theoretical proof that the Markov chain generated by *prune sampling* satisfies the conditions for regularity and reversibility is presented in the next subsection.

4.2.1 Regularity and reversibility

In order to transform state $\mathbf{x}$ to state $\mathbf{y}$ one should prevent to prune labels that correspond to state $\mathbf{y}$. This asks for the following notion.

Definition 4.5 (Pruning around state $\mathbf{x}$ and $\mathbf{y}$)**.** Let $\mathcal{C}^{\mathrm{p}}_{\{\mathbf{x},\mathbf{y}\}}$ be the subset of $\mathcal{C}$ that is constructed by pruning around $\mathbf{x}$ or pruning around $\mathbf{y}$ such that none of the labels corresponding to $\mathbf{x}$ and none of the labels corresponding to $\mathbf{y}$ is contained in $\mathcal{C}^{\mathrm{p}}_{\{\mathbf{x},\mathbf{y}\}}$. The collection of CPT-labels that do not get pruned is given by $\mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\}} := \mathcal{C} \setminus \mathcal{C}^{\mathrm{p}}_{\{\mathbf{x},\mathbf{y}\}}$.

The above definitions and notation provide a theoretical framework to prove the following theorems.

Theorem 4.6. *Prune sampling generates a regular Markov chain.*

Proof. For each two states $\mathbf{x}$ and $\mathbf{y}$ there are finitely many ways, $h = 1, \dots, H \in \mathbb{N}$, to create a pruned collection $\mathcal{C}^{\mathrm{p}}_{\{\mathbf{x},\mathbf{y}\},h}$ and a non-pruned collection $\mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\},h}$. So, $\mathbf{x}$ can make a transition to $\mathbf{y}$ by sampling from $\mathcal{U}(S_{\mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\},h}})$. The transition probability to transform $\mathbf{x}$ to $\mathbf{y}$ is given by

$$\begin{aligned}
\mathcal{T}_h(\mathbf{x} \to \mathbf{y}) &:= \left(\prod_{k(l_i) \in \mathcal{C}^{\mathrm{p}}_{\{\mathbf{x},\mathbf{y}\},h}} (1 - c_{k(l_i)}) \right) \cdot \left(\prod_{k(l_i) \in \mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\},h} \setminus \mathcal{C}_{\mathbf{x}}} c_{k(l_i)} \right) \cdot \mathcal{U}(S_{\mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\},h}})(\mathbf{y}) \\
&= \left(\prod_{k(l_i) \in \mathcal{C}^{\mathrm{p}}_{\{\mathbf{x},\mathbf{y}\},h}} (1 - c_{k(l_i)}) \right) \cdot \left(\prod_{k(l_i) \in \mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\},h} \setminus \mathcal{C}_{\mathbf{x}}} c_{k(l_i)} \right) \cdot \frac{1}{|S_{\mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\},h}}|}.
\end{aligned} \tag{4.2}$$

Equation 4.2 specifies the probability of pruning certain CPT-labels around $\mathbf{x}$, such that none of the CPT-labels corresponding to $\mathbf{y}$ are pruned. This is multiplied by the probability that $\mathbf{y}$ is uniformly sampled from the states corresponding to the CPT-labels that were not pruned.

The total transition probability of going from state $\mathbf{x}$ to $\mathbf{y}$ is therefore given by

$$\mathcal{T}(\mathbf{x} \to \mathbf{y}) = \sum_{h=1}^{H} \mathcal{T}_h(\mathbf{x} \to \mathbf{y}). \tag{4.3}$$

Hence, *prune sampling* generates a regular Markov chain. □

Theorem 4.7. *Prune sampling generates a reversible Markov chain.*

Proof. In order to verify reversibility it needs to be showed that the transition probability according to *prune sampling* satisfies the detailed balance equation

$$P(\mathbf{x})\mathcal{T}(\mathbf{x} \to \mathbf{y}) = P(\mathbf{y})\mathcal{T}(\mathbf{y} \to \mathbf{x}),$$

which equals

$$P(\mathbf{x}) \left(\sum_{h=1}^{H} \mathcal{T}_h(\mathbf{x} \to \mathbf{y}) \right) = P(\mathbf{y}) \left(\sum_{h=1}^{H} \mathcal{T}_h(\mathbf{y} \to \mathbf{x}) \right).$$

So, it is sufficient to show that

$$P(\mathbf{x})\mathcal{T}_h(\mathbf{x} \to \mathbf{y}) = P(\mathbf{y})\mathcal{T}_h(\mathbf{y} \to \mathbf{x}), \tag{4.4}$$

for $h = 1, \ldots, H$. The following computation shows that Equation (4.4) holds

$$\begin{aligned}
&P(\mathbf{x})\mathcal{T}_h(\mathbf{x} \to \mathbf{y}) \\
&= \frac{1}{Z} \cdot P(\mathbf{x}) \cdot \left(\prod_{k(l_i) \in \mathcal{C}^{\mathrm{p}}_{\{\mathbf{x},\mathbf{y}\},h}} (1 - c_{k(l_i)}) \right) \cdot \left(\prod_{k(l_i) \in \mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\},h} \setminus \mathcal{C}_{\mathbf{x}}} c_{k(l_i)} \right) \\
&= \frac{1}{Z} \cdot \prod_{k(l_i) \in \mathcal{C}_{\mathbf{x}}} c_{k(l_i)} \cdot \left(\prod_{k(l_i) \in \mathcal{C}^{\mathrm{p}}_{\{\mathbf{x},\mathbf{y}\},h}} (1 - c_{k(l_i)}) \right) \cdot \left(\prod_{k(l_i) \in \mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\},h} \setminus \mathcal{C}_{\mathbf{x}}} c_{k(l_i)} \right) \\
&= \frac{1}{Z} \cdot \left(\prod_{k(l_i) \in \mathcal{C}^{\mathrm{p}}_{\{\mathbf{x},\mathbf{y}\},h}} (1 - c_{k(l_i)}) \right) \cdot \left(\prod_{k(l_i) \in \mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\},h}} c_{k(l_i)} \right) \\
&= \frac{1}{Z} \cdot \left(\prod_{k(l_i) \in \mathcal{C}^{\mathrm{p}}_{\{\mathbf{x},\mathbf{y}\},h}} (1 - c_{k(l_i)}) \right) \cdot \left(\prod_{k(l_i) \in \mathcal{C}_{\mathbf{y}}} c_{k(l_i)} \right) \cdot \prod_{k(l_i) \in \mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\},h} \setminus \mathcal{C}_{\mathbf{y}}} c_{k(l_i)} \\
&= \frac{1}{Z} \cdot \left(\prod_{k(l_i) \in \mathcal{C}^{\mathrm{p}}_{\{\mathbf{x},\mathbf{y}\},h}} (1 - c_{k(l_i)}) \right) \cdot P(\mathbf{y}) \cdot \left(\prod_{k(l_i) \in \mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\},h} \setminus \mathcal{C}_{\mathbf{y}}} c_{k(l_i)} \right) \\
&= P(\mathbf{y})\mathcal{T}_h(\mathbf{y} \to \mathbf{x}),
\end{aligned}$$

where $Z = |S_{\mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\},h}}|$.

Hence, *prune sampling* generates a reversible Markov chain with respect to the desired stationary distribution P. □

The preceding two theorems are the main result of this thesis: *prune sampling* always generates a regular and reversible Markov chain. Hence, *prune sampling* always converges to the desired stationary probability distribution.

In the next section, the implementation of the *prune sampling* algorithm is discussed.

4.3 Practical implementation

In order to implement the *prune sampling* algorithm two non-trivial steps are required:

1. to generate an initial state of the BN;
2. to sample uniformly over the pruned BN, i.e. sampling from the distribution $\mathcal{U}(S_{\mathcal{C}_{\mathbf{x}}^{\mathrm{np}}})$.

In this section, it is explained how those requirements can be met and it is discussed how the MC-SAT algorithm deals with these challenges.

4.3.1 Generate initial states

To create an initial state, all unobserved variable nodes in the BN need to get assigned a state. To meet this need, MC-SAT works with local search SAT solvers [6]. The most straight-forward implementation of the *prune sampling* algorithm generates an initial state based on the commonly used *forward sampling* method (Algorithm 1). Since forward sampling is driven by CPT-entries, it is likely that a generated initial state of the BN ends up in a state with high probability of occurrence. This bias can be a disadvantage of this initialization method. For example, when Gibbs sampling is trapped, then the initial state of the BN decides in which subset of the state space Gibbs sampling will get trapped. To obtain more diversity in generated initial states of the BN, one could consider *random forward sampling.*

Definition 4.8 (Random forward sampling)**.** Suppose forward sampling is applied but instead of sampling variable X_i from $P(X_i \mid \mathrm{Pa}_{X_i})$, a random sample is chosen from the set with merely non-zero probability states $\{x_i : P(X_i = x_i \mid \mathrm{Pa}_{X_i} > 0\}$.

Random forward sampling is illustrated in the following example.

Example 4.9. In random forward sampling it is only relevant to know whether a CPT-entry is zero (0) or non-zero ($\star$). For example, in the Rain-Sprinkler BN from Figure 2.1 the CPT of variable *Grass wet* is reduced to

	$\mathrm{r_{Tr}},\ \mathrm{s_{Tr}}$	$\mathrm{r_{Tr}},\ \mathrm{s_F}$	$\mathrm{r_F},\ \mathrm{s_{Tr}}$	$\mathrm{r_F},\ \mathrm{s_F}$
$\mathrm{g_{Tr}}$	$\star$	$\star$	$\star$	0
$\mathrm{g_F}$	$\star$	$\star$	$\star$	$\star$

.

Table 4.1: In order to select a CPT-entry, the random forward sampling approach is only interested whether an entry is zero (0) or non-zero ($\star$). Consecutively, from all $\star$ CPT-entries it selects one entry uniform randomly.

Random forward sampling is only concerned whether a CPT-entry is zero or non-zero. In doing so, it samples a non-zero CPT-entry uniformly random. In this fashion, random forward sampling generates more diverse initial states than regular forward sampling. However, regarding MCMC simulations it feels intuitively smart to start sampling from an initial state with a high or the highest probability since this could be close to the desired posterior distribution. This challenge is known as finding the so-called *maximum a priori* (MAP) estimate or *most probable explanation* (MPE).

Since random forward sampling chooses CPT-entries uniform randomly, it is not the ideal approach to find a MAP estimate. However, random forward sampling does not suffer from the bias which comes along with regular forward sampling. So, a trade-off exists between bias and finding a MAP initialization. Based on this observation, the idea comes to mind to combine random and regular forward sampling to benefit from the advantages of both approaches. This approach is *hybrid forward sampling* and is defined below.

Definition 4.10 (Hybrid forward sampling)**.** Consider a hybrid approach in which at each variable X_i (with probability p) a state is assigned according to regular forward sampling or (with probability $1 - p$) a state is assigned according to random forward sampling.

In chapter 6 and 7 various experiments are conducted to test the performance of both regular and hybrid forward sampling initialization methods. In the next subsection, an implementation of the uniform sampling phase in the process of *prune sampling* is presented.

4.3.2 Sampling from the pruned network

In order to generate states nearly uniform, MC-SAT uses an intelligent heuristic based on a weighted SAT problem [24]. But, in contrast to *prune sampling* MC-SAT does not generate states completely uniform. In this subsection, it is explained how *prune sampling* generates its candidate states completely uniform. In addition, the algorithmic implementation and the forthcoming drawbacks are discussed.

Due to exhaustive listing of all feasible states of the pruned network – determining the set $\mathcal{U}(S_{\mathcal{C}_{\mathbf{x}}^{\text{np}}})$ explicitly by regular forward sampling – *prune sampling* guarantees completely uniform sampling. In order to guarentee completely uniform sampling, the exhaustive enumeration of all feasible states of the pruned network is unavoidable. For this reason, on large BNs with large pruned networks, it could be the case that one still runs into memory problems. To reduce this computational effort, another sampling heuristic method should be developed. Three suggestions are given which could provide such a solution:

1. an easy to implement solution is hybrid forward sampling with a fixed amount of generated feasible states of the pruned BN. One could consider a generated set V with predetermined fixed size of candidate samples. Subsequently, a sample from

V is chosen randomly. The size of V can be interpreted as a trade-off between uniformity and computational effort;

2. Wei, Erenrich and Selman show in [24] that one can exploit ideas from random walk based methods to obtain effective near-uniform sampling;
3. a more intelligent heuristics to obtain completely and near-uniform samples from the pruned BN could be the *simulated annealing* approach as suggested in [24].

The above suggestions are discussed in greater detail in the chapter 7. First, experiments are conducted to compare the performance of *prune sampling* to other conventional sampling methods.

Chapter 5 **Performance indicators**

In this chapter, three performance indicators are discussed that are used to characterise sampling methods. The three performance indicators are: accuracy, rate of convergence (ROC) and time consumption. Additionally, a procedure is presented to determine the rate of convergence of MCMC sampling methods in the limit of infinite simulation time, extrapolated from relatively short simulations.

5.1 Notation

First, additional mathematical notation is introduced to describe the process of approximate inference in more detail.

Consider the one-marginal probability from a CPT of interest as the expected value of a random variable Y, i.e. $\mu = \mathbb{E}[Y]$. This random variable can be estimated by using approximate sampling methods. Approximate sampling methods generate a collection of simulations on which inference statistics can be conducted (as described in section 2.2). Such a collection of simulations with size N is denoted by $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$. In the experimental analysis conducted in chapter 6, all simulations consist of $T = 25.000$ samples. One such a sample is denoted by $\mathbf{y}_i(s)$ with $1 \leq s \leq T$ and $\mathbf{y}_s \in \mathcal{Y}$. A so-called *mean trace plot* displays the probability that a value is assigned to the variable of interest as a function of the number of samples. A mean trace plot with $N = 1$ and $N = 100$ simulations is shown in Figure 5.1 and 5.2 respectively.

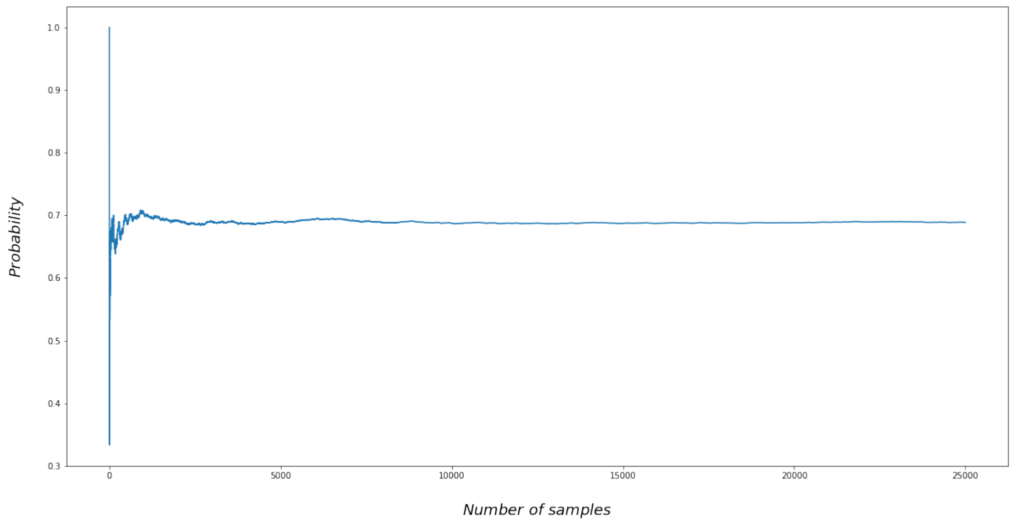


Figure 5.1: Mean trace plot shows the convergence of the posterior distribution generated by Markov chains according to the SampleSearch algorithm.

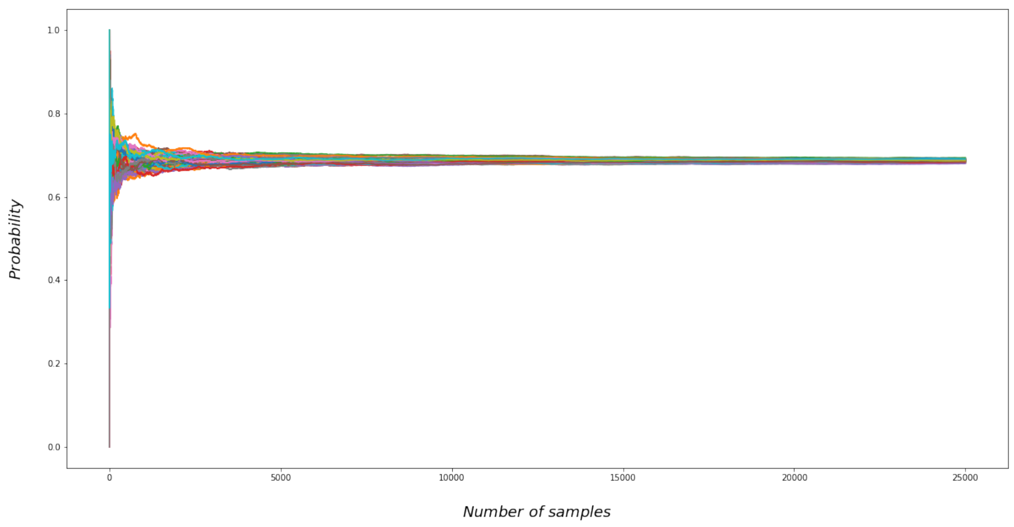


Figure 5.2: $N = 100$ simulations of $T = 25.000$ samples generated by SampleSearch. All simulations in this collection converge to the same posterior distribution.

This mathematical notation introduced in this section is used to introduce the three performance indicators.

5.2 Accuracy: the average Hellinger distance

The first performance indicator is accuracy. In this section, approaches are discussed to quantify the accuracy of sampling methods. At first, a justification is given for the use and effectiveness of (MCMC) sampling methods.

If the posterior distribution converges correctly, one can estimate the variable of interest Y easily by averaging over all simulations in the collection, i.e. $\hat{\mu}_N(t) = \frac{1}{N}\sum_{s=1}^{N}\mathbf{y}_s(t)$, especially for the last sample ($t = T$). The main justification for this type of approximation are the *laws of large numbers*. Assume that $\mu = \mathbb{E}[Y]$ exists and that $\mathbf{y}_1, \ldots, \mathbf{y}_N$ are IID. Then, the *weak law of large numbers* states that

$$\lim_{N\to\infty} \mathbb{P}\big(\big|\hat{\mu}_N - \mu\big| \leq \varepsilon\big) = 1,$$

holds for any $\varepsilon > 0$. Moreover, the *strong law of large numbers* tells us that the absolute error $|\hat{\mu}_N - \mu|$ will eventually get below ε and will always stay there

$$\mathbb{P}\big(\lim_{N\to\infty} \big|\hat{\mu}_N - \mu\big| = 0\big) = 1.$$

Both laws of large numbers tell us that (MCMC) simulation methods will eventually get arbitrary close to the probability of interest. However, the accuracy depends both on the number of simulations N and on the number of samples T. No clear guidelines exist for how large N and T need to be to guarantee a certain level of accuracy [25].

In order to characterize the accuracy of an approximate sampling method, a widely used approach to quantify the difference between the true and estimated posterior probability distribution is the *average Hellinger distance.*

Definition 5.1 (Average Hellinger Distance)**.** The Average Hellinger Distance (AHD) quantifies the closeness of two probability distributions. In the case of discrete probability distributions P and Q with binary variables (for all $i \geq 1$: $|\text{Val}(X_i)| = 2$), the AHD is defined as

$$H(P,Q) = \frac{1}{\sqrt{2}}\sqrt{\sum_{j=1}^{2}(\sqrt{p_j} - \sqrt{q_j})^2}.$$

Note that the maximum distance $H(P,Q) = 1$ occurs when for all i: $p_i = 1$ and $q_i = 0$ or vice versa.

In order to use the AHD as a measure for accuracy, it is required that the exact value μ of the marginal of interest is known. If this value is known, one of the probability distributions of the AHD equals this constant value, i.e. $Q = \mu$. The exact value can be learnt by making use of an exact inference algorithm (if computationally feasible).

In the case of calculating an overall AHD score for a collection of N simulations for a binary marginal of interest, the below approach is followed:

1. Find the exact probability of the marginal by using an exact inference algorithm (if computable). Note that the exact probability is denoted by μ;
2. Calculate for all $\mathbf{y}_s \in \mathcal{Y}$, for all $t \in \mathbf{y}_s(t)$, the AHD between the probability in this sample and the exact probability, i.e. find for $1 \leq s \leq N$ and for $1 \leq t \leq T$ $H(\mathbf{y}_s(t), \mu)$;
3. Average for every t-th step the N AHD values, i.e.

$$\text{AHD}_{\mathcal{Y}}(t) = \frac{1}{N}\sum_{i=1}^{N} H(\mathbf{y}_s(t), \mu).$$

   Note that the AHD at $t = T$ is the main AHD score of interest. However, the above procedure keeps track of the AHD scores for all $1 \leq t \leq T$ such that the development of the AHD score as a function of the number of samples can be plotted.

In chapter 6, the AHD is used to quantify the accuracy of all five approximate sampling methods. In the next section, the rate of convergence as a performance indicator is discussed.

5.3 Rate of convergence

The Rate of Convergence (ROC) is the second performance indicator by which sampling methods are characterised. A procedure is presented to obtain the ROC of sampling methods in the limit of infinite simulation time, extrapolated from relatively short simulations ($T = 25.000$).

First, one should recognize that a typical characterisation of a collection of (MCMC) simulations is the standard deviation at sample t: $\sigma^2(t) = \langle y^2(t)\rangle - \langle y(t)\rangle^2$, where

$$\langle y(t)\rangle = \frac{1}{T}\sum_{t=1}^{T}\mathbf{y}_s(t) \quad \text{and} \quad \langle y^2(t)\rangle = \frac{1}{T}\sum_{t=1}^{T}(\mathbf{y}_s(t))^2 \quad \text{for } 1 \le t \le T.$$

If one considers $\text{Var}(\mathbf{y}_s(t)) = c_t < \infty$, for all $\mathbf{y}_s \in \mathcal{Y}$ and for all $1 \le t \le T$, it follows from the unbiasedness of $\langle y(t)\rangle$ that

$$\sigma^2(t) = \text{Var}(\langle y(t)\rangle) = \frac{c_t}{t}.$$

This tells us that that σ^2 decreases with the number of samples t, which can be written as

$$\sigma_t \propto \frac{1}{\sqrt{t}}.$$

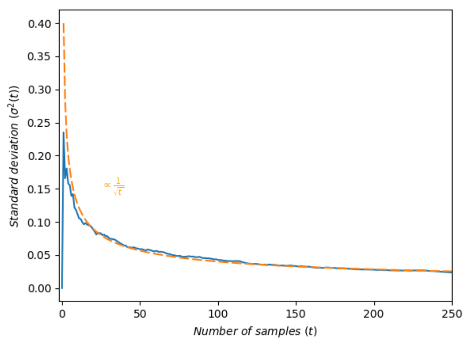
To emphasize that the rate of convergence is of order $t^{-1/2}$ and to de-emphasize σ, one can write ROC $= \mathcal{O}(t^{-1/2})$ as $t \to \infty$ [25]. Figure 5.3(a) shows that, based on the collection of $N = 100$ simulations from Figure 5.2, $\sigma(t)$ is indeed proportional to $t^{-1/2}$. Hence, one could consider $\sigma^2(t) = \alpha/\sqrt{t}$ with $\alpha \in \mathbb{R}_+$. Note that, in contrary to c_t, α is independent of t and is therefore a feature of the entire collection of (MCMC) simulations. Or to put it differently, the proportionality constant α quantifies the ROC of the (MCMC) sampling technique that belongs to the convergence class $\mathcal{O}(t^{-1/2})$. Therefore, different (MCMC) sampling techniques can be characterised by their ROC and are therefore a useful performance indicator to compare various methods.

Next, a procedure is introduced to determine this constant α for a collection of (MCMC) simulations. First of all, $\log\sigma(t)$ can be plotted versus $\log t$. The result is shown in Figure 5.3(b) and straight away a candidate proportionality constant α' can be fitted to the asymptotic linear behavior of $\log\sigma(t)$ (orange dashed line). Both, in Figure 5.3(a) and (b), $\alpha' = 0.38$. Though, on the interval $10^0 < t < 10^1$, it can be seen that $\sigma(t)$ (the blue line) does not behave as $t^{-1/2}$ (the orange dashed line). Ideally, in determining α for the asymptotic convergence of a (MCMC) method, this non-representative region is not taken into consideration. This region can be ignored in a sophisticated way. In order to do so, an auxiliary polynomial expansion is introduced such that

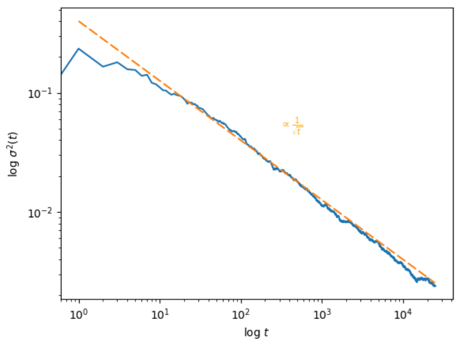
$$\sigma^2(t) = \frac{\alpha}{\sqrt{t}}(1 + \beta_1 t^{-\delta} + \beta_2 t^{-2\delta} + \beta_3 t^{-3\delta} + \ldots)\,. \tag{5.1}$$

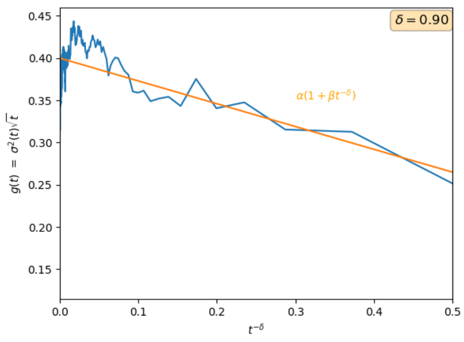
Due to the prospect of overfitting, Equation 5.1 is reduced to

$$g(t) = \sigma^2(t)\sqrt{t} \approx \alpha(1 + \beta t^{-\delta}) = f(t). \tag{5.2}$$



(a) The standard deviation $\sigma^2(t)$ of a collection of 100 SampleSearch samples (from Figure 5.2) decreases as $t^{-1/2}$, subject to the number of samples t. Hence, this sampling method belongs to the convergence class $\mathcal{O}(t^{-1/2})$.





(b) Plotting the log of the standard deviation – $\log \sigma^2(t)$ – versus the log of the number of samples – $\log t$ – yields a linear function, which can be approximated. Using this plot, a rough estimate of α' (candidate value for α) can be made. In this plot, $\alpha' = 0.38$ is taken and hence $0.38/\sqrt{t}$ defines the orange line.

(c) To determine α with greater precision, the interval $[10^0, 10^1]$ from Figure 5.3(b) needs to be ignored. This can be done by introducing a polynomial expansion to approximate the linear log plot as $\sigma^2(t)\sqrt{t} \approx \alpha(1 + \beta t^{-\delta})$ (blue line). α turns out to be the intersection with the y-axis. Hence, approximating this function linearly (orange line), one can retrieve (for $\delta = 0.90$) that $\alpha \approx 0.40$.

Figure 5.3: Asymptotic behavior of the standard deviation of (MCMC) sampling methods can be used to characterise the rate of convergence.

One should note that $g(t^{-\delta})$ in Equation 5.2 is a linear function of which α is the intersection with the y-axis. Based on the collection of $N = 100$ SampleSearch simulations from Figure 5.2, in Figure 5.4 $g(t^{-\delta})$ is plotted for $\delta = 0.4, \ldots, 0.9$. From Figure 5.4, $\delta = 0.9$ is considered to be the transformation with the most linear appearance. So, a linear function f – the orange line in Figure 5.3(c) – is fit to $g(t^{-\delta})$ in order to determine the intersection with the y-axis, i.e. the proportionality constant α. The result is depicted in Figure 5.3(c) and it turns out that $\alpha \approx 0.40$.

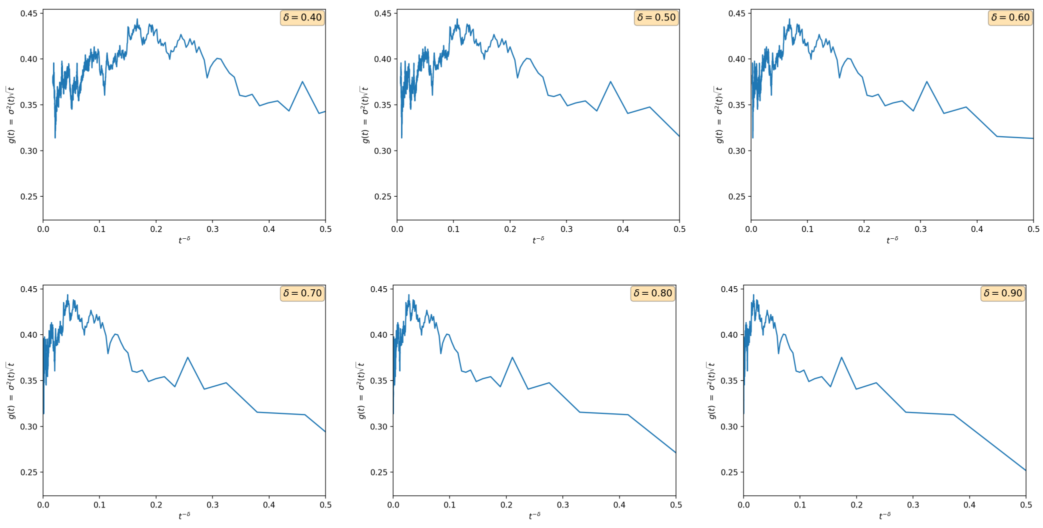


Figure 5.4: Linear appearance of $g(t^{-\delta})$ for different values of δ.

The transformation from the number of samples t to $t^{-\delta}$ can be regarded as a *multiplicative inversion*. Since t is multiplicatively inversed by $t^{-\delta}$ according to parameter δ, the right side of the plot in Figure 5.3(b) is displayed at the left side of Figure 5.3(c). Besides, where the intervals $[10^0, 10^1]$, $[10^1, 10^2]$, $[10^2, 10^3]$, $[10^3, 10^4]$ in Figure 5.3(b) on a logarithmic scale are of equal length (equidistant), the projection of those intervals in Figure 5.3(c) is not equidistant. The ratio of the lengths of those intervals is unequally distributed and depends on the value of δ, i.e. the interval $[10^0, 10^1]$ is represented more prominently as δ increases. In this manner, the interval $[10^0, 10^1]$ on which the double log plot does not appear to be linear (in Figure5.3(b)), is stretched out such that it becomes more linear (in Figure5.3(c)). The already linear appearing intervals (when t is large) are downgraded in appearance. In this manner, the value of α can be estimated more carefully.

The multiplicative inversion of the intervals $[10^0, 10^1], \ldots, [10^3, 10^4]$ to the interval $[0, 1]$ according to $\delta = 0.1, \ldots, 0.9$ are displayed in Figure 5.5.

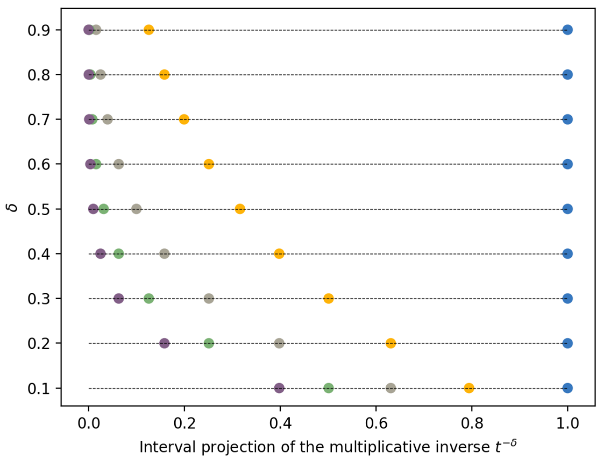


Figure 5.5: Multiplicative inverse $t^{-\delta}$ of the border points of the intervals $t \in [10^0, 10^1], [10^1, 10^2], [10^2, 10^3], [10^3, 10^4]$ for $\delta = 0.1, \ldots, 0.9$. The interval $[10^0, 10^1]$ is represented more prominently as δ increases. Hence, the value of α on this interval could be estimated more carefully.

The above procedure to characterize the ROC of a (MCMC) sampling method in the limit of infinite simulation time, extrapolated from relatively short simulations is summarized:

- generate a collection of N (MCMC) simulations, as in Figure 5.2;
- determine the standard deviation $\sigma^2(t)$ between the N simulations for every sample t, as in Figure 5.3(a);
- plot $\log \sigma^2(t)$ versus $\log t$, as the blue line in Figure 5.3(b);
- determine a candidate proportionality constant α' by fitting $\alpha'/\sqrt{t}$ to the above log plot, as the dashed orange line in Figure 5.3(b);
- plot $g(t^{-\delta})$ as in Equation 5.2 for various values of $0 < \delta < 1$, as in Figure 5.4;
- select a δ for which the multiplicative inversed interval $[10^0, 10^1]$ has the most linear appearance and fit a linear function f (the orange line in Figure 5.3(c)) to this inversed function (the blue line in Figure 5.3(c) to approximate the intersection with the y-axis (α);
- adjust α' to α.

In the next section, the last performance indicator is discussed.

5.4 Time consumption

The third performance indicator is time consumption. In order to characterise different (MCMC) sampling methods in terms of time consumption, common grounds need to be found to compare different methods.

This is done by measuring what amount of samples a (MCMC) method needs to achieve a certain level of closeness in the collection of $N = 100$ simulations, i.e. for what is the smallest $t \in (0, 25.000)$ such that $\sigma^2(t) \leq 0.01$?

With the knowledge about the ROC proportionality constant α for different (MCMC) methods (as discussed in section 5.3), the amount of samples a method needs to achieve $\sigma^2 = 0.01$ can be calculated, namely

$$0.01 = \sigma^2(t) = \frac{\alpha}{\sqrt{t}}. \tag{5.3}$$

So, if Gibbs sampling is used on, for example, the Asia BN with 0% evidence available, where α turns out to be 0.52, one needs

$$0.01 = \frac{0.52}{\sqrt{t}} \qquad \Longrightarrow \qquad t = \left(\frac{0.52}{0.01}\right)^2 = 2.704 \text{ samples} \tag{5.4}$$

to achieve $\sigma^2 = 0.01$. Hence, it is sufficient to measure the amount of time it takes for Gibbs sampling to generate this number of samples, including the generation of an initial state of the BN at the start. The time consumption is measured by making use of the `time` library in Python.

As illustrated in Example 3.14, (MCMC) approximate inference methods do not always converge towards the same probability distribution. In such a case, no $t \in (0, 25.000)$ such that $\sigma^2(t) \leq 0.01$ and no ROC constant α exists. Hence, the ROC and time-consumption are not relevant to characterise for those sampling method.

Chapter 6 **Results**

In this chapter the performance of *prune sampling* is compared to four conventional sampling methods. First, details are given about the context and conditions for the experimental research. Then, the main pitfall of Gibbs sampling is demonstrated by various experiments. Consecutively, for 3 BNs from 4 benchmark domains with an increasing amount of deterministic relations, the performance of the sampling methods in terms of accuracy, rate of convergence and time consumption are characterised.

6.1 Experiments

Prune sampling is compared to four widely used sampling methods (backward sampling, likelihood weighting, SampleSearch and Gibbs sampling) on three performance indicators (accuracy, rate of convergence and time consumption). Since pruning is devised to deal with deterministic relations in BNs, experiments are conducted on BNs with gradually increasing rates of determinism. The sampling methods are used to approximate one-variable marginals on small, medium and large BNs from four benchmark domains: simple deterministic-, block shaped-, Grid- and real world BNs. GeNIe decision modeling software is used to customize BNs in `.xdsl` format to conduct experiments on. *Prune sampling* is implemented in Python such that BNs created in GeNIe can be used as input. It turned out that the implementation of Gibbs sampling from the Python PyMC package is unreliable. Instead, all four sampling methods different than *prune sampling* are used from the ProbCog GitHub repository: a toolbox for statistical relational learning and reasoning from the Technical University Münich [26]. It is noteworthy that the input format of BNs for the ProbCog software is `.xml`. Hence, a tool needs to be used to transform BNs formatted in `.xdsl` to a BN in `.xml` format. All BNs used in this study can be found for free online in the UAI or bnlearn Bayesian network repository. Results are created without thinning [1] and if a burn-in period [2] is used, this can be mentioned from the starting number of the 'number of samples' at the x-axis. The experiments are conducted on an Intel(R) Core(TM) i5-5300 CPU 2.30GHz core machine with 8 GB RAM, running operating system Windows 10.

6.2 Pitfalls of Gibbs sampling

As illustrated in Example 3.14, the Markov chain generated by Gibbs sampling can get trapped in a subset of the state space when deterministic relations are present in a BN. In this section, this phenomena is investigated in greater detail. The accuracy

[1] *thinning*: in order to decrease auto-correlation between the samples one could use only every q-th generated sample

[2] *burn-in period*: throwing away the first r iterations at the beginning of a MCMC simulation

of Gibbs and *prune sampling* is compared on various deterministic BNs and so-called *block-shaped* BNs. Eventually, it is shown that *prune sampling* is able to move around the entire state space in all circumstances on all BNs when Gibbs sampling is not.

6.2.1 Simple deterministic network

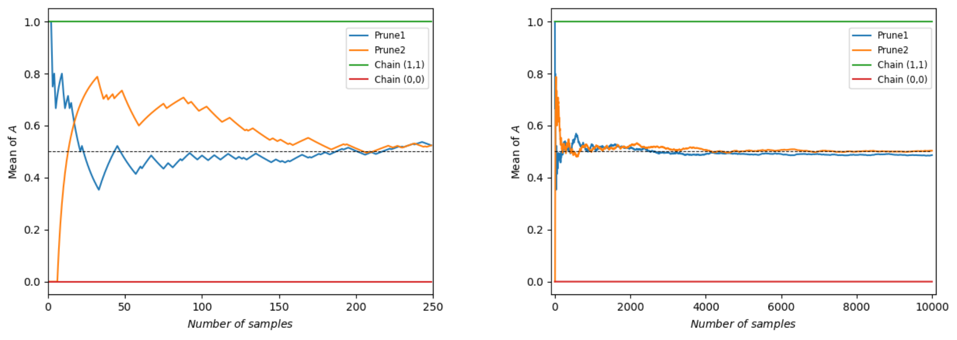
Figure 6.1 displays the convergence of the mean of variable A according to samples generated by Gibbs and *prune sampling*. The horizontal red and green line represent the trapped chains generated by Gibbs sampling, i.e. displaying $P(A = 1) = 0$ and $P(A = 1) = 1$ respectively. The converging blue and orange lines indicate the approximations of Markov chains generated by *prune sampling*. Example 3.14 elaborates on the cause why Gibbs sampling fails to converge. Why *prune sampling* is able to converge to the correct distribution follows from the next example.

Example 6.1. Without loss of generality, let's consider the initial state to be $\mathbf{x}^{(0)} = (0, 0)$. Following the *pruning* procedure from Algorithm 6 one obtains

- $\mathcal{C}_{\mathbf{x}} = \{A(1), B(1)\}$.
- Two cases can occur, both with probability 0.5, either
  - $A(2)$ does get pruned, which results in: $\mathcal{C}^{\mathrm{p}}_{\mathbf{x}^{(1)}} = \{A(2), B(2), B(3)\}$
  - $A(2)$ does not get pruned, which results in: $\mathcal{C}^{\mathrm{p}}_{\mathbf{x}^{(1)}} = \{B(2), B(3)\}$
- Following this structure of case distinction, the non-pruned sets could become
  - $A(2)$ does get pruned: $\mathcal{C}^{\mathrm{np}}_{\mathbf{x}^{(1)}} = \{A(1), B(1), B(4)\}$ (1)
  - $A(2)$ does not get pruned: $\mathcal{C}^{\mathrm{np}}_{\mathbf{x}^{(1)}} = \{A(1), A(2), B(1), B(4)\}$. (2)
- Now, feasible states can be determined from which the next state will get sampled uniformly
  - (1) gives: $S_{\mathcal{C}^{\mathrm{np}}_{\mathbf{x}}} = \{(c_{A(1)} = 0, c_{B(1)} = 0)\}$
  - (2) gives: $S_{\mathcal{C}^{\mathrm{np}}_{\mathbf{x}}} = \{(c_{A(1)} = 0, c_{B(1)} = 0), (c_{A(2)} = 1, c_{B(4)} = 1)\}$.
- From (1), with probability 1, one finds feasible state $(0, 0)$. From (2), with probability 0.5, one finds either feasible state $(0, 0)$ or $(1, 1)$. Hence, the probability to get from state $(0, 0)$ to $(0, 0)$ is given by

$$Q((0, 0) \to (0, 0)) = 0.5 \cdot 0.5 + 0.5 \cdot 1 = \frac{3}{4}.$$

This implies that *prune sampling* is able to make a transition from $(0, 0)$ to $(1, 1)$ and the other way around from $(1, 1)$ to $(0, 0)$. In Figure 6.1, it can be seen that *prune sampling* – from both initial states $(0, 0)$ and $(1, 1)$ – is able to move around the entire state space freely, i.e. it is able to converge to the correct mean. The horizontal line at



(a) 250 samples prune vs Gibbs (b) 10.000 samples prune vs Gibbs

Figure 6.1: Illustration of superior performance of *prune sampling* . Due to the deterministic relation in the BN given in Example 3.14, Gibbs sampling is trapped in the chain $(0, 0)$ or $(1, 1)$. Hence, doing inference on these Gibbs samples yield $P(A = 0) = 0$ or $P(A = 0) = 1$ (red and green line). As a consequence of the regular and reversible Markov chain generated by *prune sampling* (blue and orange line), this Markov chain is able to move around the entire state space and therefore converges to the correct probability distribution $P(A = 0) = 0.5$.

0.5 represents the exact probability of variable $A = 0$.

The above addressed problem is known as *bad mixing*. The colloquial term *mixing* is used in the field of MCMC simulations to indicate whether a Markov chain is able to visit the whole state space or not. Techniques to improve mixing of MCMC methods have been proposed [21, 23, 27]. For example, one could sample (according to the normal Gibbs sampling procedure) two or more variables at the same time from their joint distribution conditioned on all other variables. This is called *blocked Gibbs sampling*. The pairwise deterministic relations between nodes would disappear. Though, blocked Gibbs sampling is not the solution to all its limitations. As will be discussed in the next subsection.

6.2.2 Block shaped network

Simple deterministic networks are not the only class of BNs that prevent Gibbs sampling of converging to the correct posterior distribution. The following example provides a different class of BNs that results in trapped Markov chains.

Example 6.2. Consider the BN $X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_n$, where each $X_i \in \{0, 1, 2, 3\}$ for $1 \leq i \leq n$. Let X_1 be uniformly distributed and for $i \geq 2$ let each X_i be conditionally distributed according to a so-called *block-shaped* distribution, of which the CPTs of all variable X_i for $2 \leq i \leq n$ adhere to the structure of Table 6.1. Due to the structure of its CPTs, this type of BNs is called a *block-shaped BN*. If one applies Gibbs sampling on such a block-shaped BN, the Markov chain would not converge to the correct posterior

	$X_{i-1}=0$	$X_{i-1}=1$	$X_{i-1}=2$	$X_{i-1}=3$
$X_i=0$	0.5	0.5	0	0
$X_i=1$	0.5	0.5	0	0
$X_i=2$	0	0	0.5	0.5
$X_i=3$	0	0	0.5	0.5

.

Table 6.1: Block shaped CPT

distribution. To understand why, one should observe that only two possible initial states are possible, either

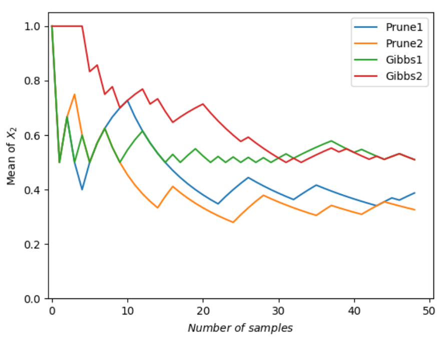
- $X_1^{(0)} \in \{0,1\}$;
- $X_1^{(0)} \in \{2,3\}$.

Suppose $X_1^{(0)} = 0$ and consider the fixed (topological) order $(X_1, \dots, X_n)$. In this case, Gibbs sampling returns

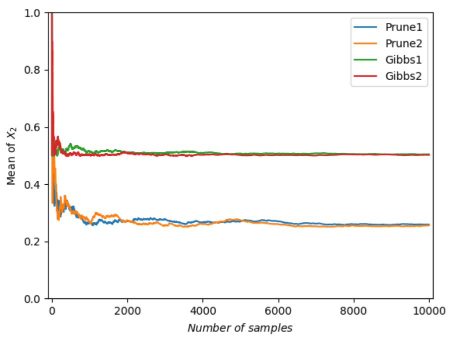
$$\mathbf{x}^{(0)} = (X_1^{(0)} = 0, \dots, X_n^{(0)} = 0), \text{ i.e. } X_i^{(0)} = 0 \text{ for } 1 \leq i \leq n.$$

After each iteration Gibbs sampling is able to make the move from block $\{0,1\}$ to $\{2,3\}$ or the other way around, i.e. $P(X_1^{(t+1)}|X_2^{(t)} = 0, \dots, X_n^{(t)} = 0)$ could select $X_1^{(t+1)} \in \{0,1\}$ or $X_1^{(t+1)} \in \{2,3\}$ both with probability 0.5. Though, in both cases $X_i^{(t+1)}$ is trapped again either in $\{0,1\}$ to $\{2,3\}$ for $i \geq 2$. So, despite the fact that Gibbs sampling is able to change blocks, the posterior distribution $P(X_2)$ converges to 0.5 instead of $P(X_2) = 0.25$.

In Figure 6.2, the red and green line represent the Markov chain generated by Gibbs sampling being trapped in block $\{0,1\}$ or $\{2,3\}$ respectively. They both find $P(X_i = 0) = P(X_i = 1) = 0.5$ and $P(X_i = 2) = P(X_i = 3) = 0.5$ respectively. In the same Figure it can be seen that the Markov chain generated by *prune sampling* – the blue and orange line – is again able to move freely around the entire state space and therefore converges, for $i \geq 2$, to the correct probability $P(X_i = j) = 0.25$, where $j \in \{0,1,2,3\}$.



(a) 50 samples prune vs Gibbs



(b) 10.000 samples prune vs Gibbs

Figure 6.2: *Prune sampling* can be superior to Gibbs sampling, even in the absence of deterministic relations. A non-deterministic block shaped CPT – as presented in Table 6.1 – can prevent a Markov chain generated by Gibbs sampling of visiting the entire state space. In this example, being trapped in the subset $\{0, 1\}$ or $\{2, 3\}$ both yield the probability 0.5 of assigning $0, 1$ or $2, 3$ to variable X_i (red and green line respectively). *Prune sampling* generates a Markov chain that is regular and reversible and therefore can move around freely through the whole state space. Hence, *prune sampling* converges to the uniformly probability 0.25 of assigning value $0, 1, 2$ or 3 to variable X_i (blue and orange line).

6.3 Performance on Benchmark Bayesian networks

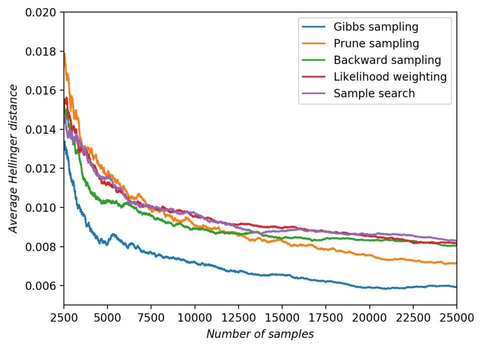
In this section, results are presented about the performance of *prune sampling* in comparison to the four other sampling methods on benchmark BNs. Experiments are conducted on BNs with various rates of determinism. Since *prune sampling* is developed to deal with deterministic relations, it is expected that *prune sampling* performs best on this type of BNs. Besides, experiments are run as well on real world BNs, with and without available evidence. As such, *prune sampling* is tested both as a specialized method for deterministic BNs and as an all-round sampling method that is able to deal with more generic BNs.

First, experiments are conducted on real-life BNs without any evidence available. Consecutively, experiments are run on the same real-life BNs with 25% available evidence. Lastly, experiments are conducted on so-called *Grid* BNs, which consist either for 25% or for 50% of deterministic relations. Grid BNs are introduced formally in section 6.3.3. For all experiments it holds that Markov chains are allowed to iterate $T = 25.000$ steps. One such a simulation is repeated $N = 100$ times.

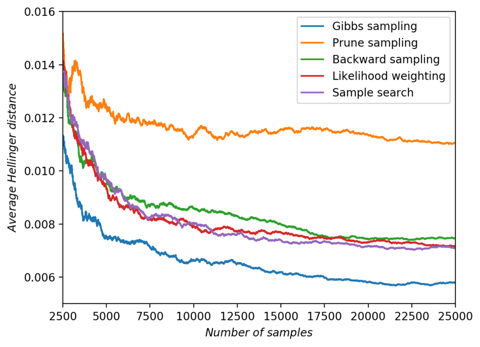
The class real-life BNs consist of three types of BNs: the small BN (8 nodes, 18 parameters) Asia [28], the medium BN (37 nodes, 509 parameters) Alarm [29] and the large BN (76 nodes, 574 parameters) Win95pts [30]. The terminology 'small', 'medium' and 'large' BNs is inspired by the Bayesian network repository on `www.bnlearn.com`.

6.3.1 Accuracy on real world Bayesian networks with 0% available evidence

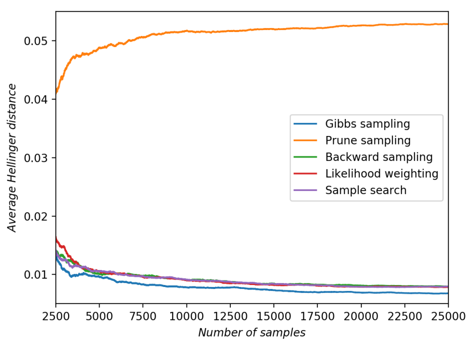
On the Asia, Alarm and Win95pts BNs with 0% available evidence, samples are generated to query the CPTs of respectively *Dyspnea*, *BloodPressure* and *Problem1*. Since *prune sampling* is devised to deal with deterministic relations, it is not expected that *prune sampling* will be the best performing sampling method on this class of BNs.



(a) Asia 0% evidence



(b) Alarm 0% evidence



(c) Win95pts 0% evidence

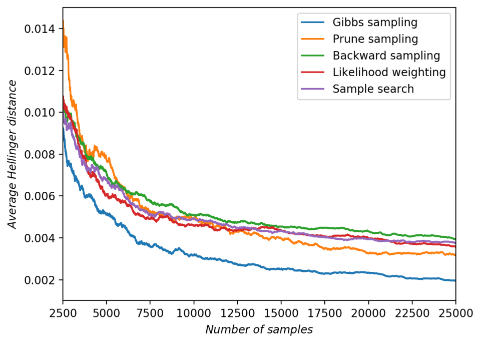
Figure 6.3: Mixed performance of *prune sampling* – in terms of accuracy – on real world BNs without available evidence. Since *prune sampling* is devised to deal with deterministic BNs, underperformance of the pruning technique on these type of general-purpose networks can be expected.

Figure 6.3 shows the performance of Gibbs sampling, backward sampling, likelihood weighting, SampleSearch and *prune sampling*. On the small Asia BN *prune sampling* is a competitive sampling method and reaches approximately 0.008 AHD (after $T = 25.000$ iterations, averaged over $N = 100$ simulations). Despite *prune sampling* beats backward sampling, likelihood weighting and SampleSearch on the Asia BN, Gibbs sampling does outperform *prune sampling* .

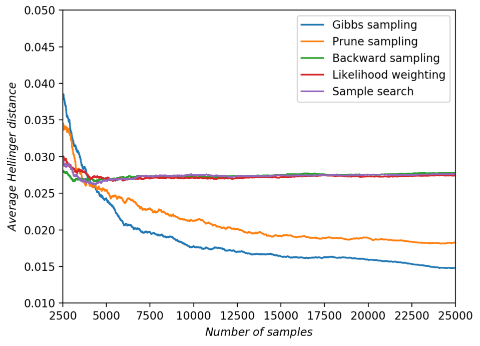
Structural underperformance of *prune sampling* is visible on both the Alarm and Win95pts BN. For all sample $2.500 \leq t \leq 25.000$ *prune sampling* has a higher AHD than all other (MCMC) sampling methods. For the Win95pts BN, the posterior distribution of *prune sampling* even diverges to $\mathbb{P}(\text{Problem1} = True) = 0.585$ where the other sampling methods converge towards the exact probability 0.541053.

6.3.2 Accuracy on real world Bayesian networks with 25% available evidence

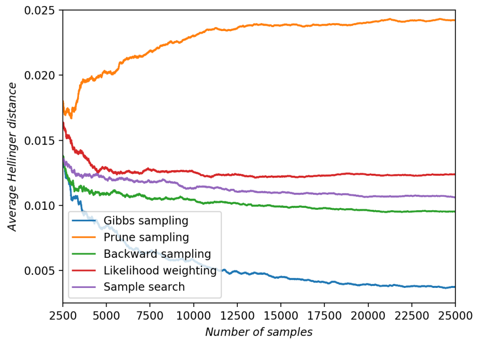
On the same real-world BNs as in the previous subsection, now experiments are conducted in which 25% of the nodes in the BN have a fixed state, i.e. evidence is available for those nodes in the network. For the same queries as before, the ability of *prune sampling* to deal with available evidence is tested.



(a) Asia 25% evidence



(b) Alarm 25% evidence



(c) Win95pts 25% evidence

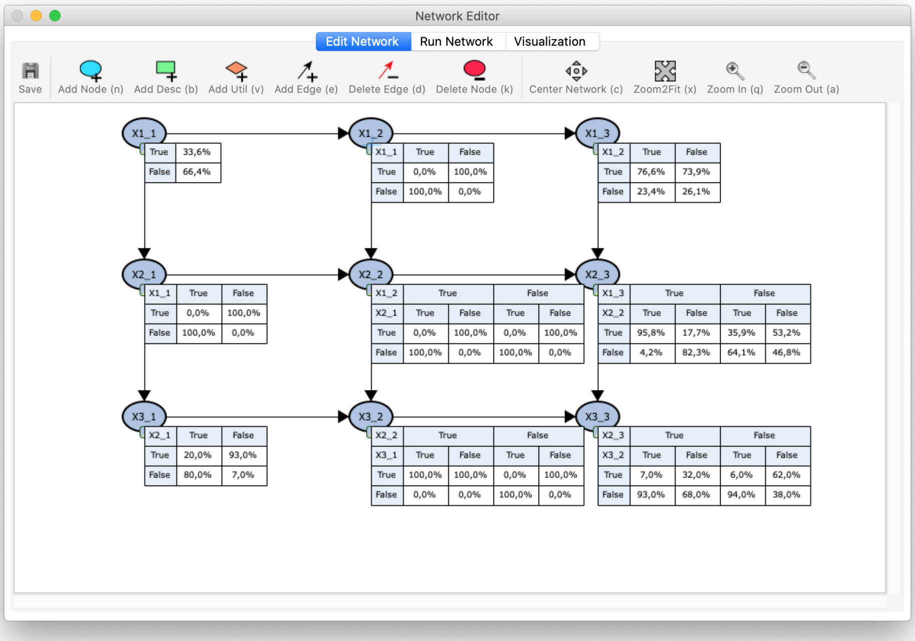
Figure 6.4: Compared with the results in section 6.3.1, *prune sampling* starts to become more competitive in terms of accuracy as more evidence is available. Though, on the Win95pts BN *prune sampling* underperforms significantly.

Figure 6.4 displays the performance of the sampling methods on the Asia, Alarm and Win95pts BNs with 25% evidence available. On the Asia BN all five sampling methods show strong performance: on average (averaged over $N = 100$ simulations, for $T = 25.000$ iterations) all methods reach at least 0.006 AHD. On the Alarm BN, the generated samples have more difficulties to reach high accuracy. Though, *prune* *sampling* starts to become competitive and even outperforms backward sampling, likelihood weighting and SampleSearch which are somehow stuck around 0.03 AHD. However, again strong underperformance of *prune sampling* becomes clear on the Win95pts BN. *Prune sampling* reaches approximately 0.024 AHD. In contrast, the other four methods reach at least 0.013 and Gibbs sampling even achieves less than 0.005 AHD.

6.3.3 Accuracy on Grid Bayesian networks

The next class on which *prune sampling* is tested is the so-called Grid BNs. This type of BNs is developed by Sang, Beame, and Kautz [31] in order to create benchmark problems that are intrinsically hard due to their rigid structure and many deterministic relations. Due to these characteristics, this is the class of BNs on which *prune sampling* is expected to perform best.

The variables of a $n \times n$ Grid BN are denoted by $X_{i,j}$, where $1 \leq i, j \leq n$. Each variable $X_{i,j}$ has parents $X_{i-1,j}$ and $X_{i,j-1}$, for $2 \leq i, j \leq n$. So, $X_{1,1}$ is a source variable and $X_{n,n}$ is a sink variable. The query of interest is to compute the marginal probability of the sink variable $X_{n,n}$. The *deterministic ratio* is the fraction of the nodes in the BN that have solely deterministic relations in its CPT. An example of a Grid 3×3 BN with (an approximately) 50% determinism is displayed in Figure 6.5.



True	33,6%
False	66,4%

X1_1	True	False
True	0,0%	100,0%
False	100,0%	0,0%

X1_2	True	False
True	76,6%	73,9%
False	23,4%	26,1%

X1_1	True	False
True	0,0%	100,0%
False	100,0%	0,0%

X1_2	True		False	
X2_1	True	False	True	False
True	0,0%	100,0%	0,0%	100,0%
False	100,0%	0,0%	100,0%	0,0%

X1_3	True		False	
X2_2	True	False	True	False
True	95,8%	17,7%	35,9%	53,2%
False	4,2%	82,3%	64,1%	46,8%

X2_1	True	False
True	20,0%	93,0%
False	80,0%	7,0%

X2_2	True		False	
X3_1	True	False	True	False
True	100,0%	100,0%	0,0%	100,0%
False	0,0%	0,0%	100,0%	0,0%

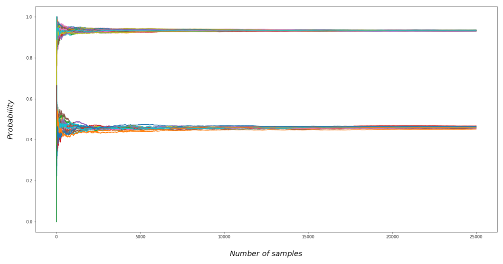
X2_3	True		False	
X3_2	True	False	True	False
True	7,0%	32,0%	6,0%	62,0%
False	93,0%	68,0%	94,0%	38,0%

Figure 6.5: Example of a Grid 3×3 BN with (approximately) a 50% deterministic ratio.

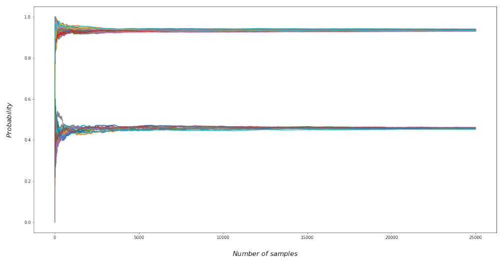
Experiments are conducted on Grid 3×3, 5×5 and 8×8 BNs, both with 25% and 50% deterministic ratio. As explained in Example 3.14 and illustrated by experiments in Example 6.2, one should take into consideration that Gibbs sampling could be incompetent to deal with Grid BNs due to the presence of deterministic relations. This scenario unfolds in all mean trace plots in Figure 6.6. For all Grid 3×3, 5×5 and 8×8 BNs, both with 25% and 50% deterministic ratio, Gibbs sampling fails to converge to the correct posterior distribution. Depending on which subset of the state space Gibbs sampling is locked in, it converges to different incorrect posterior distributions.

In Figure 6.7 the performance of the other (MCMC) sampling methods – without Gibbs sampling – is shown. Note that due to the absence of Gibbs sampling, *prune sampling* is plotted in blue. It becomes clear that, in contrary to our expectations, on none of the Grid BNs *prune sampling* is a competitive sampling method compared to the other sampling methods. Backward sampling, likelihood weighting and SampleSearch all tend to converge towards 0.01 AHD on all six Grid BNs. On the Grid 3×3 BNs, *prune sampling* is rather competitive in the sense that its AHD is close to the other AHDs, i.e. approximately a 0.002 to 0.003 difference in AHD. On the Grid 5×5 BNs, the difference in accuracy widens further: approximately a 0.004 to 0.006 difference in AHD. On the 8×8 BNs, *prune sampling* performs far below expectation by reaching approximately only 0.045 AHD. In contrary, all other (converging) methods reach at least 0.01 AHD.

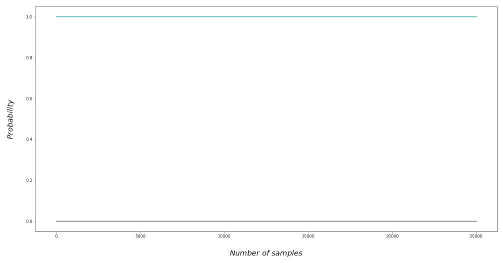
In the next section, the performance of the sampling methods in terms of the rate of convergence is presented.



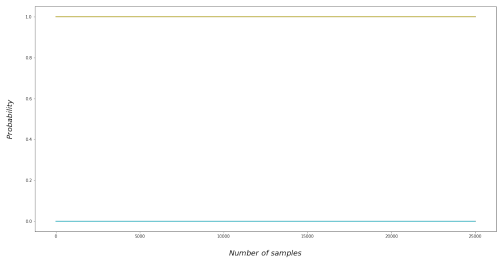
(a) 100 Gibbs simulations on the Grid 3×3 BN with 25% determinism fail to converge to the correct posterior probability: 0.8373



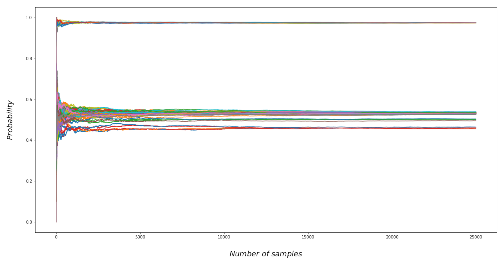
(b) 100 Gibbs simulations on the Grid 3×3 BN with 50% determinism fail to converge to the correct posterior probability: 0.8719



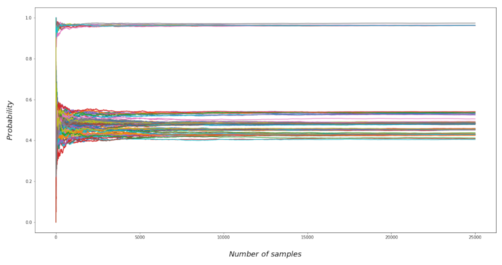
(c) 100 Gibbs simulations on the Grid 5×5 BN with 25% determinism fail to converge to the correct posterior probability: 0.5678



(d) 100 Gibbs simulations on the Grid 5×5 BN with 50% determinism fail to converge to the correct posterior probability: 0.5780

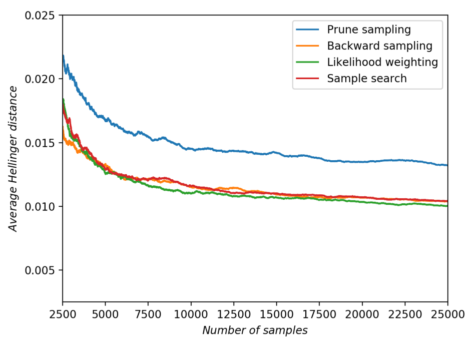


(e) 100 Gibbs simulations on the Grid 8×8 BN with 25% determinism fail to converge to the correct posterior probability: 0.5731

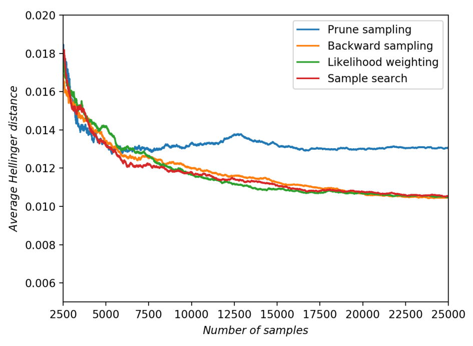


(f) 100 Gibbs simulations on the Grid 8×8 BN with 50% determinism fail to converge to the correct posterior probability: 0.5310

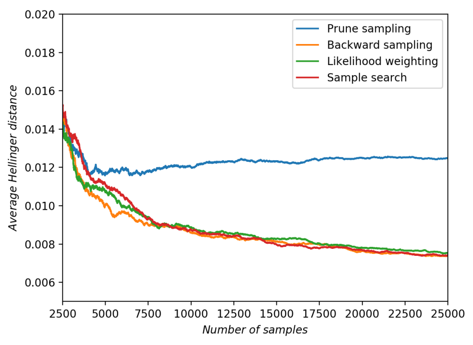
Figure 6.6: Failure of Gibbs sampling on deterministic Grid BNs. Due to being trapped in different subsets of the state space, the collection of 100 Gibbs iterations diverges to different posterior distributions.



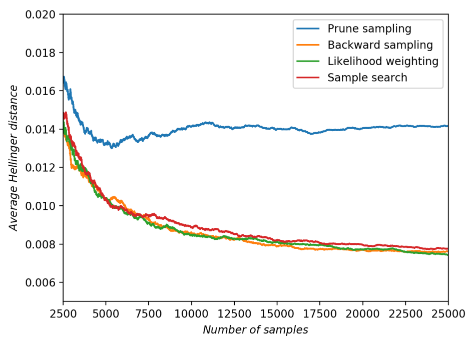
(a) Grid 3 × 3 network 25% determinism



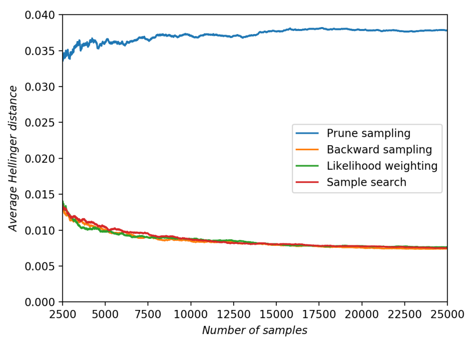
(b) Grid 3 × 3 network 50% determinism



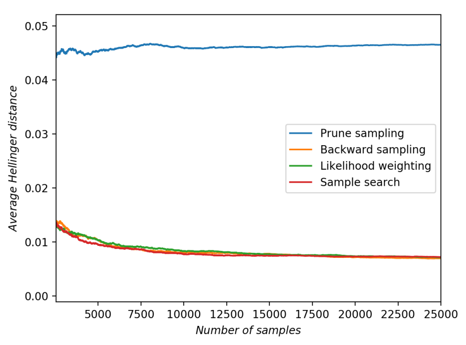
(c) Grid 5 × 5 network 25% determinism



(d) Grid 5 × 5 network 50% determinism



(e) Grid 8 × 8 network 25% determinism



(f) Grid 8 × 8 network 50% determinism

Figure 6.7: Underperformance of *prune sampling* – in terms of accuracy – on all Grid BNs. Since *prune sampling* is devised to deal with determinism and either 25% or 50% of these BNs' CPTs consist of deterministic relations, this performance is far below expectation.

6.4 Rate of Convergence

Following the procedure as presented in section 5.3, the proportionality constants of the ROC of the sampling methods on all 12 benchmark networks can be computed. The collections of samples which are used to create the accuracy plots in Figure 6.3, 6.4 and 6.7 are used to find those ROC values. For collections generated by Gibbs sampling that diverge on Grid BNs, the proportionality constants of the ROC are omitted. The result is presented in Table 6.2.

	Sampling method				
Bayesian network	Gibbs	Prune	Backward	Likelihood	SampleSearch
Asia_ev0	0.52	0.81	0.51	0.49	**0.48**
Alarm_ev0	**0.43**	0.79	0.48	0.49	0.49
Win95pts_ev0	0.48	0.59	0.48	**0.46**	0.51
Asia_ev25	**0.47**	0.77	0.50	0.48	0.48
Alarm_ev25	**0.40**	0.49	0.52	0.46	0.45
Win95pts_ev25	**0.45**	0.50	0.46	0.46	0.48
Grid 3x3_det25	x	0.56	**0.36**	0.37	0.39
Grid 5x5_det25	x	0.54	**0.46**	0.50	0.51
Grid 8x8_det25	x	0.50	0.47	0.47	**0.45**
Grid 3x3_det50	x	0.62	0.33	0.34	**0.32**
Grid 5x5_det50	x	0.55	**0.48**	0.49	0.51
Grid 8x8_det50	x	0.53	0.51	0.51	**0.48**

Table 6.2: ROC proportionality constants of *prune sampling* are always higher than the proportionality constants of backward sampling, likelihood weighting, SampleSearch and Gibbs sampling (if it converges). For collections generated by Gibbs sampling that diverge on Grid BNs, the proportionality constants of the ROC are omitted. It can be concluded that samples generated by *prune sampling* always converge slower to a stationary posterior distribution.

From the above Table it stands out that the ROC proportionality constants of *prune sampling* are always higher than the proportionality constants of backward sampling, likelihood weighting, SampleSearch and Gibbs sampling (if it converges). Hence, *prune sampling* convergences always slower to a stationary posterior distribution. Instead, on the real-world class BNs without available evidence, there is no single sampling methods that achieves a consist higher ROC over other sampling methods. For the real-world class BNs with 25% available evidence, Gibbs sampling achieves consistently the highest ROC proportionality constant. For all Grid BNs, backward sampling and SampleSearch perform best in terms of a low ROC, i.e. fast convergence.

As illustrated in Figure 5.3(b), the (MCMC) sampling methods for BNs used in this thesis belong to the convergence class $\mathcal{O}(t^{-1/2})$. Therefore, all collections generated by

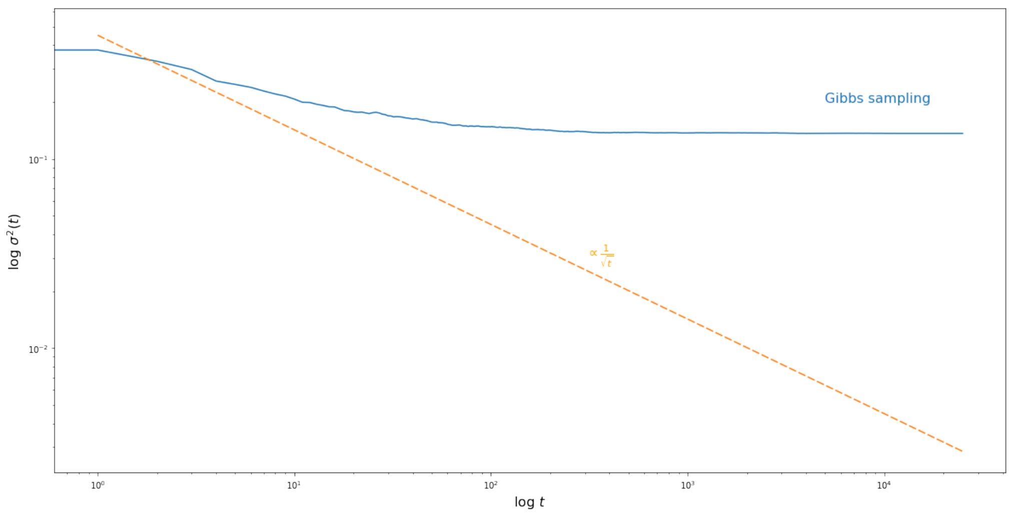


Figure 6.8: Due to being trapped in a certain subset of the state space, Gibbs sampling converges to different posterior distributions as displayed in Figure 6.6. Therefore, the double log plot of the number of samples (t) versus the standard deviation in the collection of samples ($\sigma^2(t)$) can not be characterised as being part of the $\mathcal{O}(t^{-1/2})$ convergence class.

the sampling methods on all 12 BNs converge along the orange dashed line in Figure 5.3(b), except Gibbs sampling on the Grid BNs. In line with the divergence as showed in Figure 6.6, an example of a double log plot of the number of samples (t) versus the standard deviation in the collection of samples ($\sigma^2(t)$) for Gibbs sampling on the Grid 5×5 BN with 25% deterministic ratio is shown in Figure 6.8.

In the next section, the sampling methods are reviewed in terms of time consumption.

6.5 Time consumption

In this section, the time consumption of various sampling methods are characterised. As discussed in section 5.4, the ROC results from Table 6.2 can be used to determine for distinct sampling methods how many samples t are needed to reach $\sigma^2(t) < 0.01$ within a collection of $N = 100$ simulations (see Equation 5.3-5.4). So, conducted experiments measure what time a sampling method needs to generate this specific amount of samples. This procedure provides us with a generic way to compare time consumption amongst various sampling methods.

First of all, it should be noted that Gibbs sampling is only taken into account if the posterior distribution generated by the sampling method converges, i.e. a common pitfall of Gibbs sampling is to diverge on deterministic networks like Grid BNs. Besides, once the smallest $t \in (0, 25.000)$ is known for a sampling method of interest such that $\sigma^2(t) < 0.01$, the denoted amount of time consumed (in seconds) is an average of 10 times repeated procedures to generate t samples. The results are presented in Table 6.3.

For all small BNs – Asia with 0% and 25% available evidence and Grid 3×3 with 25% and 50% determinism – *prune sampling* is the fastest sampling method. This is remarkable since from Table 6.2 it is known that *prune sampling*, on all types of BNs, has the lowest ROC. Therefore, *prune sampling* needs more samples t to reach $\sigma^2(t) < 0.01$ on a collection of $N = 100$ simulations than all other sampling methods. Hence, one can deduce that *prune sampling* is an extremely fast sampling methods on small BNs, i.e. when $|S_{C_{\mathbf{x}}^{\text{np}}}|$ stays small. As such, *prune sampling* becomes a significantly slower sampling method on medium and large sized BNs, i.e. as $|S_{C_{\mathbf{x}}^{\text{np}}}|$ increases.

	Sampling method				
Bayesian network	Gibbs	Prune	Backward	Likelihood	SampleSearch
Asia_ev0	1.37	**0.53**	1.20	1.56	1.68
Alarm_ev0	1.83	3.56	1.81	1.69	**1.32**
Win95pts_ev0	2.62	49.85	2.09	**1.52**	975.89
Asia_ev25	1.43	**0.41**	1.73	1.43	1.05
Alarm_ev25	1.25	2.83	1.43	1.25	**1.22**
Win95pts_ev25	2.02	40.03	**1.73**	1.79	717.94
Grid 3x3_det25	x	**0.75**	1.37	1.36	1.44
Grid 5x5_det25	x	2.68	1.50	**1.19**	1.56
Grid 8x8_det25	x	543.73	1.63	**1.23**	1.63
Grid 3x3_det50	x	**0.73**	1.12	1.12	1.11
Grid 5x5_det50	x	2.42	1.48	**1.44**	1.60
Grid 8x8_det50	x	105.17	1.43	**1.31**	1.68

Table 6.3: Time consumption (in seconds) of sampling methods to reach $\sigma^2(t) = 0.01$ with a collection of $N = 100$ simulations. *Prune sampling* is the fastest sampling method on all small sized BNs. On all medium sized BNs and on large Grid BNs, *prune sampling* is the slowest sampling method. On large real-world BN, only SampleSearch is a slower sampling methods than *prune sampling*. In general, likelihood weighting tends to outperform other sampling methods on medium and large Grid BNs.

On the medium sized BNs – Alarm with 0% and 25% available evidence and Grid 5×5 with 25% and 50% determinism – *prune sampling* is the slowest sampling method. On the large sized real-world BNs – Win95pts with 0% and 25% available evidence – *prune sampling* is a rather slow sampling method but not as exceptional as the very slow appearance of SampleSearch. On the large sized Grid 8×8 BNs – both with 25% and 50% determinism – *prune sampling* is by far the slowest sampling method. The discrepancy between fast and slow performance of SampleSearch on the large real-world BNs and the Grid BNs can be explained by the fact that SampleSearch is devised specifically to deal with deterministic BNs and not necessarily for real-world BNs. In addition, from Table 6.3 it can be observed that likelihood weighting tends to outperform other sampling methods in terms of time consumption on medium and large Grid BNs. On the medium and large size real-world BNs there is no sampling methods that stands out.

As mentioned before in section 4.3.2, exhaustive enumeration during the uniform sampling step on pruned large BNs can result in excessive time consumption of *prune sampling.* This is a results of the exponential blow up of $|S_{\mathcal{C}_{\mathbf{x}}^{\mathrm{np}}}|$, which is discussed in greater detail in the next chapter.

Chapter 7 Improving prune sampling

The results regarding time consumption presented in section 6.5 reveal that *prune sampling* is a fast method on small BNs, but a slow method on medium and large sized BNs. In this chapter, an idea is presented to reduce the excessive time consumption of the pruning technique on large BNs. First, the cause of the lengthy computation process is elaborated on. Thereafter, an adjusted version of *prune sampling* that is less computational intense is implemented and tested.

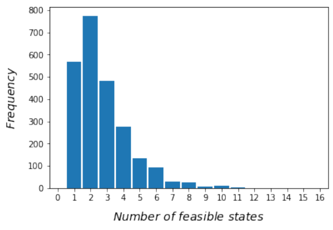
7.1 Exhaustive listing of all feasible states in a pruned network

The main reason of *prune sampling's* excessive time consumption on large BNs is the exhaustive listing of all feasible states in a pruned network. As discussed in section 4.3.2, this exhaustive listing – i.e enumerating the set $S_{\mathcal{C}_{\mathbf{x}}^{\mathrm{np}}}$ explicitly – can become computational intense for large BNs. This presumption is confirmed by the experimental results in chapter 6. In this section, more light is shed on the distribution of the set size of $S_{\mathcal{C}_{\mathbf{x}}^{\mathrm{np}}}$.

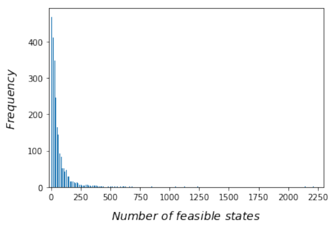
Prune sampling is a time-intensive sampling method due to extreme outliers of $|S_{\mathcal{C}_{\mathbf{x}}^{\mathrm{np}}}|$. Figure 7.1 displays a histogram of $|S_{\mathcal{C}_{\mathbf{x}}^{\mathrm{np}}}|$ based on 2.400 iterations of *prune sampling* on all real-world benchmark BNs. Those 2.400 iterations are randomly chosen from 25.000 samples generated by *prune sampling* on the Asia, Alarm and Win95pts BNs both with and without available evidence.

The key insight from Figure 7.1 is that although the vast majority of set sizes is located at the very left of the histograms, the maximum value of the set sizes is an extreme outlier on the right. The size of the largest set tends to increase excessively as the BN gets larger. For example, on the small sized Asia BN the average of $|S_{\mathcal{C}_{\mathbf{x}}^{\mathrm{np}}}|$ is 2.3, the median set size is 2 whilst the maximum set size is 16. On the medium sized Alarm BN, the average is 63.4, the median is 34 and the maximum set size is 2.218 (35 times the average set size). On the large sized BN Win95pts, the average is 426.6, the median is 50 and the maximum set size is 34.125 (83 times the average set size). Sets with a high number of feasible states need more time to be constructed than sets with a low number of feasible states. Hence, incorporating those those extreme large sets in the listing process causes the lengthy computation time of *prune sampling* on large BNs.

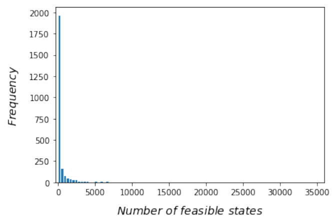
When evidence is available for a BN, more CPT-entries are fixed. Hence, the number of feasible states in a pruned network of this type of BNs tend to be less than for BNs without available evidence. This becomes evident in the histograms of real-world BNs with 25% available evidence, which are displayed in Figure 7.1(d)-(f). On this class



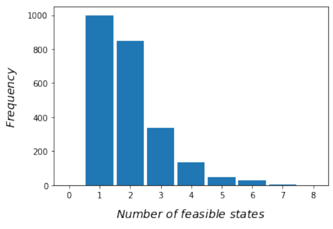
(a) Asia 0% evidence
max. set size 16



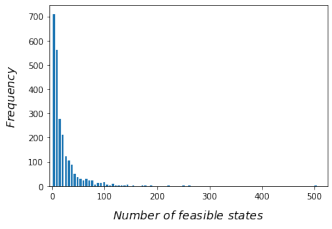
(b) Alarm 0% evidence
max. set size 2.218



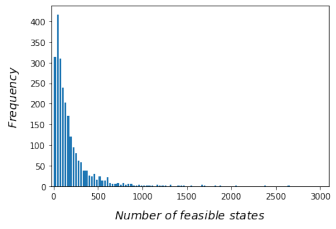
(c) Win95pts 0% evidence
max. set size 34.125



(d) Asia 25% evidence
max. set size 8



(e) Alarm 25% evidence
max. set size 505

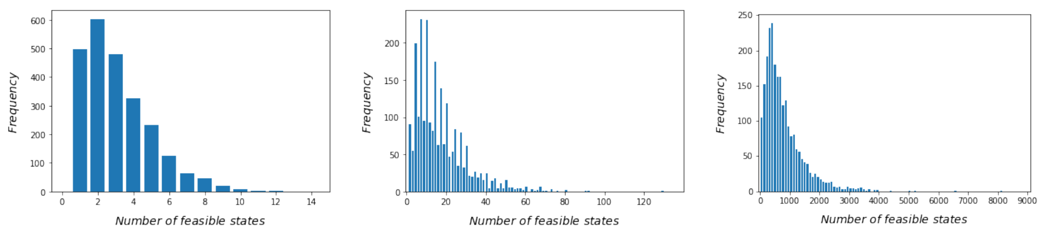


(f) Win95pts 25% evidence
max. set size 2.662

Figure 7.1: Histograms of the number of feasible states in a pruned network based on 2.400 out of 25.000 randomly chosen iterations of *prune sampling* on all real-world BNs with both 0% and 25% evidence available. As the size of the BNs (nodes and parameters) increases, the maximum value of $|S_{\mathcal{C}_{\mathbf{x}}^{\text{np}}}|$ increases excessively. The computation time for the set of 34.125 feasible states on the Win95pts BN with 0% available evidence takes 83 times longer than average and therefore captures a large part of total time consumption. As the amount of available evidence increases, more CPT-entries are fixed during the generation of feasible states. Hence, the number of feasible states in a pruned network drops and the maximum value of $|S_{\mathcal{C}_{\mathbf{x}}^{\text{np}}}|$ decreases. Though, the largest set still contributes excessively to the time consumption of *prune sampling.*

of BNs the maximum set size of $S_{\mathcal{C}_{\mathbf{x}}^{\text{np}}}$ is lower. On the Asia BN with 25% evidence available, the average set size drops from 2.71 to 1.95, the median stays 2 and the maximum set size drops from 16 to 8. On the Alarm BN with 25% evidence available, the average drops from 63.4 to 22.01, the median drops from 34 to 12 and the maximum set size drops from 2.218 to 505. On the Win95pts BN with 25% evidence available, the average drops from 409.43 to 176.76, the median raises from 50 to 108 and the maximum set size drops from 34.125 to 2.662. Since $|S_{\mathcal{C}_{\mathbf{x}}^{\text{np}}}|$, for BNs with 25% available evidence, is in (almost) all aspects lower than the set sizes with 0% available evidence, the computation time of *prune sampling* on the former class of BNs is lower than the latter class. This is confirmed by the results about time consumption presented in Table 6.3.

For the Grid 3x3, 5x5 and 8x8 BNs with 50% determinism the same histograms of set size $|S_{\mathcal{C}_{\mathbf{x}}^{\text{np}}}|$ are displayed in Figure 7.2. For the three BNs respectively, one finds an average set size of 3, 17 and 794, a median set size of 3, 14 and 609.5 and a maximum set sizes of 14, 130 and 8.157. This distribution is less scattered than real-world BNs. So, more medium sized sets $S_{\mathcal{C}_{\mathbf{x}}^{\text{np}}}$ exist. Those sets are more time-intensive to generate



(a) Grid 3x3 50% det. max. set size 14

(b) Grid 5x5 50% det. max. set size 130

(c) Grid 8x8 50% det. max. set size 8.157

Figure 7.2: Histogram of the number of feasible states in a pruned network based on 2.400 out of 25.000 randomly chosen iterations of *prune sampling* applied on Grid 3×3, 5×5 and 8×8 BNs with 50% determinism. For this class of BNs, the distribution of $|S_{\mathcal{C}^{\text{np}}_{\mathbf{x}}}|$ is less left-centered and therefore the average set size tends to be higher than on the real-world BNs. As a result *prune sampling* needs more time to generate samples for Grid BNs.

and are the main cause for excessive overall time consumption. This is reflected in the time-consumption of *prune sampling* on Grid BNs, as displayed in Table 6.3.

Exhaustive listing of all feasible states of a pruned network is essential to choose the next state of a BN completely uniform. This, in its turn, is essential for *prune sampling* to guarantee convergence. However, as illustrated in this section, the explicit enumeration of all feasible state of a pruned network during every iteration of *prune sampling* becomes rather time intensive as BNs become larger. In order to turn *prune sampling* into a faster sampling method on large BNs, one could consider to curb $|S_{\mathcal{C}^{\text{np}}_{\mathbf{x}}}|$ up to an arbitrary set size to save computation time. This approach can be interpreted as a trade-off between accuracy and computational. This idea and its implementation is discussed in the next section.

7.2 Hybrid forward sampling

In this section, an approach is discussed, implemented and tested to mitigate excessive time consumption of *prune sampling* on large BNs. As illustrated in the previous section, the extreme computation time of *prune sampling* is caused by the exhaustive listing of all feasible states in a pruned network. An adjusted version of *prune sampling* that makes use of Hybrid Forward Sampling (HFS) is introduced to limit the amount of listed feasible solution to a pre-determined amount.

The core idea to limit the computation time of *prune sampling* for large BNs, is to limit the size of set $S_{\mathcal{C}^{\text{np}}_{\mathbf{x}}}$ to an arbitrary pre-determined fixed amount $|V|$. The size of $V \subset S_{\mathcal{C}^{\text{np}}_{\mathbf{x}}}$ can be regarded as a trade-off between complete uniformity (see subsection 4.3.2) and computational effort. Besides, the existing *prune sampling* implementation – used to create the results in chapter 6 – makes use of regular forward sampling (see definition 4.8) to find feasible states in a pruned network. As discussed in section 4.3.1, regular forward sampling could generate biased samples. To avoid this bias, adjusted

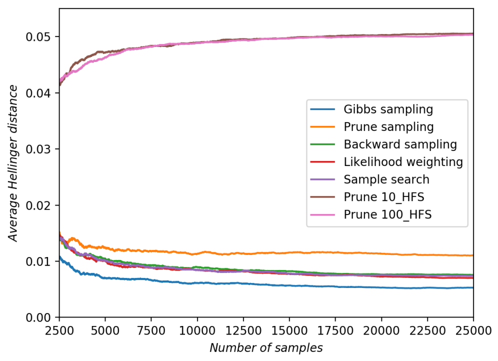
HFS *prune sampling* is used to generate candidate states of the BN. Subsequently, from this reduced set V a state of the BN is randomly chosen to be the next sample.

Taking into account the results regarding time consumption results from Table 6.3 and the set size statistics from the previous section, there is no need to reduce the fast computation time of *prune sampling* on small BNs. For medium sized BNs, time consumption is neither excessive and experiments on this type of BNs are only conducted in order to signify the trade-off between accuracy (complete uniformity) and computational effort. On this class of BNs, experiments are conducted with the adjusted HFS *prune sampling* method with $|V| = 10, 100$. The adjusted *prune sampling* method is expected to provide most added value on large BNs. For this type of BNs, experiments are conducted with the adjusted HFS *prune sampling* method with $|V| = 100, 1.000$.

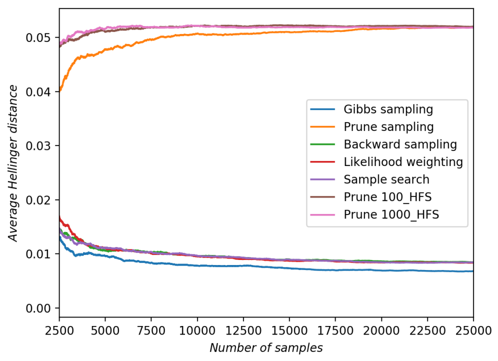
The selection for the the maximum set sizes of V are based on the distributions of the set size $|S_{C_{\mathbf{x}}^{\text{np}}}|$ as given in section 7.1. Our main interest is to what extent this adjusted version of *prune sampling* reduces its time consumption on large BNs and the effects on its accuracy. Since is this adjusted version HFS is used, it can be expected that more near-uniform samples are amongst the feasible states and hence the set tends to be less biased. So, increasing accuracy is expected due to HFS. However, curbing the size of $|S_{C_{\mathbf{x}}^{\text{np}}}|$ to $|V|$ deteriorates the uniformity principle of *prune sampling* . Hence, $|V|$ is expected to be the main parameter to tune accuracy of the adjusted HFS *prune sampling* method, i.e. accuracy is expected to increase as $|V|$ increases.

The results of the adjusted HFS *prune sampling* technique, in terms of accuracy, are presented in Figure 7.3. It turns out that adjusted HFS *prune sampling* on all type of medium and large BNs is inferior to Gibbs sampling and regular *prune sampling*. This result emphasizes the importance of exhaustive listing of *all* feasible states in a pruned network. From the accuracy plots in Figure 7.2, it becomes clear that limiting the set size of $S_{C_{\mathbf{x}}^{\text{np}}}$ effects the accuracy of the pruning technique drastically.

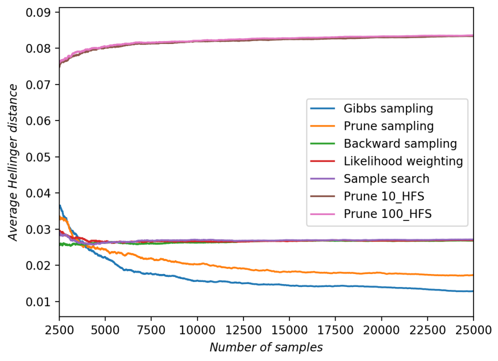
Some additional insights can be gained from these results. A rather remarkable observation is that for all BNs displayed in Figure 7.3, except the Win95pts BN with 25% available evidence, adjusted HFS *prune sampling* with $|V| = 10$ is just as accurate as adjusted HFS *prune sampling* with $|V| = 100$ or with $|V| = 1.000$. On the Win95pts BN with 25% evidence available, adjusted HFS *prune sampling* with $|V| = 100$ even performs better than adjusted HFS *prune sampling* using $|V| = 1000$. This is in contrast to our expectations, since it is expected that adjusted HFS *prune sampling* will be more accurate as $|V|$ increases. This causality is not present in the results and hence the presumed trade-off between computational effort and uniformity should be reconsidered.



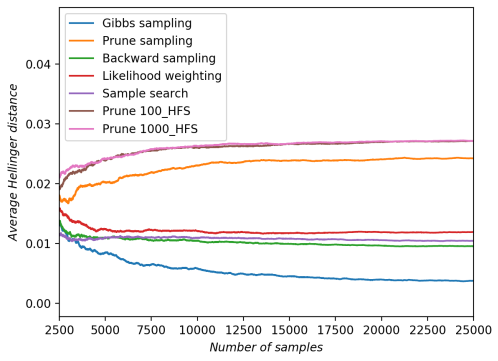
(a) Alarm 0% evidence



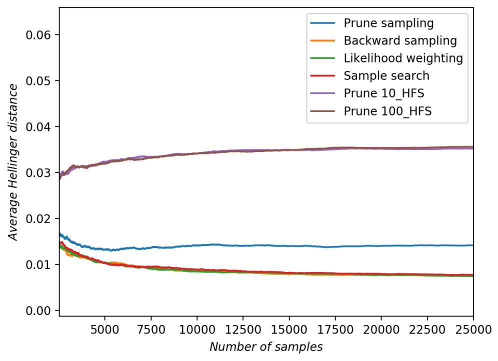
(b) Win95pts 0% evidence



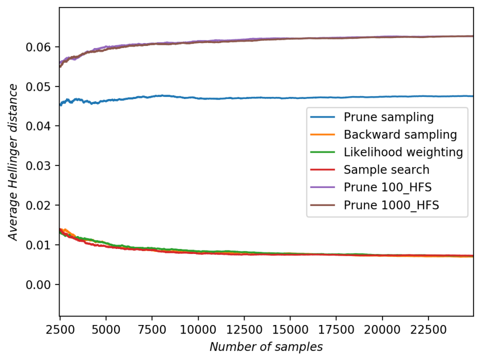
(c) Alarm 25% evidence



(d) Win95pts 25% evidence



(e) Grid 5x5 50% determinism



(f) Grid 8x8 50% determinism

Figure 7.3: Consistent underperformance of adjusted HFS *prune sampling*. This adjusted version of *prune sampling* curbs $|S_{\mathcal{C}^{\text{np}}_{\mathbf{x}}}|$ to an arbitrary size $|V|$. This prevents exhaustive listing of large $S_{\mathcal{C}^{\text{np}}_{\mathbf{x}}}$ and therefore saves substantial computation time. However, limiting this set's size comes with a price: next states of the BNs are chosen less uniform randomly. Adjusted HFS *prune sampling* can therefore be seen as a trade-off between uniformity (accuracy) and computational effort.

Bayesian network	Sampling method			
	Prune	Prune hybrid_10	Prune hybrid_100	Prune hybrid_1000
Alarm_ev0	3.56	2.19	3.12	x
Win95pts_ev0	49.85	10.97	27.36	44.19
Alarm_ev25	2.83	1.93	2.59	x
Win95pts_ev25	40.03	9.75	22.56	34.45
Grid 5x5_det50	2.42	1.38	2.06	x
Grid 8x8_det50	105.17	19.96	51.44	87.62

Table 7.1: Time consumption of adjusted HFS *prune sampling* to reach $\sigma^2(t) = 0.01$. As a consequence of limiting the set size of V to 10, 100 or 1.000, adjusted HFS *prune sampling* is a less time-intensive method than regular *prune sampling* on medium and large sized BNs. Though, this advantage is outweighed by a significant disadvantage: due to the capped set size of $|V|$, samples are chosen significantly less uniform randomly. Hence, adjusted HFS *prune sampling* is less accurate than regular *prune sampling.*

The results of the adjusted HFS *prune sampling* technique in terms of time consumption are presented in Table 7.1. Again, as in section 6.5, it is measured what time it takes to generate $t \in (0, 25.000)$ samples such that $\sigma^2(t) < 0.01$. For all versions of adjusted HFS *prune sampling* with either $|V| =$ 10, 100 or 1.000, the adjusted pruning technique is faster than regular *prune sampling*. The most time is gained when V is small. As $|V|$ increases, the time consumption of adjusted HFS *prune sampling* increases too.

It should be noted that on large BNs the time gain for adjusted HFS *prune sampling* with $|V| = 1.000$ is rather low. Compared to regular *prune sampling* on the Win95pts BNs, reducing time by approximately 10% is not that significant since even adjusted HFS *prune sampling* with $|V| = 10$ is not close to the faster other sampling methods (see Table 6.3). A rational for this could be that the initialization step of the pruning technique is quite lengthy. So, adjusted HFS *prune sampling* saves time by dismissing the largest sets $S_{\mathcal{C}_{\mathbf{x}}^{\text{np}}}$. Though, it still needs a significant amount of time by finding an initialization of the BN.

To conclude, the implemented HFS *prune sampling* method showed unexpected and poor performance, both in terms of accuracy and time consumption. It is highly peculiar that the accuracy of adjusted HFS *prune sampling* is (almost) invariant for the set size $|V|$. Besides, the time gain of adjusted HFS *prune sampling* turned out to be insufficient to compete with the fastest sampling methods like backward sampling, likelihood weighting and SampleSearch. So, the gains in terms of time consumption are outweighted significantly by the loss in accuracy. Hence, adjusting *prune sampling* by making use of HFS to limit $|S_{\mathcal{C}_{\mathbf{x}}^{\text{np}}}|$ turns out to be unfruitful.

Chapter 8 **Conclusion**

In this thesis, the performance of the recently introduced *prune sampling* algorithm [1] is characterised and is compared to conventional sampling methods. This is done by an experimental analysis. Five different sampling methods are evaluated in terms of three performance indicators (accuracy, rate of convergence and time consumption) on 12 distinct Bayesian networks. In total 60 experiments are conducted with regular *prune sampling* and 12 experiments are ran with an adjusted version of *prune sampling*. For each performance indicator conclusions are given regarding the performance of *prune sampling*.

8.1 Accuracy

Prune sampling turns out to be a broadly applicable but inaccurate approximate (MCMC) sampling method. On all types of BNs, both deterministic and real-world BNs, *prune sampling* always converges to the desired posterior distribution. This is noteworthy since convergence to the correct posterior distribution is not trivial. For example, on deterministic BNs, the widely used Gibbs sampling methods fails to converge to the correct posterior distribution and converges to different incorrect posterior distributions. *Prune sampling* is devised in order to guarantee convergence to the correct posterior distribution, but its convergence is clearly not as accurate as other sampling methods if they do converge correctly. So, in terms of accuracy *prune sampling* outperforms Gibbs sampling on so-called block-shaped and deterministic BNs, but it is not competitive compared to other (MCMC) sampling methods like backward sampling, likelihood weighting and SampleSearch.

A reason for the weak accuracy of *prune sampling* can be the deviation amongst various approximations generated by the sampling method. Figure A.1 in Appendix A displays $N = 100$ simulations of $T = 25.000$ samples generated by *prune sampling* on the Grid 8x8 BN with 50% determinism. Note that in comparison to $N = 100$ simulations of $T = 25.000$ samples generated by SampleSearch (as displayed in Figure 5.2) the collection of *prune sampling* has larger deviations amongst its individual samples. A reason for this deviation could be that $T = 25.000$ samples is rather a small amount for the pruning method. It could be true that as T is chosen to be larger than 25.000, the samples generated by *prune sampling* get more close to each other as a function of T and hence that the AHD with respect to the one-marginal variable of interest gets smaller. Some additional and conclusive notes about *prune sampling's* rate of convergence are given the next section.

8.2 Rate of convergence

Prune sampling converges relatively slow to the desired posterior distribution. As a result of Table 6.2, it becomes evident that on eleven out of twelve BNs *prune sampling* converges as slowest to the desired posterior distribution (setting aside the failure of Gibbs sampling on deterministic BNs). This observation is in line with the hypothesis from the previous section. Since *prune sampling* tends to have the slowest rate of convergence, it might hold that $T = 25.000$ is too small for the pruning technique to generate Markov chains that are able to compete with the accuracy of the other established (MCMC) sampling methods. In an early phase of this research, experiments were conducted with $T = 10.000$. It quickly became clear, due to the large deviation amongst the simulations, that $T = 10.000$ was too small to obtain the performance of *prune sampling* in the limit of infinite simulation time, extrapolated from those short simulations. For that reason, further experiments were conducted with $T = 25.000$ but there seems to be a strong rationale to evaluate the accuracy of *prune sampling* for $T > 25.000$.

8.3 Time consumption

In terms of time consumption, *prune sampling* is the fastest method on all small BNs. However, as the size of the BNs grows, *prune sampling* becomes a slower (MCMC) sampling method. As can be seen in Table 6.3, *prune sampling* is the slowest method on all medium BNs and except SampleSearch the slowest method on all large BNs. The cause of this mixed performance can be found in the fact that all feasible states in a pruned network are computed (see section 7.1). A solution to mitigate this drawback of *prune sampling* is proposed, tested and reviewed in section 7.2. This adjusted version of *prune sampling* limits the enumeration of all feasible states in a pruned network in order to save computation time. However, this comes with a price: curbing the set size of candidate feasible states does not guarantee complete uniformly sampling from the reduced sample space. Therefore, this adjusted version of *prune sampling* is (even) less accurate than regular *prune sampling*. In this respect, this adjusted version of *prune sampling* can be seen as a trade-off between accuracy and time consumption. But as discussed in section 8.1, regular *prune sampling* can not afford a deterioration of its performance in terms of accuracy.

To conclude, being devised specifically to deal with determinism, *prune sampling* thwarts its expectations. Particularly with its performance on deterministic Grid BNs. *Prune sampling* is consistently fast on all types of small BNs. However, on this type of BNs its accuracy is again significantly inadequate. On all other BNs, *prune sampling* shows serious shortcomings in terms of all three performance indicators. Hence, overall it needs to be concluded that *prune sampling* is not a competitive sampling method in comparison to established (MCMC) sampling methods.

Appendix A

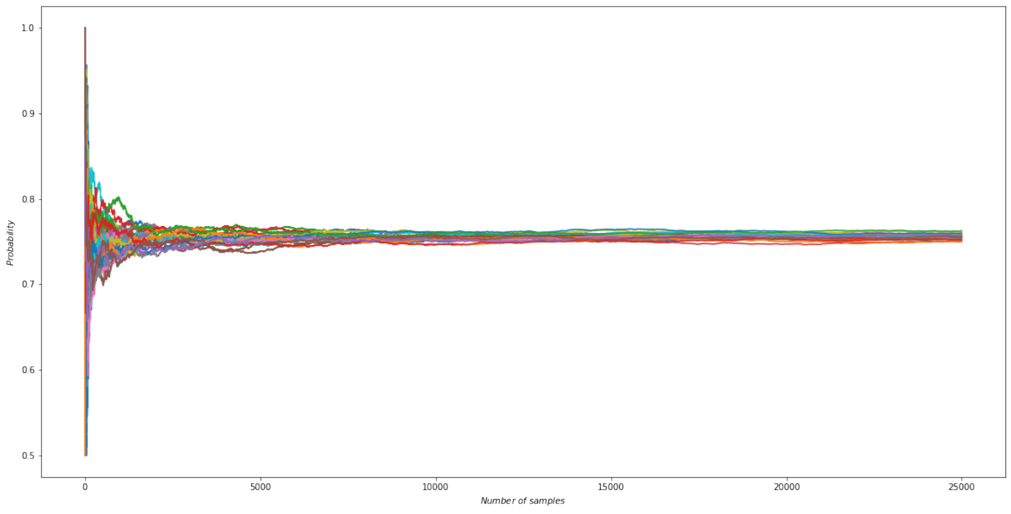


Figure A.1: $N = 100$ simulations of $T = 25.000$ samples generated by *prune sampling* on the Grid 8x8 BN with 50% determinism. Note that in comparison to $N = 100$ simulations of $T = 25.000$ samples generated by SampleSearch (as displayed in Figure 5.2) the collection of *prune sampling* has larger deviations amongst its individual samples. Hence, the AHD (measure for accuracy) of the one-marginal variable of interest tends to be larger for *prune sampling* than for SampleSearch.

List of acronyms and notation index

Acronyms

AHD	Average Hellinger Distance
BN	Bayesian network
CPT	Conditional Probability Table
DAG	Directed Acyclic Graph
HFS	Hybrid Forward Sampling
IID	Independent and Identically Distributed
MCMC	Markov chain Monte Carlo
MLN	Markov Logic Network
ROC	Rate of Convergence
SAT	Satisfiability Problem

Notation

C	a collection of (arbitrary) CPT-labels
H	number of steps needed to make a pruned collection $\mathcal{C}^{\mathrm{p}}_{\{\mathbf{x},\mathbf{y}\},h}$ and a non-pruned collection $\mathcal{C}^{\mathrm{np}}_{\{\mathbf{x},\mathbf{y}\},h}$, such that $\mathbf{x}$ can make a transition to $\mathbf{y}$
N	total number of MCMC simulations
P	probability distribution over the BN, induced by its CPTs
S_C	all feasible states of the BN which could be created from C
T	total number of samples
V	set of predetermined fixed size, contains feasible states generated by HFS
X_i	variable in a BN
Y	random variable to be estimated by MCMC simulations
$\mathcal{C}^{\mathrm{np}}_{\mathbf{x}}$	set with non-pruned CPT-labels, the BN is pruned around the state $\mathbf{x}$
$\mathcal{C}^{\mathrm{p}}_{\mathbf{x}}$	set with pruned CPT-labels, the BN is pruned around the state $\mathbf{x}$
$\mathcal{C}_{\mathbf{x}}$	CPT-labels corresponding to state $\mathbf{x}$ of the BN
$\mathcal{C}$	set with labels of all CPT-entries of all variables in the BN
Ω	state space, all possible (not necessarily feasible) states of a BN
$\mathcal{U}(\cdot)$	uniform distribution over a set
$\mathbf{E}$	set of evidence
$\mathbf{x}$	set of assigned values to random variables
$\mathbf{y}_i$	individual sample in the collection of samples, i.e. $\mathbf{y}_i \in \mathcal{Y}$

δ	parameter in the polynomial expansion of $g(t) = \sigma^2(t)\sqrt{t}$, used to tune the convex- and concaveness of $g(t)$
γ	acceptance probability in Metropolis sampling
$\hat{\mu}$	estimation of μ by N MCMC simulations
$\perp\!\!\!\perp$	symbol for independence
$\mathbf{X}$	set of random variables
$\mathbf{x}'$	candidate sample
$\mathbf{x}^{(t)}$	sample of a BN at step t
$\mathcal{A}$	acceptance probability of proposed transition in MCMC simulation
$\mathcal{C}^{\text{np}}_{\{\mathbf{x},\mathbf{y}\},h}$	h-th step in creating a set with non-pruned CPT-labels $\mathcal{C}^{\text{np}}_{\{\mathbf{x},\mathbf{y}\}}$, such that $\mathbf{x}$ can make a transition to $\mathbf{y}$
$\mathcal{C}^{\text{p}}_{\{\mathbf{x},\mathbf{y}\},h}$	h-th step in creating a set with pruned CPT-labels $\mathcal{C}^{\text{p}}_{\{\mathbf{x},\mathbf{y}\}}$, such that $\mathbf{x}$ can make a transition to $\mathbf{y}$
$\mathcal{C}^{\text{np}}_{\{\mathbf{x},\mathbf{y}\}}$	set with non-pruned CPT-labels, the BN is pruned around the states $\mathbf{x}$ and $\mathbf{y}$
$\mathcal{C}^{\text{p}}_{\{\mathbf{x},\mathbf{y}\}}$	set with pruned CPT-labels, the BN is pruned around the states $\mathbf{x}$ and $\mathbf{y}$
$\mathcal{G}$	Bayesian network structure, a directed acyclic graph
$\mathcal{I}_l(\mathcal{G})$	local independencies of $\mathcal{G}$
$\mathcal{N}$	topological ordering of nodes in the BN
$\mathcal{S}$	set with all samples generated by a sampling method
$\mathcal{T}^Q$	proposal distribution in MCMC simulation
$\mathcal{T}$	transition model
$\mathcal{X}$	set of all variables in the BN
$\mathcal{Y}$	collection of samples generated by a sampling method
$\star$	non-zero CPT-entry
ND_{X_i}	non-descendants of variable X_i
Pa_{X_i}	direct parents of variable X_i
$\text{Val}(X_i)$	values that a random variable X_i can take
$\widetilde{P}$	estimation of probability distribution P
$c_{k(l_i)}$	CPT-entry corresponding to label $k(l_i)$
f	linear function fitted to the simplified polynomial expansion g plotted in terms $t^{-\delta}$
$g(t)$	simplified polynomial expansion of $\sigma^2(t)\sqrt{t}$
h	indexation of the number of steps needed to make a pruned collection such that $\mathbf{x}$ can make a transition to $\mathbf{y}$, $1 \leq h \leq H$
i	indexation of variables in a BN, $1 \leq i \leq n$
$k(l_i)$	label of all entries in the CPT corresponding to variable X_i
k	used to denote the abbreviation of a variable name in a BN
l_i	indexation of the entries in the CPT of variable X_i, $1 \leq l \leq Val(X_i)$
m	indexation of $Val(X_i)$, i.e. the number of column entries in the CPT corresponding to variable X_i

n	total number of variables in a BN
t	indexation of samples, $1 \leq t \leq T$
$w_{\mathbf{x}}$	assigned weight to sample $\mathbf{x}$ according to likelihood weighting
x_i	assigned value to random variable X_i
α'	candidate proportionality constant during the procedure to determine the ROC α
α	quantifies the proportionality constant of the MCMC sampling technique that belongs to the convergence class $\mathcal{O}(t^{-1/2})$, used as indicator of the ROC
β_i	coefficients in the polynomial expansion of the standard deviation $\sigma^2(t)$
μ	expected value of Y
$\sigma^2(t)$	standard deviation of a (MCMC) simulations at sample t

References

[1] Frank Phillipson, Jurriaan Parie, and Ron Weikamp. Prune Sampling: a MCMC inference technique for discrete and deterministic Bayesian networks. *arXiv preprint arXiv:1908.06335*, 2019.

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

[3] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Elsevier, 2014.

[4] F Phillipson, ICL Bastings, and N Vink. Modelling the effects of a CBRN defence system using a Bayesian belief model. 2015.

[5] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques.* MIT press, 2009.

[6] Hoifung Poon and Pedro Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI*, volume 6, pages 458–463, 2006.

[7] Vibhav Gogate and Rina Dechter. SampleSearch: importance sampling in presence of determinism. *Artificial Intelligence*, 175(2):694–729, 2011.

[8] Nasser M Nasrabadi. Pattern recognition and machine learning. *Journal of electronic imaging*, 16(4):049901, 2007.

[9] Thomas Dyhre Nielsen and Finn Verner Jensen. *Bayesian networks and decision graphs.* Springer Science & Business Media, 2009.

[10] Gregory F Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial intelligence*, 42(2-3):393–405, 1990.

[11] Judea Pearl and H Gener. An improved constraint propagation algorithm for diagnosis. In *International Joint Conference on AI, Milano (Italy)*, pages 1105–1111, 1987.

[12] Robert Fung and Kuo-Chu Chang. Weighing and integrating evidence for stochastic simulation in Bayesian networks. In *Machine Intelligence and Pattern Recognition*, volume 10, pages 209–219. Elsevier, 1990.

[13] RD Shachter and M Peat. Simulation approaches to probabilistic inference for general probabilistic inference on belief networks. *Proceeding• of the Fifth Worhhop on Uncertainty in Artificial Intelli gence, Detroit, Michigan*, 1989.

[14] R Martin Chavez and Gregory F Cooper. A randomized approximation algorithm for probabilistic inference on Bayesian belief networks. *Networks*, 20(5):661–685, 1990.

[15] Michael Shwe and Gregory Cooper. An empirical analysis of likelihood-weighting simulation on a large, multiply connected medical belief network. *Computers and Biomedical Research*, 24(5):453–475, 1991.

[16] Robert Fung and Brendan Del Favero. Backward simulation in Bayesian networks. In *Uncertainty Proceedings 1994*, pages 227–234. Elsevier, 1994.

[17] Vibhav Gogate and Rina Dechter. Samplesearch: a scheme that searches for consistent samples. In *Artificial Intelligence and Statistics*, pages 147–154, 2007.

[18] Dominik Jain, Klaus Von Gleissenthall, and Michael Beetz. Bayesian logic networks and the search for samples with backward simulation and abstract constraint learning. In *Annual Conference on Artificial Intelligence*, pages 144–156. Springer, 2011.

[19] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

[20] Deepak Venugopal and Vibhav Gogate. GiSS: Combining Gibbs sampling and SampleSearch for inference in mixed probabilistic and deterministic graphical models. In *AAAI*, 2013.

[21] Julian Besag and Peter J Green. Spatial statistics and Bayesian computation. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 25–37, 1993.

[22] P Damlen, John Wakefield, and Stephen Walker. Gibbs sampling for Bayesian non-conjugate and hierarchical models by using auxiliary variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(2):331–344, 1999.

[23] WR Gilks, S Richardson, and DJ Spiegelhalter. Interdisciplinary statistics. *Markov chain Monte Carlo in practice*, 1996.

[24] Wei Wei, Jordan Erenrich, and Bart Selman. Towards efficient sampling: exploiting random walk strategies. In *AAAI*, volume 4, pages 670–676, 2004.

[25] Art B. Owen. *Monte Carlo theory, methods and examples.* 2013.

[26] D Jain. Probcog toolbox, 2011.

[27] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov chain Monte Carlo.* CRC press, 2011.

[28] Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–224, 1988.

[29] Ingo A Beinlich, Henri Jacques Suermondt, R Martin Chavez, and Gregory F Cooper. The ALARM monitoring system: a case study with two probabilistic inference techniques for belief networks. In *AIME 89*, pages 247–256. Springer, 1989.

[30] Bruce Abramson, John Brown, Ward Edwards, Allan Murphy, RL Winkler, et al. A Bayesian system for forecasting severe weather. *International Journal of Forecasting*, 12(1):57–71, 1996.

[31] Tian Sang, Paul Beame, and Henry Kautz. Solving Bayesian networks by weighted model counting. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, volume 1, pages 475–482. AAAI Press, 2005.