Final thesis

Optimizing Queries in Bayesian Networks

by

Johannes Förstner

LIU-IDA/LITH-EX-A--12/062-SE

2012-12-09
Optimizing Queries in Bayesian Networks
And Optimizing Bayesian Networks for Queries

Master thesis by Johannes Förstner
Linköping University 2012
Supervisor: Jose M. Peña
Examiner: Fang Wei-Kleiner
Abstract

This thesis explores and compares different methods of optimizing queries in Bayesian networks. Bayesian networks are graph-structured models that model probabilistic variables and their influences on each other; a query poses the question of what probabilities certain variables assume, given observed values on certain other variables. Bayesian inference (calculating these probabilities) is known to be NP-hard in general, but good algorithms exist in practice.

Inference optimization traditionally concerns itself with finding and tweaking efficient algorithms, and leaves the choice of algorithms' parameters, as well as the construction of inference-friendly Bayesian network models, as an exercise to the end user. This thesis aims towards a more systematic approach to these topics: We try to optimize the structure of a given Bayesian network for inference, also taking into consideration what is known about the kind of queries that are posed.

First, we implement several automatic model modifications that should help to make a model more suitable for inference. Examples of these are the conversion of definitions of conditional probability distributions from table form to noisy gates, and divorcing parents in the graph. Second, we introduce the concepts of usage profiles and query interfaces on Bayesian networks and try to take advantage of them. Finally, we conduct performance measurements of the different options available in the used library for Bayesian networks, to compare the effects of different options on speedup and stability, and to answer the question of which options and parameters represent the optimal choice to perform fast queries in the end product.

The thesis gives an overview of what issues are important to consider when trying to optimize an application's query performance in Bayesian networks, and when trying to optimize Bayesian networks for queries.

The project uses the SMILE library for Bayesian networks by the University of Pittsburgh, and includes a case study on script-generated Bayesian networks for troubleshooting by Scania AB.
# Table of contents

1 Introduction.................................................................................................................. 5
   1.1 What is a Bayesian network?.................................................................................. 6
      1.1.1 An example.................................................................................................. 6
      1.1.2 Bayes' theorem............................................................................................ 7
      1.1.3 Backgrounds................................................................................................. 7
      1.1.4 Relevance in Bayesian networks.................................................................. 8
      1.1.5 Independent influences................................................................................ 9
   1.2 Troubleshooting as a decision-theoretic problem............................................... 11
   1.3 Queries: definition and notation......................................................................... 13

2 Analysis......................................................................................................................... 14
   2.1 A closer look at the model of the case study...................................................... 15
      2.1.1 Usage Profiles............................................................................................. 16
   2.2 Inference algorithms............................................................................................ 18
   2.3 Software review: Bayesian network tools.......................................................... 20
      2.3.1 SMILE in-depth........................................................................................... 20

3 Approaches to query optimization............................................................................ 23
   3.1 Preprocessing Bayesian networks...................................................................... 25
      3.1.1 Divorcing..................................................................................................... 25
      3.1.2 Conversion to noisy-MAX definitions......................................................... 27
      3.1.3 Removing unnecessary arcs....................................................................... 28
   3.2 Preprocessing specific to the query interface..................................................... 29
      3.2.1 Marginalizing auxiliary nodes.................................................................... 29
   3.3 Library options..................................................................................................... 31
      3.3.1 Targets......................................................................................................... 31
      3.3.2 Relevance reasoning.................................................................................... 31
   3.4 Methodology......................................................................................................... 32
      3.4.1 The testbench............................................................................................... 32
      3.4.2 Test setup for the case study..................................................................... 34

4 Results........................................................................................................................ 36
   4.1 Usage profile: diagnostic direction..................................................................... 37
      4.1.1 First pass: targets and useful algorithms................................................... 37
      4.1.2 Second pass.................................................................................................. 39
      4.1.3 Third pass.................................................................................................... 46
   4.2 Usage profile: causal direction............................................................................ 48
      4.2.1 First pass: filtering useful algorithms........................................................ 48
      4.2.2 Second pass.................................................................................................. 50
      4.2.3 Third pass.................................................................................................... 55

5 Discussion.................................................................................................................... 58
   5.1 General observations........................................................................................... 59
      5.1.1 The stabilizing effect of divorcing on diagnostic queries.......................... 59
      5.1.2 Other preprocessing.................................................................................... 59
      5.1.3 The importance of relevance reasoning..................................................... 60
   5.2 Applicability to other domains........................................................................... 61
   5.3 Issues and future work......................................................................................... 62
1 Introduction

Bayesian networks are graph-structured models of probability distributions. The nodes and arcs in the graph represent probabilistic variables and the dependencies between them. The basic use of Bayesian networks is to compute the probability distributions of certain variables, given what is known about the values of certain other variables. This is called inference, or also belief updating.

There has been a lot of research on efficient inference in Bayesian networks. Different algorithms and forms of relevance reasoning have been developed to optimize the performance of inference computations, often successful for some kinds of models while less optimal for others. The reason for this is that Bayesian inference in general is NP-complete.

Most research on efficient Bayesian inference is conducted with the assumption of a sudden query – a model gets loaded, the evidences are given, and the algorithm is started and has to compute the posterior probability distributions of the other variables. As the algorithm should be general, the input model is taken as given. However, in real-world applications, this assumption does not hold: Bayesian network models are carefully constructed prior to their deployment and use in a software product, which provides a great opportunity for optimization.

A Bayesian network can be created by hand by someone who is an expert in his/her domain but not necessarily in Bayesian inference; it can be automatically extracted from statistical data with some form of machine learning (there is a lot of research about this topic as well), or it can be programmatically generated from some other assets. None of these methods guarantee, however, that the structure of the resulting model will be optimal for inference algorithms to perform well.

This thesis project seeks to bridge the gap between model creation and use, providing a general methodology for analyzing the model at hand and preparing it for the inference engine, as well as preparing the inference engine for the model.
1.1 What is a Bayesian network?

A Bayesian network models probabilistic variables and their dependencies in a directed acyclic graph. Each node represents a variable, and the arcs between them represent their dependencies. Each variable has several possible states with different probabilities. The probability distribution over its states depends on the values of the variables that are its parents in the graph structure. Each variable therefore has a so-called *conditional probability distribution* (CPD) that defines the probabilities of its states for each combination of the states of its parents, usually in the form of a table (CPT, for conditional probability table). Those variables which do not have parents define an unconditional, or prior, probability distribution.

On such a model, it is possible to calculate the prior probability distributions also of those variables which have parents, by summing out the probabilities from the parents to the children, starting at the roots, a process which is also called marginalization. Furthermore, it is possible to add *evidence* into the calculation. Evidence means that some variables are known to be in a specific state. This changes the probabilities of other variables' states. A probability distribution over a variable's states that results from evidence on other variables is called a *posterior probability distribution*. The process of calculating posterior probability distributions, given evidence, is called Bayesian inference, or also belief updating, because the beliefs in variables' outcomes get updated after considering the evidence.

A query, in this thesis report, specifies a set of *evidences*, variables together with their respective states, and a set of *targets*, variables of which the posterior probability distribution should be calculated (a formal definition is provided in chapter 1.3 below).

1.1.1 An example

![Bayesian Network Example](Illustration1.png)

*Illustration 1: Bayesian example network "Weather-Sprinkler-Grass". Source: Wikipedia*

Illustration 1 shows a Bayesian network with the three variables Weather, Sprinkler and Grass; Sprinkler depends on Weather, and Grass depends on both. All three variables are discrete and have two states each. The tables show the conditional probability distributions of the variables' states. For example, the network shows that it rains 20% of the time, and if it does not rain, the sprinkler will be switched on 40% of the time. In formulas, this can be expressed as $P(\text{Weather.rain})=0.2$ and $P(\text{Sprinkler.on} | \text{Weather.sunshine})=0.4$ (speak: the probability of Sprinkler.on given Weather.sunshine is 0.4). It is trivial to calculate that the sprinkler will be switched on during $0.2*0.01+0.8*0.4 = 32.2\%$ of the day ($P(\text{Sprinkler.on})=0.322$). However, if you want to calculate the probability that the water on the lawn comes from the sprinkler, or $P(\text{Sprinkler.on} | \text{Grass.wet})$, you have to make use of Bayes' theorem.
1.1.2 Bayes' theorem

According to Bayes' theorem, the conditional probability distribution \( P(A|B) \) can be calculated from the reverse conditional probability distribution \( P(B|A) \) and the prior distributions \( P(A) \) and \( P(B) \), by this formula:

\[
P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}
\]

Bayes' theorem (1)

Furthermore, since probability distributions have to add up to 1 (which means 100%), the prior distribution \( P(B) \) can be omitted by using a normalization constant instead:

\[
P(A|B) = \alpha P(B|A) \cdot P(A)
\]

Bayes' theorem (2)

So, returning to the sprinkler example, \( P(\text{Sprinkler.on}|\text{Grass.wet}) \) can be calculated as follows: First, we sum out the variable Weather by calculating:

\[
P(\text{Sprinkler}) = P(\text{Sprinkler}|\text{Weather.rain}) \cdot P(\text{Weather.rain}) + P(\text{Sprinkler}|\text{Weather.sunshine}) \cdot P(\text{Weather.sunshine})
\]

\[
P(\text{Grass}|\text{Sprinkler}) = P(\text{Grass}|\text{Sprinkler},\text{Weather.rain}) \cdot P(\text{Weather.rain}) + P(\text{Grass}|\text{Sprinkler},\text{Weather.sunshine}) \cdot P(\text{Weather.sunshine})
\]

We obtain the following distributions:

\[
P(\text{Sprinkler}) = \{\text{on}: 0.322; \text{off}: 0.678\}
\]

\[
P(\text{Grass}|\text{Sprinkler.on}) = \{\text{wet}: 0.901; \text{dry}: 0.099\}
\]

\[
P(\text{Grass}|\text{Sprinkler.off}) = \{\text{wet}: 0.234; \text{dry}: 0.766\}
\]

Then, by Bayes' theorem:

\[
P(\text{Sprinkler}|\text{Grass.wet}) = \alpha \cdot P(\text{Grass.wet}|\text{Sprinkler}) \cdot P(\text{Sprinkler})
\]

\[
= \alpha \cdot \{\text{on}: 0.290; \text{off}: 0.159\}
\]

\[
= \{\text{on}: 0.646; \text{off}: 0.354\}
\]

In conclusion, the water on the lawn comes from the sprinkler with 64.6% probability.

1.1.3 Backgrounds

Typical applications

A typical field of application for Bayesian networks is medicine. Bayesian networks can model symptoms and diseases as probabilistic variables, and can be used to calculate the probabilities of different diseases, given the concrete symptoms observed on a patient.

Bayesian networks are also often used in artificial intelligence, to make decisions in situations of uncertainty by using probabilistic inference. A method of embedding decision making directly into Bayesian networks is by adding nodes that represent decisions and utilities. These models are called Decision Networks or Influence Diagrams.

History & philosophy

The term "Bayesian network" was coined by Judea Pearl in 1985 after Thomas Bayes (c. 1701 – 1761), who was an English mathematician and Presbyterian minister, and who first found the formula now known as Bayes' theorem (see above). The formula was written in An Essay towards solving a Problem in the Doctrine of Chances which was published only after his death, in 1763. The formula was also independently discovered later by the famous French mathematician and astronomer Pierre-Simon Laplace, who published it 1812.

Bayes' theorem caused a lot of disagreement among scientists because it did not correspond to the
frequentist position, which states that a probability distribution has to be measurable by the frequency of occurrence, and that hypotheses be tested without assigning them a prior probability. Bayesian probability states that any state of belief which can be expressed with a probability distribution is a proper probabilistic variable.

The following sections will explain further concepts in Bayesian networks that will be important to query optimization later on.

1.1.4 Relevance in Bayesian networks

A node Y is said to be relevant to another node X if Y is needed to calculate the posterior distribution of X. If Y is not relevant to X, then the posterior of X cannot be affected by a change in Y's CPD, nor by setting evidence on Y or even by deleting Y. Determining relevance is useful because it allows to avoid unnecessary computations when answering queries and thus saving time and RAM space. Y is relevant to X if they are not d-separated, and Y is not barren from X. These conditions depend on the graph structure and on the evidence present in the network.

First, some definitions from graph theory: A path is a sequence of nodes connected by edges. In a directed graph, a directed path connects its nodes with each arc pointing from the previous to the next one; an undirected path, also called trail, can consist of arcs in both directions. The internal nodes of a trail are said to connect head-to-head (with both arcs pointing to the node), tail-to-tail (with both arcs pointing away from the node), or head-to-tail, respectively tail-to-head, if the path passes through the node with one arc incoming and one arc outgoing. Furthermore, if there is a directed path from a node X to a node Y, then X is an ancestor of Y, and Y is a descendant of X; this definition only applies to directed acyclic graphs.

Now, d-separation stands for directed separation and is the opposite of d-connection. Two nodes X and Y are d-connected iff there is a trail between X and Y that passes through nodes in the following ways:

a) head-to-tail (or tail-to-head) through nodes without evidence,
b) tail-to-tail through nodes without evidence, and
c) head-to-head through nodes that either do have evidence or some of the descendants have evidence.

A node is barren if neither the node nor any of its descendants have evidence.

In addition, a node Y is called a nuisance node if it is needed to calculate the posterior of X but does not change X's value. That is, Y does not have evidence and is also not transmitting any evidence information to the target variable; it is only needed for the calculation because it is an ancestor of a non-nuisance variable Z and needs to be summed out. The concept of nuisance variables is not important for the thesis but is included here for the sake of completeness.

In the example network of Illustration 2, consider B to be the target node.

B and C are d-connected but become d-separated when evidence gets set on A. In other words, when there is evidence on A, C is no longer relevant to B.

B and D are d-separated but become d-connected when evidence gets set on E, or on F.

B and F are d-connected but become d-separated when evidence gets set on E.

However, even without evidence on E, F is still irrelevant for B because F is a barren node. So, F only becomes relevant for B if there is evidence on F but not on E.
If A and C have no evidence, then A is a nuisance node for B.

There are also special forms of relevance reasoning that apply to ICI gates, which is explained in the following subchapter.

1.1.5 Independent influences

Often, the parents of a node influence the child independently of each other. An example of this is a symptom (say, a headache) that is increased by several conditions (say, a migraine, or a difficult lecture). Now, it seems sound to say that migraine diminishes the probability of being headache-free regardless of the headache probability given by other causes. Such cases of independence of causal influences (ICI) are quite common in practice when modeling real-world problems.

If the influences of parents on a variable are causally independent, then the CPD of the child can be defined by a linear number of parameters instead of a table with an exponential number of entries. A node with a CPD defined in such way is called an ICI gate. The parametric definition is much smaller than a definition by a full table, especially for nodes with a lot of parents.

There are several types of ICI gates that have different ways of combining the influences of their parents; the noisy-MAX is an especially popular one (and important to this thesis). The noisy-MAX is a generalization of a logical OR gate for use with multivalued probabilistic variables. Illustration 3 provides an example of a noisy-MAX definition. Simplified, it can be said that Headache is caused by Migraine OR DifficultLecture. Noisy-MAX is an amechanistic ICI gate, which means it is a kind of ICI gate that defines a "distinguished" state for the child and its parents, which semantically is the default state of the variable. A noisy-MAX gate regards the states of its parents as causes and the states of the child as effects. The distinguished state of the parent represents "no cause" and the distinguished state of the child represents "no effect".

An amechanistic ICI gate like noisy-MAX allows more fine-grained relevance reasoning: Usually, evidence on the child will d-connect its parents. However, when a noisy-MAX node has evidence set to its distinguished state, the parents stay independent of each other. If there is no effect, then the probabilities of the causes stay independent. This special form of relevance reasoning will be called "noisy-MAX relevance" in this thesis report.

To understand noisy-MAX relevance, consider as a graphic example that you are sitting in a lecture, and you have a headache. Now, your headache can be caused by a beginning migraine as well as a difficult lecture (as modeled by the Bayesian network in Illustration 3). If the lecture is actually difficult, you are relieved because this explains away the migraine, and you will get fine again when the lecture is over. If you, on the other hand, do not have any headache, then your belief in a beginning migraine does not get influenced by the difficulty of the lecture in any way.

The parameters of a noisy-MAX definition are defined as follows: For each state of each parent, a posterior probability distribution over the child's states is defined that specifies the probabilities of the effects given that the cause is present and other causes are absent. To account for "other unmodeled causes", a so-called leak parameter is used. In the example above, the leak parameter signifies the probability of a headache if you neither have migraine nor are sitting in a difficult lecture, for example because of meteorosensitivity, dehydration, or brain cancer. The leak parameter can be regarded as another binary parent which has evidence set to its causal state.
In the example above, \( P(\text{headache} | \text{difficultLecture} \land \neg \text{migraine} \land \neg \text{leak}) = 0.2 \).

Semantically, when several causes are present, their influences on the effect are combined, and the probability of the effect becomes higher. To obtain the probability distribution for a combination of causes, the probability of the distinguished state gets multiplied, e.g.:

\[
P(\neg \text{headache} | \text{migraine} \land \text{difficultLecture}) = 0.99 \cdot 0.8 \cdot 0.5 = 0.396.
\]

A CPD in the simple table form can be obtained from a CPD in noisy-MAX form by calculating each entry in the table with the above formula.

Literature: The material of chapter 1.1 is largely based on the well-known book by Jensen and Nielsen [1], to which we refer the interested reader.
1.2 Troubleshooting as a decision-theoretic problem

This chapter introduces the concept of troubleshooting as a decision-theoretic problem, providing a background for the case study.

Literature: This chapter is based on Scania's previous work on this topic, in the form of two PhD theses by Anna Pernestål [2] and Håkan Warnqvist [3], and a paper written at Linköping University [4].

Definitions

Troubleshooting is the task of fixing a faulty system. This means that there are observable symptoms; that these observations can be used to diagnose the faults present in the system; that actions can be taken to repair faults; and that, at the end of the process, the system is fault-free. A system can be running or at rest. Immediate symptoms are directly observable even when the system is at rest; other symptoms appear only while the system is running. A system also has an assembly state. Some symptoms are only observable in certain assembly states, and some faults can only be repaired in certain assembly states; on the other hand, a system can only be run while it is fully assembled. Beside actions for observing symptoms and repairing components, there are also actions available to the troubleshooter for changing the system's state of assembly. These actions have certain costs associated to them (time, resources, money). When viewing troubleshooting as a decision-theoretic problem, at each point during the process the troubleshooter has to decide what to do next. This is done by planning ahead, and, based on what is known so far, finding the plan that leads to the minimum expected cost of repair.

Case study: troubleshooting Scania trucks

Scania AB is a company based in Sweden that (among other things) develops and produces trucks, and also offers maintenance service for them. Like all modern vehicles, they are getting more and more complex, and frequently the mechanics at repair workshops have difficulties in keeping up with all the latest developments. Scania is therefore interested in examining how computer-aided troubleshooting with expert systems can make truck maintenance faster and/or cheaper.

An application for computer-aided troubleshooting should take as an input the driver's and the mechanic's observations about the truck's faulty behavior (e.g. oil is leaking somewhere, the brakes are not working, a failure is displayed, a display freezes/stays black, etc.), diagnose the truck's faults from these observations, find the plan that leads to the minimum expected cost of repair (see above), and give the first step of that plan as a recommendation of what to do next. Recommended actions can be to make further observations (to exclude some faults), to change the truck's assembly state, to exchange a component (thereby repairing it, in the sense given above), to operate the truck to see whether non-immediate symptoms occur, and to declare the truck fault-free.

The planner and the diagnoser

To this end, a prototypical application had been built in earlier projects to examine the theoretical background and practical applicability of decision-theoretic troubleshooting for trucks. The application consists of two parts: a planner and a diagnoser. The planner is a completely generic automated planning algorithm as used in many areas of artificial intelligence. Its utility function is the estimated cost of repair, which it tries to minimize. Planning is done by heuristic search; an anytime planning algorithm is used in order not to waste the mechanic's time, because this ultimately also adds up to the overall cost of repair.

The diagnoser is used for probabilistic reasoning about the truck. For this, it uses the mathematical framework of Bayesian networks. A Bayesian network model of a certain truck system is used that contains variables for components and their fault probabilities, as well as variables that represent behavior of the system. These can be instantiated with evidence about faulty behavior, and thus the posterior probabilities of faults in the components can be calculated. The planner will use these for its reasoning process.
To accurately represent a system under repair with Bayesian networks, a framework of event-driven non-stationary dynamic Bayesian networks had been developed, that accounts for the changes in dependencies among variables after changes in the system have been made (that is, after a component was replaced or the system was operated). However, these calculations can be done without actually making changes to the Bayesian network at run-time; this needs an incremental algorithm but the planner computes the posteriors for each planned step incrementally anyway. So, from the view of the diagnoser, the model's graph-structure and conditional probability distributions do not get changed at run-time; all it has to do is answer the planner's queries, and the faster it can answer them, the more detailed the anytime-planner can plan in the time it is given, and the better will be the quality of the produced plan.

Building Bayesian networks for troubleshooting

One major concern with Bayesian networks is the question of how to build them. Building a Bayesian network model of a truck system (or anything practically useful, for that matter) just by hand is an incredibly daunting and error-prone task. Therefore, it is imperative to use as many automatic procedures that can help in building the model from existing data as possible.

The basic idea behind building these models of systems in Scania trucks is to use the fault propagation graph of the system. The fault propagation graph shows how faults propagate in the structure of the system's software components, and already exists as a by-product of the system's development process. Furthermore, the building process uses the existing statistics about the prior probabilities of faults in the components, and the description of the Diagnostic Trouble Codes (DTC) that can occur in the system. Diagnostic Trouble Codes are occasions of faulty behavior that get recorded by the truck's on-board computer, and can give more insight on the system's state than the external and subjective observations of faulty behavior done by the driver and the mechanic.

This data is patched together to a Bayesian network with the help of a script, and therefore the additional human effort of building the model is minimal. On the downside, the model resulting from this procedure is in no way guaranteed to be optimized for inference, which is the reason why this thesis project was initiated.

For in-depth literature on the model creation process, we refer the interested reader to the papers by Mattias Nyberg and Carl Svärd [5] and Erik Lundqvist [6], which originate as a result of cooperation between Linköping University and Scania AB.
1.3 Queries: definition and notation

The concept of queries on Bayesian networks is central to this thesis, therefore a formal definition and introduction to the notation used in this thesis report is given here.

Notation of sets of nodes and evidences

In general, let a capital letter like N denote a set of nodes, and N (decorated with a dot) a set of instantiated nodes (nodes together with their selected states). In this thesis report, T is used to denote a set of target nodes, E is a set of evidence nodes, and Ė (decorated with a dot) is a set of evidences.

Queries

A query seeks to calculate the posterior probability distributions of a set of target nodes, given a set of evidences. A query q therefore consists of two elements: the set of target nodes T, and the set of evidences Ė. In other words, the query q = (T, Ė) seeks to calculate P(t|Ė) for each variable t ∈ T.

The result r_q of a query q is a set of probability distributions, one for each variable t ∈ T.

The following concepts will also be used in this thesis:

Joint probability distributions

A joint probability distribution is a probability distribution over a combination of several variables, instead of a set of separate ("marginal") distributions for each variable. We notate as follows:

A query for a joint probability distribution jq = (T, Ė) seeks to calculate P(T|Ė), or P(t_1,t_2,...,t_n|Ė). The result r_{jq} is a single joint probability distribution over the combination of the variables in T.

Query interfaces

The concept of query interfaces is an abstraction of the concept of queries. A query interface on a Bayesian network defines which nodes can be used as target nodes by queries, and which nodes can be used as evidence nodes.

More formally, a query interface Q = (T, E) defines a general set of possible target nodes T and a set of possible evidence nodes E, such that, for each query q = (T, Ė) using the query interface Q, T ⊆ T and E ⊆ E.

Implicitly, a query interface Q = (T, E) also defines the set of auxiliary nodes A, the nodes which are not going to be used by queries at all, which is the set of nodes of the network that are neither possible targets nor possible evidence nodes.
2 Analysis

Part two of the report describes the initial analysis phase of the project. Chapter 2.1 takes a closer look at the Bayesian network model that was used for the case study, pointing out its particularities, and explains how it is going to be used (what kinds of queries are going to be posed to it by the troubleshooter). Chapter 2.2 introduces different algorithms (exact as well as approximate) used to calculate posteriors in Bayesian networks. Chapter 2.3 looks at what tools and software for using Bayesian networks exist in practice, with a sub chapter paying special attention to SMILE, the library that was used for the project.
2.1 A closer look at the model of the case study

The Bayesian network provided by Scania for the case study is a model of a truck system called **Selective Catalytic Reduction**, or short SCR. The SCR system injects urea into the exhaust of the diesel engine before it leaves the truck. It consists of sensors, filters, pumps, a reagent tank, and a dosing control unit, among other things. Due to the way the model is built (see Building Bayesian networks for troubleshooting on page 12), it consists of four types of nodes: Hardware Components, Hardware Services, Software Services and Diagnostic Trouble Codes.

Hardware Components (HWC) represent the actual physical components of the system. These nodes are the root nodes of the graph, they are non-deterministic, and contain the prior fault probabilities of different faults in each component. Each HWC variable has one non-faulty state and varying numbers of faulty states (usually one, but up to five). The prior probabilities of the fault states are very small, usually around $10^{-8}$.

Hardware Service (HWS) and Software Service (SWS) variables come from the error propagation tree of the software components. The error propagation tree shows how errors propagate in the system's software architecture. Every HWS variable is connected to its HWC variable, and basically is a deterministic pass-through node representing the fact that a hardware component is providing its service if it does not have faults. Every service variable (HWS and SWS) has one state representing "working normally" and another state representing "not working". Actually, the general methodology of Scania's error propagation trees allows for more fine-grained levels in between, say "limited functionality", but the service variables in the used model all happened to be binary. Also, their CPDs are all deterministic and simple MAX\(^1\) gates of their inputs (a service does not work when one of the services it depends on does not work). Again, the general methodology of Scania's error propagation trees allow for more complex logic, and might also support noisy (non-deterministic) logic in the future.

The variables for Diagnostic Trouble Codes (DTC) are directly connected to the HWC variables. They show which DTC can occur due to which faults in the hardware. Some of them have a lot of parents (up to nine in one case). Also they were deterministic MAX gates, which will probably not be the case for models of other truck systems once Scania's methodology of building these models is fully developed. DTC records can be directly read out from the truck's board computer, so it is expected that the troubleshooter in action will usually have lots of evidence available on these nodes.

---

1 A MAX gate is a generalization of the logical OR gate for multivalued variables. The HWS and SWS (and the DTC) are all binary variables, but the HWC are not binary in general. If the HWC were reduced to binary variables ("faulty" and "non-faulty"), the rest of this network would resemble logical OR gates.
Illustration 4: The Bayesian network model of the SCR system used in the case study.

Illustration 4 shows a low-resolution screenshot of the SCR model in GeNIe, to illustrate the model's graphic and semantic structure. It is laid out bottom-up, so the nodes at the bottom are the root nodes and the arcs point upwards. The orange nodes are HWC, the gray ones are their corresponding HWS variables, the green ones are the SWS nodes and the yellow ones are the DTCs. Note that the DTC nodes are all directly connected to the HWC, while the HWS and SWS from the error propagation description form a more fine-grained service structure with internal variables. The service structure ends in the topmost node which represents overall system functionality.

The model consists of 72 HWC, 46 HWS, 38 SWS and 35 DTC nodes, a total of 191 nodes.

To wrap it up, the model is not a decision network / influence diagram, it is no DBN, and it only contains discrete nodes. Most nodes are deterministic Boolean functions, except for most of the roots which are chance nodes and have states with very small prior probabilities. According to Scania, the deterministic relationships can also be more complex, multi-valued formulas in error propagation trees of systems other than the SCR, and in the future they might be defined non-deterministic as well.

In the provided model, the Boolean gates were not explicitly defined by noisy-MAX parameters, but with a full truth table instead.

2.1.1 Usage Profiles

A usage profile describes the way a BN is going to be used in general. Here, the BN will be used by the surrounding diagnoser in "query mode". There are two basic kinds of queries the planner poses to the diagnoser, therefore our case study consists of two use cases, called the "diagnostic" and the "causal" use case, due to the direction of reasoning in the graph. The usage profiles of these use cases are described as follows:

1. The diagnostic use case
Initially, the diagnoser has to calculate the probabilities of the faults in the hardware components, given the driver's and the mechanic's observations about the system's external behavior, and the DTC that can be loaded from the truck's on-board computer. Using HWC, SWS and DTC as names for sets of nodes, the query interface (as defined in chapter 1.3) is as follows:

\[ \mathcal{C}_D = (\text{HWC, SWS } \cup \text{ DTC}) \quad \text{Diagnostic queries (3)} \]

In other words, the queries \( q = (T|\hat{E}) \) in the diagnostic use case will use as targets \( T \subseteq \text{HWC} \), and as evidence nodes \( E \subseteq (\text{SWS } \cup \text{ DTC}) \).

To be exact, the planner does not work with marginal probabilities of hardware faults, but instead with the probabilities of fault combinations, also called joint probabilities. In other words, the actual planner will use \( P(T|\hat{E}) \) instead of \( P(t|\hat{E}) \) for each \( t \in T \). To limit the combinatorial explosion a little, the planner limits itself to reasoning about 'reasonable' combinations of at most three faults. There is a small chapter about calculating joint probability distributions included in the appendix (chapter 7.2, see page 69), but during the practical work on this thesis project, the computation of joint probability distributions was not considered, only marginal distributions, which represents the probability of a fault being present independently of other faults.

This direction of reasoning (against the direction of the arcs, from the leaves to the roots in the graph) is called diagnostic reasoning.

Evidence can be expected to be present on most, if not all, of the DTC nodes, and also a bit more scarcely on some of the SWS nodes. Targets will always be all of the HWC nodes.

Accuracy of the results is very important. These results form the basis for all following calculations, as described in the framework of event-driven non-stationary dynamic Bayesian networks ([2], [3]).

2. The causal use case

After the initial query, the diagnoser has to answer queries that go in the opposite direction. The planner needs to know how the observations are expected to change as faults in the components get repaired. The query interface for this use case is as follows:

\[ \mathcal{C}_C = (\text{SWS } \cup \text{ DTC}, \text{HWC}) \quad \text{Causal queries (4)} \]

In other words, the queries \( q = (T|\hat{E}) \) in the causal use case will use as targets \( T \subseteq (\text{SWS } \cup \text{ DTC}) \), and as evidence nodes \( E \subseteq \text{HWC} \).

This is called causal reasoning, because it goes along the arcs. (Note that this does not necessarily have to correspond to the abstract definition of causality, because variables in Bayesian networks can be connected in both directions and still represent the same joint probability distribution.)

Evidence can be expected to be put on all the HWC nodes, with at most 3 of them being in a faulty state and the rest being in their non-faulty state. Targets might be distributed more or less scarcely among the SWS and DTC nodes.

The accuracy of the results is not as important as query speed, since the planner can produce plans of higher quality when it can pose more queries in the given time.

Because the nodes (except the HWC) are all deterministic functions, the causal reasoning for this model could actually be done with a logic framework (constraint solver). However, Scania wants to keep the troubleshooter tool general enough to work with arbitrary Bayesian networks of systems other than the SCR.

Note, both usage profiles use the same set of nodes, but for opposite purposes: the target nodes for reasoning in the diagnostic direction are the evidence nodes when reasoning in the causal direction. In both cases, HWS are auxiliary variables.
2.2 Inference algorithms

Theoretically, a very straightforward way to compute posterior probabilities in a Bayesian network would be to build the complete JPT (joint probability table) from the network. By summing out entries in the JPT, one can calculate any $P(t|\mathcal{E})$, either by summing up the values for $t$ in all entries that correspond to the evidence $\mathcal{E}$, or by using the definition of conditional probability below, where again both parts of the division can be obtained by summing out entries in the JPT.

$$P(A|B) = \frac{P(A,B)}{P(B)} \quad \quad P(t|\mathcal{E}) = \frac{P(t,\mathcal{E})}{P(\mathcal{E})}$$

Conditional probability (5)

The size of a JPT, however, explodes exponentially with the number of variables in the model, making the computation intractable, which is why Bayesian networks were invented in the first place.

A more practical way to do Bayesian inference is shown with the example in Chapter 1.1 "What is a Bayesian network?". This method is called Variable Elimination. Variable Elimination calculates the posterior distribution of a variable by working directly on the graph structure of the network instead of its JPT. The network gets reduced variable by variable, propagating evidence, summing out variables along the arcs and using Bayes' theorem against the arcs, until the target variable remains. It is, however, NP-hard to decide in which order the variables should be reduced in an optimal way so as to minimize the redundant computational work. Also, the algorithm only computes the posteriors of a single target variable, and has to be restarted for the next one. For a detailed description, see A simple approach to Bayesian network computations [7].

In 1982, Judea Pearl proposed an algorithm called Message Passing [8]. To understand this algorithm, imagine the model being implemented as a network of microprocessors. Every node in the network holds its own probability distribution and can communicate with its neighbors (parents and children) by messages to adjust each other's beliefs. Pearl showed that with one complete message pass through the network (from one end to the other and back again), the nodes' beliefs would arrive at the correct posterior values. This, however, works only in networks whose undirected graph forms a tree: else, messages might be circulating in loops forever. Therefore, this algorithm could be applied to the example networks in Illustration 2 and Illustration 3, but not to the Weather-Sprinkler-Grass example in Illustration 1.

To use the algorithm on a multiply connected network like the one in Illustration 1, you have to make a tree out of it. This can be done by clustering variables into a so-called join tree with the following procedure: First, the graph gets moralized, which means the graph gets undirected and all parents of each node get connected by edges. Secondly, it gets triangulated by adding more edges. An undirected graph is triangulated iff every cycle of length four or greater contains an edge that connects two nonadjacent nodes in the cycle. This also enables the formation of a join tree. In the third step, the join tree is created: Each maximal clique (completely connected group of nodes, which is not part of a bigger clique) in the triangulated graph is represented by a node in the join tree; the contained variables of the clique are combined in the join node by building their joint conditional probability distributions. The resulting graph is an undirected tree where each node represents a group of variables in the original graph. On this join tree, a variation of Message Passing can be executed. In the end, to obtain the marginal posterior distributions of specific nodes, the algorithm has to find the cluster node that contains the node's family (the node and its parents), and sum out its value. There is always a cluster node that contains a node's family, because of the moralization step in the beginning. A detailed and illustrated description of this procedure is given in Inference in Belief Networks: A Procedural Guide [9].

The complexity of clustering algorithms is exponential in the networks' treewidth, which is defined to be the number of variables of the biggest cluster node in its resulting join tree, minus one. Each cluster's JPT size is exponential in the number of its variables. Furthermore, during message passing, each cluster's JPT matrix has to be multiplied with its neighbors' matrices. Therefore, clustering takes long time in networks where a lot of nodes end up in the same cluster. During moralization, each node's
family forms a cluster, so nodes with a lot of parents result in big clusters, and the size of the biggest family in a network can be used as an admissible estimation of its treewidth. The triangulation step is a bit less problematic: the triangulation algorithm is free to decide which nodes it should connect. Finding the optimal triangulation (the one that leads to the smallest increase in treewidth) is an NP-hard problem in itself, but good heuristics exist to find near-optimal triangulations.

Beside Pearl's original Message Passing algorithm, there are other algorithms that only work on a specific subset of Bayesian networks. Another example for this is the Quickscore algorithm [10] which is specifically designed for BN2O networks. BN2O networks are Bayesian networks that consist of two layers only, one layer of causes directly connected to the other layer of effects, and all the effect nodes are simple noisy-OR gates. Quickscore would not be applicable to the network of the case study, and is not included in the SMILE library.

**Approximate inference**

Clustering is still NP-hard, and since exact results might not always be needed, there exists a plenitude of approximate algorithms as well. Approximate algorithms try to find a good estimate for the exact posteriors in less time, usually by stochastic sampling.

Arguably the most straightforward sampling method is called Logic Sampling. The network's nodes get sampled from the roots to the leaves, giving each node a random value according to its CPD. Passes that do not correspond to the evidence get discarded. In the end (after enough passes have been conducted), the frequency of the values give an estimate for the nodes' posterior probability distributions.

This can be varied in a lot of ways. Samples can be weighted instead of discarded. Instead of complete passes, MCMC sampling can be used. MCMC sampling only re-samples one node's value at a time.

Every algorithm works well on some networks and poorly on others. A typical weakness of all sampling algorithms is when nodes' values have very small prior probabilities, or when the evidence is unlikely.

An approximate algorithm that does not use stochastic sampling is called Loopy Belief Propagation (LBP) [11]. In essence, it is a variation of Pearl's original Message Passing algorithm for polytrees, where the circulating messages are stopped after a while. LBP does not always converge, and if it converges there is no guarantee that the values are correct. LBP is not mathematically sound, yet it often yields good results in practice. The conditions for LBP to work are not clearly understood.

There exists a proof [12] that converging the approximation of stochastic simulation algorithms towards the exact values does take exponential time as well (so, Bayesian inference is still NP-hard).

Literature on Bayesian inference is collected in the subchapter 7.4.2 On Bayesian inference in the literature review section in the appendix.
2.3 Software review: Bayesian network tools

There are several tools available for using Bayesian networks. Some are commercial, some freeware; some are open source, others closed. Care has to be taken when finding lists of tools for Bayesian networks: a lot of the tools (and lists) are outdated.

**GeNIe & SMILE** is a tool developed at the Decision Systems Laboratory (DSL) at the University of Pittsburgh (Pennsylvania, USA). SMILE stands for *Structured Modeling, Inference and Learning Engine* and is a closed-source C++ library which is available for free as binaries for different operating systems. GeNIe is DSL's own Windows GUI for it. The tool is free for use. The library has also wrappers for Java, C# and more. Homepage: [http://genie.sis.pitt.edu/](http://genie.sis.pitt.edu/)

**SamIam** ("Sensitivity Analysis, Modeling, Inference and more") is another GUI for SMILE, written in Java at the University of California. Homepage: [http://reasoning.cs.ucla.edu/samiam/](http://reasoning.cs.ucla.edu/samiam/)

**Hugin** is commercial software by Hugin Expert A/S, a company specialized in decision support systems. There is a freeware trial version called *Hugin Lite* which is restricted to networks with a maximum of 50 nodes; also, it is licensed for evaluation use only. Hugin has an API with wrappers for multiple programming languages. Homepage: [http://www.hugin.com/](http://www.hugin.com/)

**Netica** is commercial software by Norsys Software Corp. There also is a freeware trial version (to be exact, the full features have to be unlocked by purchasing a key). Netica features an API with wrappers for multiple programming languages. Homepage: [http://www.norsys.com/](http://www.norsys.com/)

**MSBNx** is a Bayesian network tool by Microsoft Research, with a COM-based API that is recommended for use with Visual Basic and JScript. MSBNx is free for non-commercial use. Homepage: [http://research.microsoft.com/en-us/um/redmond/groups/adapt/msbnx/](http://research.microsoft.com/en-us/um/redmond/groups/adapt/msbnx/)

**BNT (Bayes Net Toolbox) for Matlab** is a collection of Matlab scripts and C program modules for using Bayesian networks. The BNT is free and open source software, published under the GNU Library GPL. Last updated 2007. Homepage: [http://code.google.com/p/bnt/](http://code.google.com/p/bnt/)

**BNJ, Bayesian Network tools in Java**, is an open-source software suite for Bayesian networks written in Java, and has a GUI using SWT. Built by the Kansas State University. Last updated 2004. Homepage: [http://bnj.sourceforge.net/](http://bnj.sourceforge.net/)

**Elvira** is free open source software, written in Java. It is the product of a project among Spanish universities. Homepage: [http://leo.ugr.es/elvira/](http://leo.ugr.es/elvira/)

There are also several file formats for saving Bayesian networks. Every tool has its own native format, and features import/export to other formats to varying degrees (often, only a rudimentary version of the file format definition). There are efforts for having an industry standard file format, but until now the only tool that really supports it is MSBNx by Microsoft.

**Case study**

Since we needed to run automated tests, we were only interested in libraries, not GUI tools. The software used should also be free (free of cost and free for eventual later commercial use by Scania).

The network for the case study came in XDSL format which is the XML-based format of DSL’s tool GeNIe & SMILE. It was therefore natural to use the same tool for the tests. Hugin Lite does not support more than 50 nodes in its network, which was not enough for us (see chapter 2.1 A closer look at the model of the case study), and GeNIe’s export to Netica’s file format NET did not work satisfyingly well. While it would have been interesting for the project to be able to compare the features and performance of several libraries, considering the project’s time constraints it was decided to limit the project to the use of the SMILE library.

2.3.1 SMILE in-depth

The main C++ classes of SMILE are **DSL_network**, which represents a Bayesian network, and
DSL_node which represents a node in the network. All class names are prefixed with "DSL_".

The DSL_network class has a method to set the inference algorithm that should be used for belief update (as SMILE supports several algorithms), methods to set flags for the types of relevance reasoning that should be used, and a method to set nodes as targets.

A node in the graph is represented by a DSL_node object. Within the DSL_node object, the node's definition as a probabilistic variable is encapsulated in a DSL_nodeDefinition object. This contains the number and names of the variable's possible states, and its CPD. The node's current value is represented by a DSL_nodeValue object, also within the DSL_node object. A node's value can be its posterior probability distribution, be set to an evidence, or be invalid.

The basic API mechanism for Bayesian inference in SMILE works like this: First, node values are set to their evidence states. Then, upon call of the method DSL_network::UpdateBeliefs(), the defined inference algorithm will calculate the posterior values of all nodes which are set as target nodes and whose values are invalid. UpdateBeliefs first performs a relevance reasoning step to prune the network from irrelevant nodes, before the defined inference algorithm is executed.

### Inference algorithms

SMILE supports the following inference algorithms for (discrete) Bayesian networks. References to the corresponding papers are given in the list; see also chapter 7.4.3. Papers describing the inference algorithms in SMILE in the appendix. The algorithm names given here will be used throughout the thesis report.

- **Lauritzen**: exact inference, the standard clustering algorithm as developed by Lauritzen and Spiegelhalter. [9]
- **Henrion**: approximate inference. Not extensively documented (the corresponding paper is not available online), but it is a variant of Logic Sampling. [13]
- **Pearl**: exact inference, executing Judea Pearl's Message Passing algorithm directly on the network's graph structure, which works only on polytrees. (If the network is multiply connected, UpdateBeliefs will fail with a corresponding error message.) [8]
- **LSampling**: approximate; another logic sampling algorithm like described above. [14], [15]
- **SelfImportance**: approximate; sampling weighted by self-importance. [14], [15]
- **HeuristicImportance**: approximate; sampling weighted by a heuristic function.
- **Backsampling**: approximate inference. This algorithm reverts the arcs from the evidence nodes outwards, then perform sampling on the modified network. The advantage of this is that it solves the problem of improbable evidence, which usually effects all sampling algorithms, because all evidence nodes are roots. The disadvantage is that arc reversal costs time and often adds further arcs to the network (necessary to keep the JPD), which also increases time of the sampling step. [16]
- **AISSampling**: weighted sampling with a learning phase up front to learn the weights. [17]
- **EpisSampling**: improved AIS that works without the learning phase. [18]
- **LBP**: loopy belief propagation (approximate algorithm). This algorithm option is deprecated as it is buggy and DSL does not plan to fix it. [11]
- **LauritzenOld**: old implementation of Lauritzen, deprecated but left in the library for performance comparisons. The new implementation has a better performance. [9]
- **RelevanceRngLinDec**, or **LinDec** for short: this option performs relevance-based linear decomposition of the Bayesian network, then uses Lauritzen to calculate posteriors. Internally, this works by executing inference several times with only a subset of the target nodes activated each time. This can speed up inference because it avoids large cliques: relevance reasoning can
prune larger parts of the network for each partial query. The default size for target groups is 32 nodes. [19]

- RelevanceRngRecDec, or RecDec for short: similar to LinDec, but uses recursive decomposition: the set of target nodes gets split in two halves; and for each half, if the resulting join tree's size is above a certain threshold (default 65536), it recurses and splits that group into two again. [19]

Notes on Lauritzen clustering: The clustering is performed after the pruning step of UpdateBeliefs. Therefore, the complete procedure of moralization, triangulation, and message passing is run each time UpdateBeliefs is called; SMILE does not keep the join tree in the background between the queries in any way.

Furthermore, there is a fallback scheme between the Lauritzen clustering algorithms: standard Lauritzen will fallback on LinDec if the graph couldn't be triangulated using all targets at once. LinDec in turn can fallback on RecDec for a target group if the resulting join tree's size is above a certain threshold (also this one is by default 65536).

Relevance reasoning

Some care has to be taken when using the term "relevance reasoning" in SMILE, because it can mean two different things. First, relevance reasoning is performed when changes are made to the network, for example when the graph structure is changed, a CPD is changed, the number or order of a node's outcomes is changed, or when evidence is set (or changed or cleared). The way Scania's troubleshooter uses Bayesian networks does not change the network itself, so the only action that triggers this form of relevance reasoning is setting the evidence. SMILE propagates evidence through deterministic nodes and invalidates the posterior values of nodes which can be changed by the added evidence (nodes to which the added evidence is relevant). SMILE distinguishes between evidence that is set on nodes explicitly, and evidence that is propagated from explicit evidence on other nodes.

The second kind of relevance reasoning is used within the method UpdateBeliefs. SMILE reasons about which nodes are relevant for the target nodes, in order to internally simplify the network before the actual inference (e.g. clustering algorithm) takes place. Irrelevant nodes get pruned, thus making the network smaller. The algorithms RelevanceRngLinDec and RelevanceRngRecDec take this even one step further by decomposing the set of target nodes into separate subproblems.

In places where the term "relevance reasoning" might be ambiguous, it will be called "evidence processing" respectively "pruning" in this thesis report.

Because SMILE uses a kind of join tree that needs CPTs, it converts noisy-MAX definitions to CPTs before inference. Also, deterministic nodes are internally held as CPT nodes. After reading this in the documentation, our impression was that noisy-MAX definitions do not give any advantage for inference itself (rather a little disadvantage, because of the conversion step). This however turned out to be irrelevant, as huge benefits can be gained from noisy-MAX definitions during relevance reasoning.
3 Approaches to query optimization

A troubleshooter, or any application that uses Bayesian networks for probabilistic queries, can be divided into several levels, and on all these levels it can be possible to optimize the application's overall performance. Here is a brainstorming list of ideas where to start approaching the problem on different levels, from the lowest to the highest:

- **Implementation level**: try to tweak and optimize the given algorithm's implementation to work faster.

  If you know the models that are going to be used by the application, or if you know general properties of the models that are going to be used, you can try to use this knowledge in order to specifically tweak the algorithm geared towards these models.

  A lot of research has already been conducted in order to optimize the algorithms at hand; and since there are a lot of algorithms available that work well for different kinds of models, the question of using knowledge about the models to be used might better be solved on the tool level. In other words, it is more practical to look at which of the existing algorithms works best with your model.

  One issue on the implementation level that we see potential in is parallelism. Many inference algorithms lend themselves to parallel implementations; in fact, Pearl originally proposed his message passing algorithm to be implemented by a network of independent microchips working in parallel; and sampling algorithms can be run on massively parallel supercomputers. One reason this is not researched a lot is the fact that there are not really that many practical Bayesian networks that would profit from this, because building a practical Bayesian network that is so big that it needs a supercomputer is a problem in itself. Also, in many application scenarios, such as a troubleshooter in a truck workshop, supercomputers are simply not an option.

  Another idea is to make use of noisy definitions during message passing, that is, work directly with the parameters instead of the fully expanded CPT. This can be possible depending on the exact type of join tree that is used. The type of join tree used by SMILE does not support this.

  Yet another idea is to keep the join tree cached between queries, instead of recreating it for every belief update. This is not done in SMILE because the applied relevance reasoning can prune the network in different ways depending on the evidence, so the join tree is different for each query. Research conducted by DSL suggests that this is much more efficient in general. Note that the whole process of relevance-based pruning, clustering (building the join tree), and message passing is opaque to the SMILE user (not exposed in the API).

- **Tool level**: This may be called "How to use the library in the most efficient way". Which library should you use? Which of the available inference algorithms performs fastest? How should other flags, like for relevance reasoning, be configured? These settings will collectively be called "library options".

  Also when approaching the tool level, you might make use of the knowledge you have about the particularities of the models that are going to be used.

- **Model level**: This level requires that the application does not have to deal with sudden queries, meaning that a user can load any model at runtime and the tool has to answer random queries in the fastest way possible. Instead, you have time available during a preprocessing phase where you can assess and actually modify the model itself in order to optimize it for inference.

  If you furthermore create the models that the application will use yourself, you can also look at
how to create more optimal models right from the beginning.

On this level, it might be of use to you to know what kinds of queries the application will pose to the Bayesian network. A model could be optimized in different ways for different kinds of queries.

– **Application level**: Of course, you can always ask yourself whether you should use Bayesian networks at all, or maybe not all the time. For example, if you suspect that the planner part of the troubleshooter application might pose the exact same queries to the network several times during its heuristic search, you may consider caching the results of queries in a hashtable. Or, if your networks consist of deterministic nodes only, you might consider using a different framework altogether, like inference engines for propositional logic.

To set the scope of this project, we decided to focus on the tool level and the model level. This means that we, on one hand, used an existing library instead of writing our own inference algorithms (we used SMILE), and on the other hand we took as granted that the outer application works as it should and the queries it poses are all necessary. Within this scope, the problem of optimizing the diagnoser reduces to the problem of benchmarking the execution time of queries for different options available in the library, and for models modified in different ways. In the benchmarking framework, the optimizations on tool level and model level are closely intertwined: models modified in different ways may perform better with different library options. The next chapters will explain the details of our preprocessing approaches.
3.1 Preprocessing Bayesian networks

Preprocessing in this thesis report means to analyze and modify the model before it is used in an end product. The time that a preprocessing step needs therefore does not matter much. The following sub-chapters present ways of preprocessing that were applied to the case study.

3.1.1 Divorcing

Divorcing means splitting the parents of a node. Helper nodes get added between a node and its parents. The idea is to break up a big family into several small ones, so that the resulting join tree of clustering algorithms will contain a few more small clusters instead of one big cluster. This is expected to increase performance because the speed of clustering algorithms is dependent on the cluster sizes, and the size of a cluster node is exponential in its number of variables. Since all nodes of a family (due to moralization) end up together in at least one cluster node, a big family in the Bayesian network will result in a big cluster node in the join tree.

Divorcing can not in all cases be done without losing information (which means that the queries answered by the modified network will not be completely accurate). Actually, it is possible to do so by using an exponential number of states in the helper nodes – but then no performance improvement can be expected. If you want to keep the number of states in the helper nodes the same as in the child node, groups of parents can only be divorced without loss if they are causally independent. This in turn means that nodes with noisy-MAX definitions can always be divorced without loss into binary trees, leaving a maximum family size of 3. Therefore, a simple approach to divorcing a node is to first convert its CPT definition to an ICI definition like noisy-MAX, and then divorce the modified node into a binary tree.

Since noisy nodes already define their parents' influences in an independent way, divorcing them is a trivial process and all information is preserved (the numeric values do not have to be touched at all). For reasons of aesthetics and human comprehension, a divorcing structure of a balanced binary tree is recommended; the other straightforward option is a chain of helper nodes. Independent of the structure chosen, the resulting nodes' CPDs are to be defined in the following way:

- All added helper nodes have noisy definitions.
- All nodes have the same states as the original child.
- The leak value of the original child stays in the "final" child (the leaf child of the resulting divorce tree); all other leaks are 0.
- The parents' influence strengths of the original node stay directly "below" the parents (they get assigned to the node of the divorce tree that is directly connected to the parent). All other influences are deterministic "pass-through" influences, which are 1 where the states are the same, and 0 where the states are not the same.
Illustration 5 shows an example of divorcing a noisy-MAX node. Above is the noisy-MAX node X with parents A, B and C, which below is divorced into a balanced binary tree with the added helper node AB. The colors of the parameters show which parents they belong to. The parameters in pink are the deterministic pass-through parameters that get added because of the helper node: the empty leak parameter in the helper node and the deterministic influences in its child X. As can be seen, the original leak parameter stays in the final child, while the original influences end up in the children directly below the original parents.

**Verification**

A verification step is only necessary to test the correctness of the implementation of the divorcing algorithm. This section is therefore only interesting to the reader who wants to comprehend or reproduce the implementation of the procedure.

In order to verify the correctness of the implementation of this procedure, it is not possible to compare the noisy definitions of each node before and after, or their resulting CPTs, since the structure of the network changed. On the other hand, it is not necessary to form the model's old and new JPT (joint probability table), sum out the added nodes of the modified model and compare them. Also, it would be infeasible. Since the structural changes of the network "stay within the families" (i.e. only the structures between nodes and their parents are changed), it suffices to compare the "indirectly conditional" probability distributions of the model's families, given the original parents. This means that, after building the divorce tree, one has to obtain again the probabilities of the child's states conditioned on the "final" parents of the divorce tree, which were its parents in the original model; thereby making the intermediate divorce helper nodes transparent. Because the family-internal structure is a tree, this table can be obtained by a recursive algorithm that integrates each parent's CPT into the
child's CPT by multiplication, stopping the recursion at the original parents.

These "transparent" CPTs of the resulting model should be equal to the CPTs of the original model. There are several ways to define distance metrics or divergences between CPTs, but since the process is information-preserving, the difference should be zero with any of them, so simple Euclidean distance is sufficient for this purpose.

### 3.1.2 Conversion to noisy-MAX definitions

Converting nodes to noisy-MAX definitions means changing a node's definition from a full CPT to a definition by noisy-MAX parameters. However, not every full CPT (of exponential size in the number of parents) can be perfectly fit with noisy-MAX parameters (which are linear in number). However, it has been proven [20] that for every CPT there exists exactly one best-fit noisy-MAX definition. SMILE provides a linear gradient algorithm to find the best-fit noisy-MAX parameters to a CPT, given an ordering of the states of the node and its parents. The internal workings of this conversion algorithm are described in a paper by Zagorecki and Druzdzel [20]. So, in order to automatically convert any CPT node to noisy-MAX, one has to iterate over the possible combinations of state orderings and see which ordering yields the noisy-MAX parameters that provide the closest fit to the original CPT. Also, a threshold can be defined such that, if no noisy-MAX parameters can be found that approximate the original CPT close enough, the node shall remain in CPT form in order not to lose too much accuracy by the modification.

Conversion of nodes to explicit noisy-MAX definitions was expected to speed up inference for two reasons: first, it enables further relevance reasoning that uses the properties of amechanistic ICI gates (as described in subchapter 1.1.5 Independent influences), and second, it enables simple divorcing of a node’s parents (as mentioned above). Actually our results suggest these two things mean the same thing.

**Partial noisy divorcing**

In the model of the case study, all nodes with parents represented simple Boolean OR or AND gates, so the linear gradient algorithm mentioned above was always able to find a perfect fit of noisy-MAX parameters. This however need not be the case in Bayesian networks in general. A CPT can also have a more complex underlying Boolean formula. If the underlying formula is, for example,

\[ X = A \lor (B \land C) \]

then \( X \) could be divorced without information loss into two noisy-MAX nodes representing

\[ X = A \lor \neg Y \]

\[ Y = \neg B \lor \neg C \]

Illustration 6: A CPT representing the Boolean formula \( A \lor (B \land C) \)

\[
\begin{array}{|c|c|c|c|c|}
\hline
& A & B & C & \text{PND (6)} \\
\hline
& \text{true} & \text{true} & \text{true} & \text{true} \\
\hline
& \text{true} & \text{true} & \text{false} & \text{false} \\
\hline
& \text{false} & \text{false} & \text{false} & \text{false} \\
\hline
\end{array}
\]

\[ X = A \lor \neg Y \]

\[ Y = \neg B \lor \neg C \]

However, taking the two-step approach of first converting a node to a single noisy-MAX gate and then divorcing it, this possibility will not be found and inaccuracies are introduced into the network. So, model optimization would gain from a method that performs conversion and divorcing as a combined task. This is, as far as we know, not researched yet, but was also not further pursued in this project because, as said, the nodes in the network at hand all happened to have underlying distributions of simple OR or AND gates, and so partial noisy divorcing was beyond the scope of this project.
Representation of the disjunctive normal form

As a side note, it should be mentioned that any CPT can be represented by a structure of noisy-MAX gates that resemble its disjunctive normal form (DNF), but this approach requires adding an exponential number of nodes.

To understand this, remember that the noisy-MAX is a generalization of the logical OR gate (with possible negations of its inputs and output) for use with multi-valued probabilistic variables. Any arbitrary Boolean function can be written in disjunctive normal form, which is a disjunction of conjunctions of input literals. Based on the DNF formula, the function can be implemented using a two-layered structure of logical gates, one layer of AND gates with one AND gate for each conjunction, and a second layer consisting of a big OR gate.

This principle can be generalized to noisy-MAX gates in Bayesian networks, but since the size of the DNF formula is exponential in the number of the function's input variables, such a structure would need a number of nodes exponential in the variable's number of parents. Therefore, this idea does not look promising; it is mentioned again in the brainstorming list of chapter 5.3 Issues and future work for the sake of completeness.

3.1.3 Removing unnecessary arcs

If a parent does not have any influence on a child's probability distribution, i.e. if the CPT entries are always the same for different states of the parent, then the arc between the parent and the child can be removed without loss of information. Also, if the CPT entries are very similar for different states of a parent (differences are below a given threshold), then the arc can be removed with minimal information loss.

This was expected to result in speedup because it reduces family sizes.

It might surprise that a Bayesian network contains such redundancy; and it would not occur in machine-learned models, but if the model is script-generated this is likely to happen. In the provided SCR model of the case study, 22 of its 238 arcs were found to be unnecessary, and removing them yielded a significant speedup.
3.2 Preprocessing specific to the query interface

In many cases where a Bayesian network is used for routine probabilistic reasoning by some surrounding application, the application's usage of the Bayesian network is known. There are some nodes which are typically observable, some nodes which are typically targets of the queries, and some nodes which are typically auxiliary. Thus, from a (verbal) description of the application's usage profile, a formal query interface can be derived which then can be used to optimize the model specific to the usage in an automated way.

The formal definition of a query interface (or short QI) can be found in chapter 1.3 Queries: definition and notation on page 13. Essentially, a query interface \( \mathcal{Q} = (\mathcal{F}, \mathcal{E}) \) defines a general set of possible target nodes \( \mathcal{F} \) and a set of possible evidence nodes \( \mathcal{E} \), such that, for queries using this query interface, \( T \subseteq \mathcal{F} \) and \( E \subseteq \mathcal{E} \).

Model modifications that do not know about the QI have to keep the full JPD (joint probability distribution) of the model intact. They can introduce approximations (like the conversion of CPTs to noisy-MAX definitions) and additional variables (like divorcing), but variables cannot be removed because they might be the target or evidence node for queries.

To formalize this, let "org" be the original model and "mod" the modified one. When \( N \) is a set of nodes in the Bayesian network \( m \), then \( \text{JPD}_m(N) \) denotes the joint probability distribution of these nodes (where those nodes not in \( N \) are marginalized). Using this notation, (backward) equivalence can be defined as:

\[
\text{JPD}_{\text{mod}}(N_{\text{org}}) = \text{JPD}_{\text{org}}(N_{\text{org}})
\]

where \( N_{\text{org}} \) is the set of all nodes in the original model. The JPDs may be exactly equal, or just approximately within a certain threshold.

On the other hand, a model modification that is aware of the QI can relax this condition somewhat to what could be called interface equivalence: queries that use the given QI have to produce the (approximately) same results when using the modified network. In other words, the conditional JPD of \( \mathcal{F} \cup \mathcal{E} \) has to remain intact. Let "mod(\( \mathcal{Q} \))" denote the Bayesian network modified with consideration of the query interface \( \mathcal{Q} \).

\[
\text{JPD}_{\text{mod(}\mathcal{Q})}(\mathcal{F} \cup \mathcal{E}) = \text{JPD}_{\text{org}}(\mathcal{F} \cup \mathcal{E})
\]

Interface equivalence (10)

For verification of the correctness of the implementation, instead of comparing the JPT it might be easier to verify it empirically, that is, see that queries on the modified model produce the same results as on the unmodified model.

Three model modifications that make use of the information provided by a QI were considered: marginalizing auxiliary nodes, reverting arcs into the "causal" direction from evidence nodes to targets, and making the network a bipartite graph (as supported by GeNIe's diagnostic framework). The marginalization of auxiliary nodes was further pursued in this project. Reverting arcs did not promise to yield much benefit because we got quite similar best-case query times for our two use cases of causal and diagnostic reasoning (both were around 2ms); the somewhat radical step of completely restructuring the network into a bipartite graph did not work because it would have produced nodes with more than 18 parents, which made the bipartite graph generation impossible and also would rather have resulted in far worse inference times than in the original model.

3.2.1 Marginalizing auxiliary nodes

Marginalizing means integrating a node's CPD into those of its neighbors, often introducing additional arcs among the neighbors, and then removing the node.

Auxiliary nodes are not of direct interest to the queries, and thus they can be removed. Removing a node can be expected to yield speedup if the family sizes in the model decrease by the marginalization.
The "sum of family sizes" was used as a heuristic here to estimate query time. To give a formal definition: Let $N$ be the set of all nodes in the network, and $|n|$ denote the number of states of the node $n$.

$$\sum_{n \in N} |n| \prod_{p \in \text{parents}(n)} |p|$$  \hspace{1cm} \text{Sum of family sizes (11)}

The sum of family sizes is equal to the total number of CPT entries in a Bayesian network. Due to the moralization step during clustering, the number of parents in the biggest family of a network is an admissible heuristic for its treewidth. Since the sum of family sizes is exponential in that number, and the complexity of the clustering algorithm is exponential in a network's treewidth, the sum of family sizes can be used to qualitatively estimate query time.

In the SCR network of the case study, the HWS nodes were the auxiliary nodes in both use cases. The HWS nodes are often just "copies" of their corresponding HWC nodes, stating that the component is fault-free if it does not contain faults, and passing this information on from their single parent to their single child node. These nodes could be removed in many cases, usually reducing the sum of family sizes by 2.
3.3 Library options

This chapter describes four options available in the SMILE library that can increase or decrease performance.

3.3.1 Targets

Target nodes can be explicitly set in SMILE. SMILE uses this knowledge to save time by avoiding calculations that are of no relevance to the posterior values of the target variables. Before clustering (or sampling, for that matter) takes place, the network gets pruned of irrelevant nodes. To find out which nodes are relevant to the targets, SMILE uses algorithms for relevance reasoning.

If no nodes in the network are set as target nodes, SMILE will compute the posteriors of all nodes.

3.3.2 Relevance reasoning

There are flags in SMILE that activate/deactivate relevance reasoning at certain occasions like changes in the graph structure or the CPD of a variable. The troubleshooter application only uses the network in what we call "query mode", which means it does not change the structure or JPD of the model and only uses it to answer queries. The only occasion of interest in "query mode" is the setting and clearing of evidence, whereby SMILE distinguishes between the setting and clearing of explicit evidence, and the setting and clearing of propagated evidence. By default, during SetEvidence SMILE reacts on the evidence set explicitly, by invalidating the values of dependent nodes. During the benchmarking tests, we also tried the options to reason about both explicit and propagated evidence, and not to perform relevance reasoning about either kind of evidence.

Also during UpdateBeliefs, relevance reasoning is used. This serves to prune the network from irrelevant nodes, so that inference runs faster (as described in the above section about targets). Relevance reasoning on this occasion can not be deactivated by any flags.

There is, however, one more flag in SMILE that activates/deactivates relevance reasoning specific to noisy-MAX nodes; this option is used only during UpdateBeliefs and considers their special mechanistic ICI property: As described in 1.1.5 Independent influences, if the evidence on a noisy-MAX node is set to its "off" state, its parents remain independent of each other (which would not be the case if the node was a general CPT). If this flag is not set, SMILE will treat noisy-MAX nodes like CPT nodes while pruning the network.

Relevance reasoning often leads to improvement of overall inference performance because it helps in simplifying the network and thus avoiding unnecessary computations. Relevance reasoning typically runs in polynomial time and yields substantial savings for the inference algorithms.

To summarize, there are four options provided by the SMILE library: using targets, relevance reasoning about explicit and propagated evidence during evidence processing, and relevance reasoning about noisy-MAX nodes during pruning.
3.4 Methodology

To evaluate a use case, you need the system’s Bayesian network model, along with a description of how the model is going to be used, e.g. what kinds of queries are going to be posed by the surrounding application. We call such a verbal description a usage profile. A usage profile should describe which nodes are going to be used to set evidence and which nodes will be the targets of inference. It should also describe in more detail what the queries will typically look like, for example, evidence will be usually provided on most of these nodes and only a few of those nodes; the observed state is mostly the one that is most/least probable; all queries want to know the posteriors of all target nodes. From a verbal description of the usage profile, you can then derive a query interface (QI) and a more formal definition of the tests to be run for time measurement.

You should also define the goal of optimization. Things to consider include query time, accuracy, and query stability. The most important goal might often be speedup of average or maximum query time, but you may also want to set boundaries on the average or maximum accuracy loss of the results. Query stability means that different queries take about the same time, which makes the application’s performance more predictable.

To measure the times, I built a testbench framework that poses configurable queries to a network and analyzes the results. It is built in C++ like the SMILE API, with Visual Studio. The only Windows-specific code is `QueryPerformanceCounter` for the time measurement, because `clock()` proved not to be precise enough. The testbench was divided into two parts, with the general code in one VC++ project that compiled as a library, and Scania-specific code in a separate project compiled as an executable. The former is FOSS available online (see part 6 Miscellaneous).

3.4.1 The testbench

The testbench serves to automatically modify BN models, to set up series of tests (specifying which queries should be run on which models with which library options), to run tests measuring performance and accuracy, and to analyze these results. The functionality of the testbench includes:

- Model modifications as described in the chapters 3.1 and 3.2: removing unnecessary arcs, marginalizing auxiliary nodes, conversion to noisy-MAX definitions, and divorcing noisy-MAX nodes to binary trees. Modified models are saved into new files.
- Creating a series of tests from a more abstract TestPlan definition; including generation of random but valid evidence.
- Saving test series to files, and loading them.
- Running the tests: loads the model, sets the specified library options, executes the query measuring its performance, and compares the obtained posteriors to the exact results (obtained with an exact algorithm on the unmodified model) by calculating the Kullback-Leibler divergence, to measure accuracy.
- Saving test results to files, and loading them.
- Analyzing the results: summarizes tests with different queries calculating stats like the average time and the standard deviation, produces a ranking table of the fastest test options, and several comparison tables that compare the average performances of different inference algorithms, different models, or different settings for relevance reasoning.
- Saving the result tables to CSV files, so that they can be loaded by spreadsheet applications.

The definition of a "Test" in the testbench includes three things: the model id, the library options, and the query. The model id specifies which model is used (the original model or one of the modified ones); the library options specify which inference algorithm is used, which relevance reasoning flags are set, whether or not relevance reasoning about noisy-MAX nodes is used, and whether or not target nodes are set explicitly; and the query specifies the evidence and the target nodes.
Executing a Test requires five steps in terms of API calls: setting the relevance reasoning flags, setting the algorithm, setting the target nodes, setting the evidence, and calling `UpdateBeliefs`. The framework measures the times of these five steps separately. Executing a Test produces a TestResult. Tests can fail in different ways (or get skipped to circumvent bugs in the library), else a TestResult contains the measured times and the accuracy of the produced posterior probability distributions\(^2\).

To set up a series of Tests, a TestPlan is used. A TestPlan is an abstract definition of the concrete tests to produce. For example, a TestPlan can define that 100 queries have to be generated, with random evidence on a set of nodes and targets on another set of nodes; which options for algorithm and relevance reasoning to use, and which models to use. The TestPlan is then unfolded to a list of tests with one test for every possible combination of options defined in the plan, and the tests are configured with random-generated queries as defined by the plan. The defined TestPlan should reflect the verbal description of the usage profile.

When the tests are run, five times are measured for each test: tRel for setting the relevance flags, tAlgo for setting the algorithm, tTargets for setting the targets, tEvidence for setting the evidences, and tInference for updating the beliefs. The overall query time is the sum of these five time steps. In addition, the error (inaccuracy of posteriors) is measured.

During analysis, one wants to compare the average performances of different test options for different queries. For this, test results of different queries are grouped together, and for each of the seven measurements (tRel, tAlgo, tTargets, tEvidence, tInference, overall query time, and error), five stats are calculated: the average value, standard deviation, minimum, maximum, and median value. Then, all different test options are ranked by their average query time, and of course the fastest one (which does not contain any failed tests) is the recommended setting for the use case. In addition, the following comparison tables are generated: comparison of algorithms (the inference algorithms available in SMILE), comparison of relevance reasoning flags (evidence processing and noisy-MAX relevance), comparison of using versus not using targets, comparison of the models used (the original and the modified ones), and a comparison of models that were modified under consideration of the query interface versus those modified without this knowledge. The last one is generated to draw conclusions about the usefulness of model modifications that make use of the knowledge of the query interface.

There are four model modifications supported by the testbench:

- **remArcs**, the removal of unnecessary arcs, with an epsilon of \(10^{-50}\). This model modification proved to run very fast and is thus recommended to be included in all modified models.

- **margAuxNodes**, the marginalization of auxiliary nodes where the model's sum of family sizes does not increase. This modification makes use of the query interface.

- **det2noisyMax**, the conversion of deterministic and CPT nodes to nodes with noisy-MAX definitions, with a defined threshold on accuracy loss\(^3\).

---

\(^2\) The accuracy of the results is measured in Kullback-Leibler divergence from the obtained results to the exact results; the exact results are calculated using the exact algorithm Lauritzen on the original model. The Kullback-Leibler divergence formula is defined as

\[
\sum_i P(i) \cdot \ln \frac{P(i)}{P_{\text{exact}}(i)}.
\]

When using the KL-divergence for probability distributions where \(P_{\text{exact}}(i)\) can be zero, it is usual to use an epsilon cutoff, which means \(P_{\text{exact}}(i)\) is replaced by \(\varepsilon\) where \(P_{\text{exact}}(i) < \varepsilon\). The testbench uses \(\varepsilon = 10^{-20}\).

\(^3\) The accuracy of the conversion is evaluated by measuring the difference between the variable's original CPD and the new CPD defined by the found noisy-MAX parameters. This difference is measured with a formula that compares two CPTs A and B as follows:

\[
diff(A, B) = \sqrt[\text{cells}]{\sum_{i=1}^{n} (\sqrt{A_i} - \sqrt{B_i})^2}
\]

where \(n\) is the number of cells in the CPTs. This basically is the Hellinger distance, applied to a conditional instead of a posterior distribution.

To avoid accuracy loss, only nodes with a difference value less than 0.01 are actually converted.
– divorce; the divorcing of noisy-MAX nodes into balanced binary trees.

A model that had its unnecessary arcs removed and then nodes converted to noisy-MAX definitions carries the model ID "original-model-id/remArcs/det2noisyMax".

3.4.2 Test setup for the case study

Our case study contains two use cases, one in diagnostic direction and one in causal direction. For both of them, three passes were executed: the first pass with only a few iterations but using all options available, to sort out those which were obviously unnecessary; the second pass with more iterations and the reasonable options, which was used for the main analysis of the use case; and the third pass with even more iterations and only using the fastest options, to get more accurate measurements for the most interesting cases. So we had 6 TestPlans in total.

We used the following 9 models:
– Network1, the original model
– Network1/remArcs, the original model with unnecessary arcs removed
– Network1/remArcs/margAuxNodes, the previous model with auxiliary nodes marginalized
– Network1/remArcs/margAuxNodes/det2noisyMax
– Network1/remArcs/margAuxNodes/det2noisyMax/divorce
– Network1/remArcs/margAuxNodes/divorce*
– Network1/remArcs/det2noisyMax
– Network1/remArcs/det2noisyMax/divorce
– Network1/remArcs/divorce*

*Note: our method of divorcing required prior conversion to noisy-MAX. Therefore, to create non-noisy divorced models, we divorced the noisy models and then converted their definitions (including the definitions of the added divorce helper nodes) back to CPT. So, "Network1/remArcs/divorce" is actually "Network1/remArcs/det2noisyMax/divorce/noisyMax2cpt". The non-noisy divorced models were created to see how much of the speedup of noisy divorced models came from the noisy-MAX conversion and how much from the divorcing. As will be seen below, the non-noisy divorced models were actually the ones that performed fastest.

Note also that it was not necessary to distinguish between models modified for the two different use cases, because their query interfaces contained the same nodes. The target nodes in one use case were the evidence nodes in the other use case and vice versa; in other words, \( \mathcal{J}_D = \mathcal{E}_C \) and \( \mathcal{J}_C = \mathcal{E}_D \), which also means that \( \mathcal{A}_C = \mathcal{A}_D \). The only QI-aware model modification used in the project was margAuxNodes, which marginalizes nodes \( \mathcal{A} \) and thus produces the same modified models for both \( \mathcal{E}_C \) and \( \mathcal{E}_D \).

Generation of test queries

There were two use cases: the queries in diagnostic direction and those in causal direction. Their usage profiles are described in detail in 2.1.1 Usage Profiles on page 16. Here follows a description of how the test queries for each use case were generated in a random way. Admittedly, these did not reflect the verbal descriptions of the usage profiles perfectly, but without a deep understanding of the SCR system, this is difficult in any case. Also keeping in mind that the final planner will use joint posteriors in the diagnostic use case, the executed tests were accurate enough to serve as a proof of concept for the methodology, and to provide insights on what is important for query optimization.

Diagnostic direction: Targets were always all HWC nodes. SWS and DTC are described to be evidence nodes, with evidence on "most" DTC nodes and "scarcely" on the SWS. For the tests, evidence was provided on about two thirds of the DTC nodes, by choosing randomly between evidence on
"Active", evidence on "Passed", or no evidence. The SWS nodes did not have evidence provided.

Causal direction: Targets were always all the SWS and DTC nodes. Evidence was always present on all HWC nodes, with a maximum of 3 HWC nodes being in one of their faulty states and the rest in their non-faulty state. It was chosen uniformly between queries of 0, 1, 2, or 3 faults.
4 Results

Part 4 basically shows the diagrams resulting from the test runs. The diagrams are annotated and commented with our observations and interpretations. The more general and in-depth discussion of the results follows in part 5.
4.1 Usage profile: diagnostic direction

4.1.1 First pass: targets and useful algorithms

TestPlan: The first pass of tests was executed with the following settings (compare 3.4.2 Test setup for the case study):

- 8 random-generated queries that had random evidence on most DTC nodes, and all Hardware Component nodes as targets.
- The queries were executed with or without actually setting the targets in SMILE (if targets are not set, the posteriors of all nodes are computed; see 3.3.1),
- on all 9 models,
- with all 13 inference algorithms available in SMILE (see 2.3.1),
- noisy-MAX relevance switched on and off (see 1.1.5 and 3.3.2), and
- relevance reasoning during evidence processing for explicit evidence, explicit and propagated evidence, or none (see 2.3.1 and 3.3.2).

Altogether $8 \times 2 \times 9 \times 13 \times 2 \times 3 = 11232$ tests, resulting in 1404 data points based on 8 tests each.

Some of these tests had to be skipped because of a bug in the library which made the process crash when using noisy-MAX relevance reasoning on a noisy model, when either LauritzenOld or any of the approximate algorithms was selected. According to DSL, the bug will not be fixed any time soon. 1728 tests (216 data points) were skipped, which left a total of 9504 tests resulting in 1188 data points.

Comparison of algorithms

Illustration 7 shows the comparison of average query times for the different inference algorithms. It is apparent that some algorithms have a similarly good performance, one algorithm performs much worse, and a number of algorithms are missing entirely. The algorithms on the bottom were the exact ones (Lauritzen, LauritzenOld, LinDec and RecDec). Pearl did not work because the network is not a
polytree. The approximate algorithms have serious problems with the deterministic network and the unlikely evidence. Backsampling is the only approximate algorithm that delivered any results at all, the others usually failed saying not enough samples could be generated. Backsampling seems to work best of the approximate algorithms because it does not have the problem of unlikely evidence (as described in chapter 2.3.1 SMILE in-depth and further in Backward Simulation in Bayesian Networks [16]). Backsampling still performs much poorer than the exact ones, so for the second pass only exact algorithms were used.

Conclusion: a tradeoff of time vs. accuracy is NOT an issue for this use case. After obtaining these results, deciding a specific tradeoff value for Scania’s troubleshooter use case was not further pursued.

Using targets

Illustration 8: Benefit of using targets, first pass, diagnostic use case

Illustration 8 proofs that the library can save time by not calculating the posteriors of all nodes. The slowdown is different for different cases; it was not further examined what makes the difference.

There are 2 peculiar outliers to the right-hand side which were not further examined because they couldn't compete with the fastest results anyway. These tests were using Lauritzen, no relevance reasoning (neither for evidence processing nor for pruning), and the noisy undivorced models (one test with Network1/remArcs/det2noisyMax, one with Network1/remArcs/margAuxNodes/det2noisyMax). One possible explanation of this behavior is that Lauritzen's decomposition fallback mechanisms happened to end up with smaller clusters as a different number of targets was used. Due to the unpredictability of this effect in general, making use of it for optimization does not seem to make sense.

Conclusion: in all further passes, targets were used.
4.1.2 Second pass

TestPlan configuration of the second pass:

- 128 random-generated queries that had random evidence on most DTC nodes, and all Hardware Component nodes set as targets.
- The queries were executed setting the targets in SMILE,
- on all 9 models,
- with the 4 exact inference algorithms available in SMILE (which were the ones that proved to be useful),
- noisy-MAX relevance switched on and off, and
- evidence processing for explicit evidence, explicit and propagated evidence, or none.

Altogether 128*1*9*4*2*3 = 62208 tests, resulting in 486 data points based on 128 tests each. Again, some tests with LauritzenOld had to be skipped because of the bug in SMILE. 12 cases (1536 tests) were skipped, leaving a total of 60671 tests resulting in 474 data points.

Comparison of algorithms

Illustration 9, like Illustration 7, shows the performance of different algorithms in SMILE, this time with more exact results, only using the fast algorithms, and sorted by the performance of Lauritzen ascending. It can be seen that LinDec and LauritzenOld are always slower than Lauritzen and RecDec, which seem to have the same performance in almost all cases. This might be explained with the fact that the algorithms fall back on each other.

Conclusion: the library default Lauritzen looks like a reasonable choice, as well as RecDec. The final ranking below will decide the winner (see page 47).
Illustration 10: Effects of relevance reasoning, second pass, diagnostic use case

Illustration 10 shows a performance comparison for different settings of relevance reasoning flags. The points are connected by lines to make it easier to see which points belong together, and to make small differences more visible. To explain the names of the lines: the first character ("n" or ".") shows whether noisy-MAX relevance was used during pruning ("n") or not ("."); the second character ("r" or ".") shows whether relevance reasoning was used during evidence processing ("r") or not ("."). If relevance reasoning was used during evidence processing, the integer value of the flags follows. The flags 463 are the default of SMILE, specifying (among other things) that relevance reasoning is used for explicit evidence, but not for propagated evidence. The flags 316 on the other hand specify that relevance reasoning is used for both kinds of evidence. Explicit evidence is part of the query; propagated evidence occurs when the value of a node can be deduced deterministically from the explicit evidence.

In the results, there is no visible difference between the flags 463 and 316.

It can be clearly seen that relevance reasoning during the setting of evidence is counterproductive. This is because it is performed for every single evidence node that gets set, and does not directly serve to answer the query.

Noisy-MAX relevance plays a big role in noisy models (this can especially be seen in the middle part of the diagram) but makes no difference in non-noisy models (as can be seen well on the right-hand side). In undivorced models, noisy-MAX relevance gives huge benefit, but in divorced models it even slows queries down a bit (see on the left hand side). This is presumably the case because noisy-MAX relevance reasoning can recognize independencies in the noisy models and use them for pruning, while in divorced models the independencies are already clear from the divorced graph structure.

Conclusion: evidence processing is completely counter-productive; noisy-MAX relevance during pruning can yield huge time savings, but can also decrease the performance in some cases.
Comparison of model modifications

Illustration 11: Comparison of model modifications, second pass, diagnostic use case

Illustration 11 shows a comparison of the 9 models modified in different ways. The points are connected by lines to make it easier to see which points belong together.

The original model performs worst in all cases (for all combinations of algorithms and relevance reasoning flags). A substantial improvement is already achieved by removing unnecessary arcs, and an even greater improvement by divorcing. The difference made by removing auxiliary nodes is barely visible. The divorced models are all performing well, no matter whether they are noisy or not. In between, there are undivorced noisy models which often perform bad but can perform as good as the divorced ones: this is the case where noisy-MAX relevance was used for pruning, but no relevance reasoning was done during evidence processing.

Conclusion: it seems that divorcing is a kind of silver bullet. Noisy pruning, a completely different approach, can achieve the same results.
Illustration 12: Speedup by QI-aware model modification, second pass, diagnostic use case

Illustration 12 shows the improvements in speed by the QI-aware model modification of marginalizing auxiliary nodes. Speedup is defined by the following formula:

$$S = \frac{T_A}{T_B}$$  \hspace{1cm} \text{Speedup (12)}

Note that this means that a speedup value of 1 = 100% means that there was no difference in speed.

As can be seen in the diagram, the speedup achieved by QI-aware model modification was mediocre; in some cases it actually slowed down inference. Across all used algorithms, one can see that the divorced models benefited much more from margAuxNodes both in terms of speedup stability and relative speedup. The relative speedup was higher in divorced models because inference runs faster on them to start with, and the absolute speedup seems to be about the same in average. One can see that the best achieved relative speedup was only about 104% – luckily these were cases of good absolute speed as well, so the 104% does have an effect on the best end results.

Conclusion: this result is rather depressing since it suggests that considering the QI for model optimization does not have much potential. What's even more: in some cases, having useless nodes in the network actually increased performance. This may be explained by unpredictable effects on triangulation.
The results seen in Illustration 13 suggest that the single most important optimization is divorcing (along with using a clustering algorithm). The queries also perform well in noisy models with noisy-MAX relevance enabled and evidence processing disabled. So, the important condition is that the model is "divorcable by relevance reasoning". (EP in the diagram means relevance reasoning during evidence processing.)

The diagram shows a lot of jumps in performance, some of which can be explained by the clusters of test options that we recognized and drew onto the X axis. I will not go into explaining every jump in detail here. The fastest cases are those who are divorcable and do not use evidence processing or LinDec.

Illustration 13: Ranking, second pass, diagnostic use case
More stats of query times

Illustration 14 shows the same curve as Illustration 13, with the other stats added (in addition to the average query time, it also shows the standard deviation, and the minimum, maximum, and median values). Luckily, the cases on the left hand side that perform the fastest are also the cases that perform the most stable (meaning that different queries take about the same time, which makes the application performance more predictable).

There were two outliers of maximum query time, the peak in the middle of the diagram. These outliers were not further examined. They were the same peculiar cases that produced the outliers in Illustration 8 (Lauritzen, noisy undivorced models, no relevance reasoning). The outliers were actually around 180ms; they were cut down to 80ms in order to keep the rest of the diagram readable.

Conclusion: Query time stability is not an issue in the fastest cases.
Illustration 15: Ranking with query steps, second pass, diagnostic use case

Illustration 15 again shows the curve of Illustration 13, this time split up into the different steps of API calls. Setting the options for relevance reasoning and algorithm does not take any time; setting the targets takes constant time which does not account for a significant amount of the query time. In the slow cases it can be seen that evidence processing takes an unreasonably large amount of overall query time; it actually does speed up the final inference step but takes too much time itself.
4.1.3 Third pass

Test Plan of the third pass:

- 1024 random-generated queries that had random evidence on most DTC nodes, and all Hardware Component nodes set as targets.
- The queries were executed using targets,
- with the exact inference algorithms Lauritzen, RecDec and LauritzenOld
- on the 6 models that were divorced and/or noisy,
- noisy-MAX relevance switched on and off for the 4 divorced models, and only off for the 2 non-divorced noisy models, and
- no evidence processing.

Altogether $1024 \times 3 \times (4 \times 2 + 2) = 30720$ tests, resulting in 30 data points based on 1024 tests each. Due to the bug in SMILE mentioned earlier, 4 cases (4096 tests) had to be skipped, leaving a total of 26624 tests executed resulting in 26 data points.

Comparison of algorithms

Illustration 16: Comparison of algorithms, third pass, diagnostic use case

Conclusion for Illustration 16: Lauritzen and RecDec are competing for the fastest computations; however, Lauritzen always wins in the important cases.
The final diagram to the diagnostic use case shows the best speedups each model has achieved, compared to the original model. Removing unnecessary arcs yields a 132% speedup already; however, to achieve a speedup of around 600% the network has to be divorcable. As can be seen on the top, the best model was the one with arcs removed, auxiliary nodes marginalized, parents divorced but non-noisy nodes, at 716% speedup (closely followed by the same without auxiliary nodes marginalized, at 700%). The relative speedups were calculated by dividing the modified models' best times through the best time achieved with the original model. "Best time" here means the average query time that was achieved with the respectively best combination of library options. The line 9 at the bottom, which appears to be empty, represents the original model, which has 100% speedup compared to itself, of course.

Library options: The first four results (with the divorced models) were all achieved using Lauritzen and no noisy-MAX relevance reasoning. The best results for the non-divorced noisy models were achieved with RecDec and noisy-MAX relevance activated. The models that were neither noisy nor divorced performed best with Lauritzen again, with no noisy-MAX relevance, except for the original model that seems to have performed better with noisy-MAX relevance activated. The difference was minimal, since noisy-MAX relevance reasoning does not do much in a non-noisy model. That these tests happened to perform faster than the ones without noisy-MAX relevance reasoning might be due to measurement inaccuracies.

Remember this diagram compares the speedup between the best speeds each model has achieved, i.e. the given model with its fastest library options against the original model with its fastest library options. To calculate the overall speedup achieved with the testbench, one has to compare the best test options to the original model with default library options, which lies at 26.4475ms / 1.14082ms = 2318%.

Final conclusion: for the diagnostic use case, the "Network1/remArcs/margAuxNodes/divorce" model should be used, together with Lauritzen, and all relevance reasoning disabled.

**Illustration 17: Model highscore, third pass, diagnostic use case**

The final diagram to the diagnostic use case shows the best speedups each model has achieved, compared to the original model. Removing unnecessary arcs yields a 132% speedup already; however, to achieve a speedup of around 600% the network has to be divorcable. As can be seen on the top, the best model was the one with arcs removed, auxiliary nodes marginalized, parents divorced but non-noisy nodes, at 716% speedup (closely followed by the same without auxiliary nodes marginalized, at 700%). The relative speedups were calculated by dividing the modified models' best times through the best time achieved with the original model. "Best time" here means the average query time that was achieved with the respectively best combination of library options. The line 9 at the bottom, which appears to be empty, represents the original model, which has 100% speedup compared to itself, of course.

Library options: The first four results (with the divorced models) were all achieved using Lauritzen and no noisy-MAX relevance reasoning. The best results for the non-divorced noisy models were achieved with RecDec and noisy-MAX relevance activated. The models that were neither noisy nor divorced performed best with Lauritzen again, with no noisy-MAX relevance, except for the original model that seems to have performed better with noisy-MAX relevance activated. The difference was minimal, since noisy-MAX relevance reasoning does not do much in a non-noisy model. That these tests happened to perform faster than the ones without noisy-MAX relevance reasoning might be due to measurement inaccuracies.

Remember this diagram compares the speedup between the best speeds each model has achieved, i.e. the given model with its fastest library options against the original model with its fastest library options. To calculate the overall speedup achieved with the testbench, one has to compare the best test options to the original model with default library options, which lies at 26.4475ms / 1.14082ms = 2318%.

Final conclusion: for the diagnostic use case, the "Network1/remArcs/margAuxNodes/divorce" model should be used, together with Lauritzen, and all relevance reasoning disabled.
4.2 Usage profile: causal direction

4.2.1 First pass: filtering useful algorithms

The TestPlan for the first pass of the causal use case was similar to the one of the diagnostic use case, since the purpose of the first passes was to try out all possibilities and filter out those which are obviously useless:

- 8 random-generated queries that had evidence on all Hardware Component nodes (with maximum 3 fault states), and all DTC and Software Service nodes set as targets.
- The queries were executed using targets,
- on all 9 models,
- with all 13 inference algorithms available in SMILE,
- noisy-MAX relevance switched on and off, and
- evidence processing for explicit evidence (library default), or none.

Altogether $8 \times 9 \times 13 \times 2 \times 2 = 3744$ tests, resulting in 468 data points based on 8 tests each. 72 cases (576 tests) were skipped, leaving 3168 tests resulting in 396 data points.

Comparison of algorithms

Illustration 18: Comparison of algorithms, first pass, causal use case

As can be seen in Illustration 18, many more algorithms were useful this time. Pearl of course still did not work, and also LBP did not provide any results (it was later found out in correspondence with DSL that LBP is buggy and will not be included in future GeNIe&SMILE releases). AISSampling performs much worse than the rest and was therefore also discarded for the second pass. AIS means Adaptive Importance Sampling and uses a machine learning algorithm to learn the weights for samples up front. It was designed for extremely big Bayesian networks and has been superseded by EPIS (Evidence Pre-
propagation Importance Sampling) which does not need the machine learning step.

Conclusion: Sampling algorithms work better in causal direction; Pearl, LBP and AISampling were excluded for the second pass.
4.2.2 Second pass

Test Plan of the second pass:

– 128 random-generated queries that had evidence on all Hardware Component nodes (with maximum 3 fault states), and all DTC and Software Service nodes set as targets.
– The queries were executed using targets,
– on all 9 models,
– with 10 inference algorithms (all except Pearl, LBP, and AISSampling),
– noisy-MAX relevance switched on and off, and
– evidence processing for explicit evidence (library default), or none.

Altogether 128*1*9*10*2*2 = 46080 tests, resulting in 360 data points based on 128 tests each. 56 cases (7168 tests) were skipped, leaving 38912 tests resulting in 304 data points.

Comparison of algorithms

Illustration 19 shows that the remaining approximate algorithms perform poorer than the exact ones in all cases; but at least in a comparable range (one order of magnitude). So, also for the causal use case, trading accuracy against speedup is NOT an issue. There seem to be two groups of approximate algorithms, with SelfImportance worst, the others better, and Backsampling sometimes in one group and sometimes in the other. The speculation is that SelfImportance and Backsampling engage in some kind of elaborate computations that are of no use in the causal direction; however, the behavior was not further researched, as the main goal of the project was to draw conclusions about the cases that perform best.

Conclusion: Also for this use case, the exact inference algorithms are the ones that perform best.
Effects of relevance reasoning

Illustration 20: Effects of relevance reasoning, second pass, causal use case

Evidence processing still often spoils the performance, but not in all cases: it can be seen in Illustration 20 that for some cases, the blue line goes above the yellow line. These cases are using the approximate algorithms (except Backsampling) working on divorced models. However, in the fastest cases, seen on the left hand side of the diagram, switching evidence processing off is still crucial to query performance.

Relevance reasoning about noisy-MAX nodes does not make much difference in any of the cases, but it sometimes improves performance a bit when no evidence processing is used, and it sometimes decreases performance a bit when evidence processing is used. Of course, noisy-MAX relevance only helps pruning the network when evidence is set on the noisy-MAX child nodes, which does not happen in this use case.

Note that evidence processing is much more useful when inferring in causal direction: the chance roots of our case study are set to fixed values, and the rest of the network is deterministic which means that most nodes' values are already set by propagated evidence, leaving only a few nodes to compute through inference.

Conclusion: This diagram shows that relevance reasoning during evidence processing can actually increase the overall query performance in some cases. For the use case at hand, this phenomenon is not relevant though.
Comparison of model modifications

Unlike in the diagnostic use case, here no clear division between the models can be found. It should be noted that the original model is not the worst in all cases; for some library options, some model modifications can actually make things worse. On the other hand, it can be seen on the left hand side of the diagram that, for some library options, all models perform well.

Conclusion: For the fastest cases (which were in the focus of the research), model modification does not play a big role when inferring in causal direction.

Illustration 21: Comparison of model modifications, second pass, causal use case

The points are connected by lines to make it easier to see which points belong together.
Speedup by QI-aware model modification

Illustration 22: Speedup by QI-aware model modification, second pass, causal use case

Illustration 22 shows the increase in performance gained through marginalization of certain auxiliary nodes, a model modification that makes use of the query interface. Regarding the conclusion of the previous Illustration 21, stating that model modifications do not play a big role for the use case at hand, the above Illustration 22 provides surprisingly good results.

First, we note that the speedup is consistently positive in all cases: across all algorithms, settings for relevance reasoning, and other model modifications, the model modified with margAuxNodes performs better than the one without. This was not the case in the diagnostic use case. Second, the highest achieved speedup lies at almost 109%, which is higher than the 104% achieved in the diagnostic use case. Still, 109% in the best case is just not enough to be called statistically significant. The speedups achieved in the cases with the fastest queries are around 105-106%.

The highest relative speedups through marginalizing auxiliary nodes are achieved in divorced models, with either exact algorithms, or approximate algorithms not using evidence processing. Among those benefiting the least from the modification are approximate algorithms running on undivorced models using evidence preprocessing.

Our interpretation: Evidence processing takes up a substantial part of the overall query time, giving performance benefits to the inference part. Therefore, in cases where evidence processing is used, the speedup by margAuxNodes depends more on its influence on the evidence processing, and in cases where it is not used, the speedup by margAuxNodes depends more on its influence on inference. This is supported by the observation that, where EP was used, the absolute speedup is about the same across all algorithms (which result in very different relative speedups however), while there are visible differences in absolute speedup between different algorithms in cases where EP was not used (which result in similar relative speedups). In other words: exact algorithms run faster and therefore profit more from consistent absolute speedups in EP, while approximate algorithms, which run slower, profit more from relative speedups that come from reducing the complexity of the model for inference.

Conclusion: margAuxNodes provides better results in causal direction, but falls short of proving that QI-aware model modification can yield statistically significant speedups.
The best achieved query times, around 2 ms, are about double as big as in the diagnostic use case. This may be the case because the queries as specified in the causal usage profile are bigger (more evidence, more target nodes). The curve has a very obvious jump where query time doubles within just 3 configurations. The fastest ones are using exact algorithms without evidence processing (the model used does not make much difference), then after the first small jump over the 5.0 ms come divorced models with exact algorithms using evidence processing. After the big jump it continues with undivorced models, and then more and more approximate algorithms appear. The last third of the curve consists of approximate algorithms only.
4.2.3 Third pass

The third pass was conducted with exact algorithms, not using evidence processing. TestPlan of the third pass:

- 1024 random-generated queries that had evidence on all Hardware Component nodes (with maximum 3 fault states), and all DTC and Software Service nodes set as targets.
- The queries were executed using targets,
- on all 9 models,
- with the 4 exact inference algorithms (Lauritzen, LauritzenOld, LinDec and RecDec),
- noisy-MAX relevance switched on and off, and
- relevance reasoning during evidence processing switched off.

Altogether 1024*1*9*4*2*1 = 73728 tests, resulting in 72 data points based on 1024 tests each. 4 cases (4096 tests) were skipped, leaving 69632 tests resulting in 68 data points.

**Query steps / API calls in detail**

Illustration 24 shows that almost all query time is taken by the actual inference. Evidence processing runs really fast, even though evidence propagation sets most of the nodes.

Note that this diagram is not directly comparable to Illustration 15, which is produced from the diagnostic use case's second pass, not its third pass.

The analysis of the fastest configurations shows another jump between the faster Lauritzen and RecDec algorithms and the slower LauritzenOld and LinDec algorithms. Within each of these two groups, the models with marginalized auxiliary nodes perform consistently better (albeit without any visible jump).
More stats of the query times

Illustration 25 displays the same curve as the previous chart, this time enriched with query stats like minimum and maximum query times instead of the average fractions of query time used by the different query steps. The data contained two outliers of the maximum query time, which lay around 178 ms and were shortened to 40 ms for viewability purposes. These cases were using undivorced noisy models, running Lauritzen without any relevance reasoning.

The red line indicates query stability (small values indicate that all queries took about the same time, while bigger values mean that query time varied more for different queries). Obviously, the performance of Lauritzen and RecDec on divorced models is largely immune to changes in the queries’ evidence. This raised the question whether a large part of inference time in these cases might be used up for the clustering process, which would not only take about the same time for all queries but might indeed yield the same join trees. In this use case, the evidence is different, but the evidence nodes stay the same (evidence is always provided on all Hardware Component nodes). If this is true, then average query time for this special use case could be further (and drastically) reduced by keeping the clustering in cache in between queries. This is however not accessible in the SMILE library.

Conclusion: As in the diagnostic direction, in the causal use case query stability is not an issue for the fastest query configurations.

Illustration 25: Ranking with query stats, third pass, causal use case
Illustration 26: Model highscore, third pass, causal use case

Illustration 26 is the final diagram for the causal use case. It compares the different models by the best query times that were achieved for each of them. Unlike in the diagnostic use case, the good models are only the actually divorced ones. Being "divorcable by relevance reasoning" does not work, because evidence is never set on the noisy child nodes, and noisy pruning can only yield benefits when this is the case. The best model is the one which had its unnecessary arcs removed, auxiliary nodes marginalized, and which has been divorced (converting the noisy nodes back to CPT after divorcing). This model achieved a speedup of 178% compared to the original model. It is closely followed by the same model with noisy-MAX nodes, at 177% speedup.

Library options: All results of the divorced models (places 1-4) were achieved with Lauritzen, no relevance reasoning enabled. The same holds for the non-noisy, undivorced models (places 5 and 6). The models on the places 7 through 9 ran fastest with Lauritzen together with noisy pruning.

These results have to be compared with those of the diagnostic use case, where speedups of over 700% were achieved, and where converting undivorced models to noisy-MAX provided a benefit instead of a disadvantage.

The overall speedup achieved with the testbench methodology (best test options compared to original model and default library options) lies at 17.5356ms / 2.14797ms = 816%.

Final conclusion for the causal use case: For this use case, Scania's troubleshooter software should use the model "Network1/remArcs/margAuxNodes/divorce”, together with Lauritzen, all relevance reasoning disabled, and using targets. Note that this turns out to be exactly the same recommendation as for the diagnostic use case, despite all observed differences in the behaviors of these two use cases.
5 Discussion

Part 5 will provide more abstract and in-depth discussion of the results and their meaning to Bayesian queries in general practice, as well as summaries and conclusions.
5.1 General observations

5.1.1 The stabilizing effect of divorcing on diagnostic queries

Divorcing provides excellent results especially for the diagnostic queries. Library options do not matter much – you cannot really do much wrong with a divorced model. After excluding obviously not working settings in a first pass of tests, the worst possible configuration for a divorced model takes just about three times the time of the best configuration (see Illustration 11 on page 41). A divorced structure seems to make both evidence processing and inference more efficient.

The initial purpose of divorcing was to obtain smaller cliques in the clustering algorithms. It is striking, however, that we get a very similar speedup in the undivorced noisy models if used with appropriate library options. The undivorced noisy models (those containing big noisy-MAX nodes) do not yield speedup because of the smaller node size; noisy nodes in SMILE get CPT-instantiated before clustering. They yield speedup because they model more independencies between variables, which facilitates relevance reasoning (both for evidence processing and for model simplification during belief update). These very similar speedups suggest the conclusion that the main improvement of divorcing actually stems from the fact that relevance reasoning can work more fine-grained. A divorced network structure implicitly encodes independencies between the variables, and it runs fast because it facilitates relevance reasoning.

The other theory is that relevance reasoning in the undivorced noisy models produced small join trees because so large parts of the network could be pruned, and the very similar speedup in comparison to the divorced models was mere chance. This question is difficult to answer without a deeper analysis of the produced join trees, which are however hidden within SMILE.

As we came to see, all these observations do not hold for causal queries. In the causal use case, good results as well as bad ones can be achieved with any of the models; there is no silver bullet in terms of model modification (see Illustration 21 on page 52).

It should also be remembered that divorcing worked especially well in the model that was given in our case study, because all the nodes (with more than two parents) were perfectly divorcable without loss of accuracy, and some had a rather large number of parents. In Bayesian networks where this is not the case, the results will be not as good as here. Networks without big families will profit less from the modification, and for networks with CPT definitions less suitable for noisy-MAX conversion, there will be a tradeoff between the speed and the accuracy of the queries. As Zagorecki and Druzdzel [20] suggest, about half of the nodes in practical networks have approximately an underlying noisy-MAX distribution, which is the basis for our divorce implementation. This number might be raised further when research is conducted on more flexible forms of divorcing (see chapter 5.3 Issues and future work).

5.1.2 Other preprocessing

Removing arcs runs very quickly, plus it addresses trivial redundancy, and therefore this modification was included in all modified models. It yields considerable improvement in the given model. It is not expected to yield improvements in most models, but still, addressing trivial redundancy should be a basic step in any form of model optimization.

Noisy-MAX conversion adds information about independencies to the model, which enables more fine-grained relevance reasoning. Therefore it is very important that such a model is used with the right library options. Noisy-MAX conversion furthermore serves for knowledge elicitation for better human understanding of the model (e.g. looking at a giant truth table, it is not immediately obvious if the node represents an OR relationship), but that is not the main point of model optimization here. Once the noisy-MAX nodes are divorced, however, their amechanistic ICI property does not play a big role anymore. This can be seen by the fact that the non-noisy divorced models performed even a bit better than the noisy divorced ones, even though they theoretically contain less information about
independencies – this benefit was lost by the time taken for the CPT instantiation.

**Interface-specific preprocessing**

Marginalizing auxiliary nodes performed not so impressively. Not many nodes in the network were actually removed by the modification, and they were no big nodes, so there was not much effect on either sampling or clustering algorithms. Encouraged by a speedup of 109% achieved in one case, it seems possible to construct a situation where margAuxNodes does yield a statistically significant speedup; moreover, other QI-aware model modifications might be added to the palette in the future, which might further increase the usefulness of the concept of query interfaces to model optimization. Proving this point would have been a nice additional success for this thesis.

On the other hand, interface-specific preprocessing may not have a big potential in general. The point is that relevance reasoning can prune large unused parts of a network fast enough to be done for each query, and this also includes reasoning about the actual evidence provided. In other words, the query-specific relevance reasoning algorithms might already be too good to achieve much speedup by interface-specific model optimization.

### 5.1.3 The importance of relevance reasoning

Based on the observations described in 5.1.1, our impression is that relevance-based pruning is the actual key to optimize queries. This means that, when optimizing models, the most important thing to look at is making the model suitable for relevance reasoning. This can largely be achieved by divorcing.

Relevance reasoning during setting of evidence, however, is in the most cases counter-productive for queries, because it does a lot of redundant computation.
5.2 Applicability to other domains

The modifications of the SCR model can be expected to yield similar results for other BN of Scania that are script-generated from system development assets for troubleshooting. It is also relevant for other modeling approaches that are prone to producing models with suboptimal structures, i.e. models that are programatically generated from similar data. The least improvement can be expected in completely random-generated models, and in models that are structure-learned with an algorithm that supports recognition of ICI relations. Structure-learning automatically tries to create "minimal" models, and so these models already have the structure that contains the most information about independencies, and are unlikely to contain redundant information. Random-generated models are unlikely to benefit from this methodology because random-generated CPTs are unlikely to have a noisy distribution.

In terms of the library usage, the methodology could be made more applicable by supporting more libraries in addition to SMILE, both to handle more file formats and to make use of tweaks that are implemented in other libraries. Apart from this limitation, it provides a basic framework for testing the effects of different library options that should be applicable to any model and any use case.
5.3 Issues and future work

The testbench configures, runs and analyzes tests pretty much automatically. What still needs to be done manually is the test planning (implementing the query generation, deciding the test options), the evaluation of the result tables (for example by creating charts), and some manual aid that is necessary when marginalizing auxiliary nodes.

The implemented model modifications are far from perfect. The conversion to noisy-MAX and divorcing functions were written for the case study model at hand, but will perform poorly on networks with more complex nodes than logical OR. Also, only a single QI-aware modification was examined, and it yields only insignificant speedup.

Questions for further research:

– How does the current methodology actually perform with other models, like (a) similar models for troubleshooting, generated with a script from system development assets, (b) common models like Hailfinder etc, (c) structure-learned models, (d) completely random-generated models?

– How much more would these models be improved with a proper method of partial noisy divorcing? And can there be an even more flexible clique size vs. accuracy loss tradeoff if the method allowed a flexible number of states in the helper nodes? (Partial noisy divorcing was described in chapter 3.1.2 Conversion to noisy-MAX definitions, page 27.)

– Any CPT node can be represented as a structure of noisy-MAX nodes with the help of a disjunctive normal form. This, however, can produce an exponential number of nodes. Does this yield any benefit through more fine-grained relevance reasoning and smaller clique sizes, or does performance decrease too drastically because of the number of nodes introduced to the network?

– How do other libraries (free or commercial) perform in comparison to SMILE?

– How much relevance reasoning (and other kinds of query preparation) could be performed with only the knowledge of the QI? In use cases where queries are to be carried out on a small part of a very large network, large parts of a network could be pruned simply by knowing that these parts will always be barren or nuisance nodes, which would otherwise be performed for each query.

– What benefit can be expected from caching? This is highly dependent on the application, because the application has to post the same queries several times, but this can be included in the verbal usage profile description and implemented in the TestPlan. Caching could be implemented as a layer between the library and the surrounding application, and does not necessarily need to know anything about either of them. This layer was not included in the scope of this thesis project, but could be evaluated with the same methodology.

– How can a joint posterior probability distribution be computed in the most efficient way, using SMILE or other libraries? This is not part of the concept of queries in this thesis, but that concept could well be extended. The presented troubleshooter application will need to calculate joint posteriors. See a brainstorming list of approaches in the appendix.

– Can constraint solving be incorporated into Bayesian inference (or dynamically replace it), in order to reason faster about deterministic nodes? What speedup would be gained in Scania’s troubleshooter? Significant speedup is to be expected especially for the causal use case, because the remaining network is fully deterministic.

– Regarding the open question in 5.1.1: How did the produced join trees differ between noisy and non-noisy, divorced and undivorced models? How can they explain the similar speedups achieved by two very different approaches? Is it true that a divorced network structure performs well because it facilitates relevance reasoning?
– Evaluate recording of queries. This means that, instead of intelligently guessing what kind of queries will be posed to the network by the outer application, the application is run instead, and a layer between application and Bayesian network records all queries that occur. The recorded queries can be used in the TestPlan. The usage profile, then, has to specify what a user typically does with the application, which would be on a more abstract and usage-oriented level than specifying the queries directly.

– In cases where the usage profiles of an application are not obvious, one could go a different route: Determine for each query separately which library options and model modifications it performs well with. Then, try to find groups of queries that perform well with the same settings, and try to find commonalities between these queries. The purpose is to find simple criteria by which the application can decide for each new query what settings to use.

The last two points are interesting because they can further reduce the amount of human work necessary to optimize queries. In the case study of this thesis, both the diagnostic and the causal use case resulted in the same usage recommendation. These recommendations however are based on the average of several queries. It might well be that certain queries perform better with different settings, and it would be interesting to understand what makes the difference. Also, of course, it would increase the overall performance of the application.
5.4 Conclusion

The performance behavior of a Bayesian network is very much dependent on the use case, that is, what types of queries are going to be posed to the model.

Several automatic model modifications, along with several options available in the SMILE library, have been evaluated for their impact on query performance. Using targets has been found to be universally positive; the other library options can develop a complex interaction with each other and with the used model modifications, which result in different query performance. For the case study, and probably for a large class of Bayesian networks, divorcing the model, using a clustering algorithm, and switching off relevance reasoning during evidence processing proved to be good choices to optimize query performance. Good speedups were measured.

With the help of the testbench framework, all that needs to be done is to formalize the verbal description of a usage profile into a TestPlan, and evaluating the test results will tell you which library options and model modifications result in what speedup, accuracy loss, and query stability. Trading off these goals against each other will decide which settings the application should use. Using the three-passes-method is recommended, as it saves time during testing by not running too many tests with settings that are slow and therefore useless.

For Scania

Our recommendation is to re-write the script that creates the Bayesian networks, so that it automatically creates noisy-MAX nodes from the Boolean gates in the error propagation tree. If there are more complex Boolean functions in the error propagation tree than mere AND or OR gates, create a structure of several noisy-MAX nodes; generating truth tables from these functions would need partial noisy divorcing to recover all independencies. In the resulting model, routinely perform automatic removal of unnecessary arcs. This procedure will keep the runtime of the script very small, and will produce a model that is very easy to understand for humans and very easy to divorce by the testbench to produce models that are more query-friendly.

For the SCR case study at hand, as mentioned before, the troubleshooter should use the model "Network1/remArcs/margAuxNodes/divorce", together with Lauritzen, all relevance reasoning disabled, and using targets, for both the diagnostic and the causal use case.

Another issue for the troubleshooter application is that the planner works with joint fault probabilities, so it needs to be investigated either whether that can be changed, or how to compute joint posteriors with SMILE in the most efficient way. Also caching of query results might be worth further investigation.
5.5 Summary

As a basis for our work with model modifications, a testbench software was built that allows us to run configurable queries on Bayesian networks and compare their performances for different settings of the SMILE library (the inference algorithm and flags for relevance reasoning), and for different models.

We implemented the automatic model modifications of divorcing, conversion to noisy definitions, marginalizing auxiliary nodes, and removing unnecessary arcs. These modifications were carried out on the model of our case study, to see what effects they have on query performance.

We developed a general methodology for optimizing queries in Bayesian networks. The methodology consists of formulating a verbal description of the outer application's usage profile, determining its query interface, composing the test plan (determine which model modifications are to be tested, and implement a random query generation), evaluating the resulting rankings and comparison tables, and, based on a tradeoff between query time, accuracy, and stability, deciding which options to use in the end product. Our method included three test passes with increasing focus on the most promising options.

Our results suggest that the most important issue to consider for optimization of models and queries is relevance reasoning, the ability of the used library to prune the model from irrelevant nodes in order to avoid redundant computations. This is greatly facilitated by a divorced network structure, which also improves performance because it leads to small join trees. Relevance reasoning during evidence processing, on the other hand, has been found to be largely counter-productive.
6 Miscellaneous

Thanks

Special thanks go to:

– my supervisor Jose M. Peña for actively supporting the project with valuable input and tireless help during correction,

– the DSL team at Pittsburgh University for responsive user support, inspiring discussions and insights about the inner workings of SMILE,

  – Marek J. Druzdzel in particular for a GeNIe t-shirt,

– my examiner Fang Wei-Kleiner,

– my opponent Jay Zimmermann,


Download

The fruits of this work can be downloaded at:

http://www.lysator.liu.se/~jonnie/thesis/

These include the thesis, copies of related literature, the test results of the case study, and of course also the Visual C++ project of the testbench framework (without the Scania-specific code).

Disclaimer

The core of my implementation is based on the SMILE reasoning engine for graphical probabilistic model contributed to the community by the Decision Systems Laboratory, University of Pittsburgh and available at http://genie.sis.pitt.edu/.
Appendix
### 7.1 Acronyms

A list of acronyms used in the thesis, with explanations.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN</td>
<td>Bayesian Network</td>
</tr>
<tr>
<td>DBN</td>
<td>Dynamic Bayesian Network (modeling time slices)</td>
</tr>
<tr>
<td>CPD</td>
<td>Conditional probability distribution</td>
</tr>
<tr>
<td>CPT</td>
<td>Conditional probability table</td>
</tr>
<tr>
<td>JPD</td>
<td>Joint probability distribution</td>
</tr>
<tr>
<td>JPT</td>
<td>Joint probability table</td>
</tr>
<tr>
<td>ICI</td>
<td>Independence of Causal Influence (for example, noisy-MAX)</td>
</tr>
<tr>
<td>SCR</td>
<td>Selective Catalytic Reduction (case study truck system)</td>
</tr>
<tr>
<td>DTC</td>
<td>Diagnostic Trouble Code (also used for a node type)</td>
</tr>
<tr>
<td>HWC</td>
<td>Hardware Component (node type)</td>
</tr>
<tr>
<td>HWS</td>
<td>Hardware Service (node type)</td>
</tr>
<tr>
<td>SWS</td>
<td>Software Service (node type)</td>
</tr>
<tr>
<td>LBP</td>
<td>Loopy Belief Propagation (approx. BN inference algorithm)</td>
</tr>
<tr>
<td>BN2O</td>
<td>Bayesian Network, 2-layer noisy-OR (subclass of BN)</td>
</tr>
<tr>
<td>SMILE</td>
<td>Structured Modeling, Inference, and Learning Engine (a C++ library for BN)</td>
</tr>
<tr>
<td>DSL</td>
<td>Decision Systems Laboratory (at the University of Pittsburgh)</td>
</tr>
<tr>
<td>DAG</td>
<td>Directed Acyclic Graph</td>
</tr>
</tbody>
</table>
7.2 Computing joint posteriors with SMILE

This is a brainstorming list of approaches to compute joint posterior probability distributions with standard Bayesian network software such as SMILE.

Standard BN software such as SMILE feature the direct calculation of marginal (separate) posteriors of nodes, but not joint (combined) posteriors of several nodes. Building a joint posterior by multiplying the probabilities of each combination does not work (it works if the variables are independent, but this depends on the graph structure and the evidence present).

7.2.1 Three methods

Disclaimer: I have NOT tried out these three methods in practice to compare their performance and feasibility. Which method is best depends on your model and your usage pattern.

**Probability of evidence**

One way to calculate a joint posterior with a tool is by using the probability of evidence. First, you calculate \( P(\mathcal{E}) \). Then, to calculate the probability of a single combination in \( JP(T|\mathcal{E}) \), you instantiate the wanted combination in \( T \) to \( \mathcal{T} \) and calculate \( P_{\mathcal{T}} = P(\mathcal{T}|\mathcal{E}) \), the probability of evidence for the observed evidence and the instantiated target combination at once. Then, \( P_T = P_{\mathcal{T}} / P(\mathcal{E}) \) is the joint posterior probability of \( T \) given \( \mathcal{E} \), or \( P(T|\mathcal{E}) \). The calculation of \( P_{\mathcal{T}} \) has to be done once, and \( P_{\mathcal{T}} \) for every state combination you’re interested in. If you want to compute all values of \( JP(T|\mathcal{E}) \), this number will grow combinatorically.

**Sequential instantiation**

A second way to calculate a joint posterior \( JP(T|\mathcal{E}) \) is by sequentially instantiating the target nodes. Starting with the first target node \( t_1 \in T \), you first calculate the probability of its wanted state \( t_1.s \) by using \( P_{t_1} = P(t_1|\mathcal{E}) \). Then, \( t_1 \) gets instantiated to \( t_1.s \), and added to the network's evidence. Then you continue with the next target node \( t_2 \), calculating \( P_{t_2} = P(t_2|\mathcal{E}, t_1) \). Along the way, you multiply the \( P_{t_i} \), and in the end you get \( P_T = P_{t_1} * P_{t_2} * P_{t_3} ... * P_{t_n} \). This method requires \( n \) steps of inference for each state combination you’re interested in.

**Join nodes**

A third way to calculate the joint posterior \( JP(T|\mathcal{E}) \) is by using a join node. You add a node to the network that represents the joint probability distribution of \( T \). The join node \( n_j \) has all the nodes \( t_i \in T \) as parents. The node has one state for each state combination of its parents (a number growing combinatorially), and has a deterministic CPD which basically says that for each combination of the parents’ states, the corresponding join state in \( n_j \) is selected. This method will compute all values in the joint posterior probability distribution with a single pass of inference by querying \( P_{n_j} = P(n_j|\mathcal{E}) \). However, this inference pass needs exponential time just like the other methods, and also exponential RAM space, which will most likely make it infeasible for more than a handful of target nodes.

7.2.2 Optimization

It is also a good question how one or the other method may be optimized. In all cases, it might be interesting to apply relevance reasoning, to see whether the big mass of target nodes \( T \) can be split into independent groups. Only dependent nodes need to be computed by the methods given above, and the resulting partial joint posteriors can be combined by simple multiplication.

Also, when doing sequential instantiation, one should consider the possibility of calculating several probability values at once when we already instantiated half of the nodes along the way. If you want to compute the probabilities of combinations with all states of some nodes and only specific states of other nodes, you can put the latter to the front of the "ordered target queue".
Also for the join node method, you can restrict yourself from calculating the values of all state combinations to only specific combinations. For this, simply use a "other uninteresting combinations" state in nj, and let the CPD select it in all those cases where the parents have an uninteresting state combination. It might also be interesting to see whether divorcing can do any good. It probably will not do any good if you want to calculate the probabilities of all state combinations, because then the number of states in the divorce helper nodes will be exponential just as well. But even in this case, consider that divorcing (and noisy-MAX definitions) can always yield performance improvements through relevance reasoning.

7.3 Lookup tables

Illustrations
Illustration 2: Relevance example.................................................................8
Illustration 3: Example of a node with a noisy-MAX definition...............................9
Illustration 4: The Bayesian network model of the SCR system used in the case study.............16
Illustration 5: Example of divorcing a noisy-MAX node....................................26
Illustration 6: A CPT representing the Boolean formula A or (B and C).......................27
Illustration 7: Comparison of algorithms, first pass, diagnostic use case....................37
Illustration 8: Benefit of using targets, first pass, diagnostic use case.......................38
Illustration 9: Comparison of algorithms, second pass, diagnostic use case..................39
Illustration 10: Effects of relevance reasoning, second pass, diagnostic use case............40
Illustration 11: Comparison of model modifications, second pass, diagnostic use case........41
Illustration 12: Speedup by QI-aware model modification, second pass, diagnostic use case.42
Illustration 13: Ranking, second pass, diagnostic use case...................................43
Illustration 14: Ranking with query stats, second pass, diagnostic use case..................44
Illustration 15: Ranking with query steps, second pass, diagnostic use case...................45
Illustration 16: Comparison of algorithms, third pass, diagnostic use case...................46
Illustration 17: Model highscore, third pass, diagnostic use case.............................47
Illustration 18: Comparison of algorithms, first pass, causal use case........................48
Illustration 19: Comparison of algorithms, second pass, causal use case.....................49
Illustration 20: Effects of relevance reasoning, second pass, causal use case................50
Illustration 21: Comparison of model modifications, second pass, causal use case...........51
Illustration 22: Speedup by QI-aware model modification, second pass, causal use case....52
Illustration 23: Ranking, second pass, causal use case.......................................53
Illustration 24: Ranking with query steps, third pass, causal use case.........................54
Illustration 25: Ranking with query stats, third pass, causal use case........................55
Illustration 26: Model highscore, third pass, causal use case....................................56

Formulas
Bayes' theorem (1).........................................................................................7
Bayes' theorem (2).........................................................................................7
Diagnostic queries (3)....................................................................................17
Causal queries (4)..........................................................................................17
Conditional probability (5).............................................................................18
PND (6)..........................................................................................................27
PND (7)..........................................................................................................27
PND (8)..........................................................................................................27
Equivalence (9)...............................................................................................29
Interface equivalence (10)...............................................................................29
Sum of family sizes (11)................................................................................30
Speedup (12)....................................................................................................42
7.4 Literature review

This chapter reviews the papers and other literature referenced throughout this report, as well as additional literature for the interested reader, providing a short comment about the contents relevant to the thesis, and a URL where available. The formal reference list is provided in chapter 7.5 References.

  Served as the standard reference for all basic concepts within Bayesian networks during the work on this thesis project.
  Homepage of the book: http://bndg.cs.aau.dk/

7.4.1 On Scania’s troubleshooter

These papers and dissertations originate as a result of the cooperation between Linköping University and Scania AB on the topic of computer-aided troubleshooting.

- [2] Probabilistic Fault Diagnosis with Automotive Applications:
  PhD thesis on the diagnoser part of the troubleshooter application.
  https://www.fs.isy.liu.se/Publications/PhD/09_PhD_1288_AP.pdf

- [3] Computer-Assisted Troubleshooting for Efficient Off-board Diagnosis:
  PhD thesis on the planner part of the troubleshooter application.
  http://liu.diva-portal.org/smash/get/diva2:411037/FULLTEXT01

- [4] Planning as Heuristic Search for Incremental Fault Diagnosis and Repair:
  Paper on the planner part of the troubleshooter application.

  Paper about software services and error propagation graphs as used by Scania AB, and the procedure of building Bayesian networks from these assets.
  http://www.fs.isy.liu.se/Publications/Articles/DX_10_MN_CS.pdf

- [6] Design Patterns for Service-Based Fault Tolerant Mechatronic Systems:
  Master's thesis about software services and error propagation graphs as used by Scania AB, and the procedure of building Bayesian networks from these assets.
  http://liu.diva-portal.org/smash/get/diva2:441919/FULLTEXT01

7.4.2 On Bayesian inference

- [21] Efficient Inference in Bayes Networks as a Combinatorial Optimization Problem:
  This paper regards Bayesian inference as a factorization problem, and different (exact) algorithms as different factorization strategies.

- [22] Inference in Bayesian Networks, 1999:
  This paper gives an overview of different inference algorithms (exact as well as approximate) and the ideas and mathematical theory behind them.
  http://sclab.yonsei.ac.kr/courses/06mobile/6-2.pdf

- [7] A simple approach to Bayesian network computations:
  Paper on Variable Elimination.

- [10] A Tractable Inference Algorithm for Diagnosing Multiple Diseases:
  Paper on using ICI gates in inference.

- [23] On the impact of causal independence: Paper on using ICI gates to reduce the complexity of inference (and other calculations such as finding the most probable explanation).

- [24] Efficient computation for the noisy MAX: Paper on using ICI gates to reduce the complexity of inference, and different types of join trees and message passing.
  http://www.ia.uned.es/~seve/publications/MAX.pdf

  http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.4.9917&rep=rep1&type=pdf


- [12] Approximating probabilistic inference in Bayesian belief networks is NP-hard: This paper contains a proof that also Bayesian inference with stochastic sampling algorithms is NP-hard.

- [27] A Factorized Representation of Independence of Causal Influence and Lazy Propagation: Paper on using ICI gates to reduce the complexity of inference; based on a type of join tree that is different than that used in SMILE.

7.4.3 Papers describing the inference algorithms in SMILE

These are the papers corresponding to the inference algorithms in SMILE, as described in chapter 2.3.1 SMILE in-depth.

Source: http://genie.sis.pitt.edu/wiki/Reference_Manual:_DSL_network, the online manual for SMILE maintained by the Decision Systems Laboratory at the University of Pittsburgh.

  http://staff.aub.edu.lb/~darwiche/Papers/ijar95.pdf


  http://www.ece.tamu.edu/~bjyoon/ecen689-604-fall10/Pearl_1986.pdf

  http://uai.sis.pitt.edu/papers/89/p112-fung.pdf


- [16] Backward Simulation in Bayesian Networks:
Paper on Backsampling.

Paper on AISSampling.

– [18] An Importance Sampling Algorithm Based on Evidence Pre-propagation:
Paper on EpisSampling.

Paper on LBP.

– [19] Computational Advantages of Relevance Reasoning in Bayesian Belief Networks:
Paper on RelevanceRngLinDec and RelevanceRngRecDec.

### 7.4.4 On preprocessing

Collected papers about parent divorcing and/or conversion to ICI gates.

Paper about the CPT to noisy-MAX conversion algorithm, a linear gradient algorithm which finds the noisy-MAX parameters that provide the best fit to the original CPT.
http://www.pitt.edu/~druzdzel/psfiles/ecai06.pdf

– [28] Exploiting Functional Dependence in Bayesian Network Inference:
This paper describes a promising approach to a more flexible way of divorcing CPT nodes into binary trees of ICI gates.
http://www.mathcs.emory.edu/~whalen/Papers/BNs/DistributedBNs/CausalDecomposition/ExploitingFunctionalDependenceinBayesianNetworkInference.pdf

– [29] Causal Independence for Knowledge Acquisition and Inference:
Paper on sequential divorcing (and its benefits during model creation).

– [30] A MUNIN network for the median nerve - A case study on loops:
This paper is cited as the standard description of divorcing in Bayesian networks. Not freely available online.
7.5 References


Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

© Johannes Förstner

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Johannes Förstner