TNO report

TNO 2023 R10096| 1.0

Guided root cause analysis of machine failures - Status 2022

Date	10 January 2023
Author(s)	Emile van Gerwen
Copy no	
No. of copies	
Number of pages	82 (incl. appendices)
Number of appendices	
Sponsor	Canon Production Printing APPL.AI TNO program
Project name	Carefree
Project number	060.51615

Summary

Today's complexity of high-tech systems makes diagnosing system failures a tough task for service engineers. Increasing product variability and fast market introduction of new generation systems prohibit the expertise build-up that served service engineers in the past. Traditionally, system knowledge is transferred to the service organization through service manuals and training. This turns out to be inadequate in the complex world with customers expecting high system availability.

Our goal is to transfer design knowledge to the service organization in the form of computational models such that the service engineer has an actionable tool to assist them in their diagnostic task.

Major part of our research is to create these computational models in a structured, scalable, and maintainable way that fits into the system development way of working.

The basic idea is to

1. Define input/output behavior every component type used in the system, both its normal behavior as well as for every failure mode. A prior probability for every failure mode needs to be established.
2. Compose a system model with the component descriptions as building blocks following the physical or functional structure of the system.
3. Define a set of tests (observations, measurements, service actions) that can help in diagnosis.

Based on these description, we automatically create a computational diagnostic model that can

1. List the most suspected components that have failed, including the uncertainty associated with these hypotheses.
2. List the best tests that will increase the accuracy of the diagnosis the most, i.e. reduce the uncertainty the most at the least cost and effort.

Iteratively feeding the test results into the model then iteratively leads to a improved diagnosis until a service action is appropriate.

Next to the model, we developed a prototype service engineer oriented user interface to convey the ideas and way of working.

This idea sounds easy and is certainly not new, but to apply this methodology in practice has many pitfalls. Next to outlining the methodology in detail, in this report we also describe our solutions to the stumbling blocks we came across while applying the methodology on an industrial printer use case.

Contents

0 Preamble

0.1 This report in the grand scheme of things

This report describes the status of the Carefree project on diagnostics. The Carefree project is a collaboration between Canon Production Printing and TNO-ESI, as a use case within the Appl.AI research program at TNO.

End 2021 we published a similar report but as that was Canon confidential [1], we decided to copy large parts of it verbatim, leaving out sensitive information, into this report and add the 2022 progress where applicable. This way the reader should be able to understanding the full picture on assisted root cause analysis without having to refer to earlier work.

Although extensive, this report does not cover all the work done in 2022. The work on performance degradation and failure prediction, as opposed to root cause analysis of system failures, is document in two other reports: [2] for Canon internal use and [3] publicly available. The work on discovering relationships between printer usage and system errors from available machine data is documented in [4].

0.2 Why to read this document, or why not

This report tries to be self-contained in that it assumes little prior knowledge. It discusses the technology, the approach, and implementation details. As such it is quite a lengthy document. For more concise descriptions of the approach we kindly refer to general publications such as [5] and [6].

0.3 What's new?

Readers familiar with the 2021 report and interested in the 2022 specific progress should focus on the following sections:

- **Section 5.5.2 Multiple properties**, taking advantage of the multiple-variables-per-node feature of the Bayes Belief Network tool for generating compact networks.
- **Section 5.5.3 Property dependencies / Constraints**, an extension to the specification language that allows to define constraints on properties. This improves the diagnostics because physically impossible situations are removed from the hypothesis set.
- **Section 5.6 Loops**, where we implemented the loop handling approach following up on earlier solution explorations.
- **Section 5.7 Upstream effects**, yet another way of dealing with upstream effects such as a short-to-ground in an electrical circuit. The new approach is more compositional.
- **Section 5.10 and 6.8 Interventions**. Following up on earlier investigations, we implemented a solution that deals with interventions, i.e. actions that

change the system in order to get more information and ultimately fix the error.

- **Section 5.11 Diagnosing wrong inputs.** A “failure” to get a desired result is not necessarily caused by a component failure. It could be that an input is different from the expected value. In this section we address diagnosing wrong inputs.
- **Section 6.5.3 Decision networks**, a research line in cooperation with our academic partners in which we investigate using decision networks to calculate the optimal test to do.

1 Setting the stage

1.1 Background

This technical report describes the state as of the end 2022 of the "Carefree" project, cooperation between Canon CPP and TNO-ESI. The purpose of the project is to improve the service process so that

1. Service costs are reduced
2. System down time is reduced

The starting point is an existing system and the goal is to improve the service process. Note that these objectives could also be obtained by increasing the serviceability and/or diagnosability of the printer at design and manufacturing time, but that is not the focus of this study.

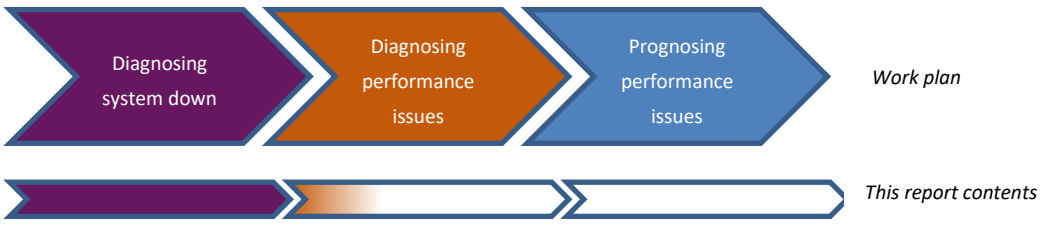
1.2 Scope

1.2.1 Corrective, preventive, and predictive maintenance

Maintenance can be either corrective or preventive. Corrective maintenance is the action to repair a broken or badly performing system while preventive maintenance is taking service actions when the system is still functional. Preventive maintenance has the big advantage that it can be planned. This is beneficial for Canon but mostly for the customer because the *unexpected* system down time will be lower. In preventive maintenance the maintenance interval is an important steering parameter. When too small, maintenance costs and system down time are too large, and for a too large interval the system will still suffer from unscheduled down times and high repair costs. Typically the maintenance interval is determined on time or usage indicators such as number of products produced. When more sophisticated measurements are used, such as vibration levels on bearings, preventive maintenance is usually called predictive maintenance.

In this work we start with improving the corrective maintenance and extend towards predictive maintenance. First we consider hard system down situations in which the system is not functioning at all. We then extend towards performance where the system is running but performing sub-par, and then towards prediction where the system is still functioning according to specification.

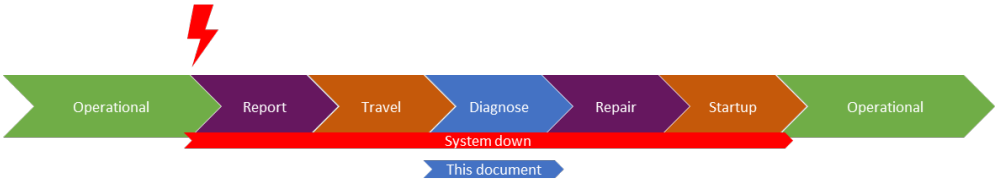
This document reports on the first phase only. Work on performance and prognosis is documented in [2] and [3].



Although we start from diagnosing hard-system down situations, all design choices are made with extension towards performance and prediction in mind.

1.2.2 Corrective maintenance steps

The time required for corrective maintenance can be broken down into several parts.



Report
- Customer trying to restart or repair
- Service desk availability

Travel
- Free up service engineer
- Travel to customer

Diagnose
- Diagnosability (e.g. unique error codes)
- Engineer experience
- Supporting material (e.g. technical service manual)

Repair
- Spare part availability
- Engineer competence
- Serviceability (e.g. component accessibility)

Startup
- Rebooting
- Stabilizing (e.g. heating up to specific temperature)
- Recalibration
- Acceptance test

Phases of corrective maintenance and some impact factors

In this work we focus on the time to diagnose and specifically the supporting material. The intention is to provide a tool for the service engineer, regardless of experience, to increase his efficiency.

Diagnosing efficiency is becoming important because of several trends:
- Increasing product variability makes it harder to gain experience on specific systems
- Increasing product complexity makes diagnosing inherently harder
- Service is delegated to dealer organizations, effectively reducing engineer experience

The proposed approach here also helps in assessing and improving diagnosability (briefly addressed in chapter 2) and increasing service engineer system understanding (*Explainable AI*, briefly addressed in chapter 6) but that is not the emphasis of the work presented here.

1.2.3 Remote diagnostics and Diagnosability

Remote diagnostics allows for an investigation before an on-site visit. This allows for sending a prepared service engineer with the right spare parts to the site.

Diagnosability is a property of a system indicating how easy it is to diagnose the system. Creating a system model as outlined in this report can aid in assessing the diagnosability and e.g. assess the value of adding a particular sensor to increase observability.

The techniques discussed in this report can be used for remote diagnostics and diagnosability but these applications will not be discussed in detail.

1.2.4 Use case

The method described in this report is developed based on the Paper Input Module (PIM) of professional printers developed by Canon. However, we try to ensure that the method is applicable to (high tech) systems in general. With few exceptions, all examples in this report illustrating the method are based on every-day simple “machines” and can be understood with common understanding, not requiring specific printer design knowledge.

2 Method overview

In this chapter we will give an overview of the proposed way of working for diagnoses. Individual aspects will be address in subsequent chapters.

2.1 Current way of working

In short, the typical current way of working is as follows:

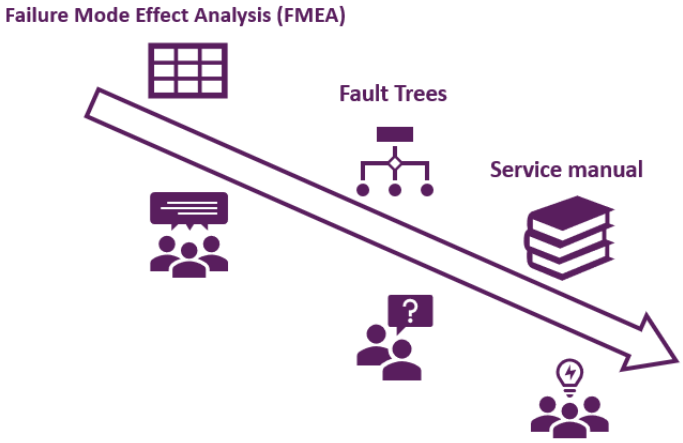


Figure 1 Overview of the typical current way of working

1. After the initial system design, the subsystem design team is assembled in an FMEA-session (Failure Mode Effects Analysis). Its purpose is to identify potential problems and decide, if considered needed, on a mitigation such as redesign or a way for the service engineer to detect and repair the problem. The process is guided by a table (either in Excel or Word) that also captures the result of the session.

2. Based on the FMEA outcome, a technical service manual is written that captures the detect-and-repair tactic. This is not an easy task since multiple failure modes map to the same detection (error code) and it is not clear how a failure mode can be resolved.

2.2 Proposed way of working

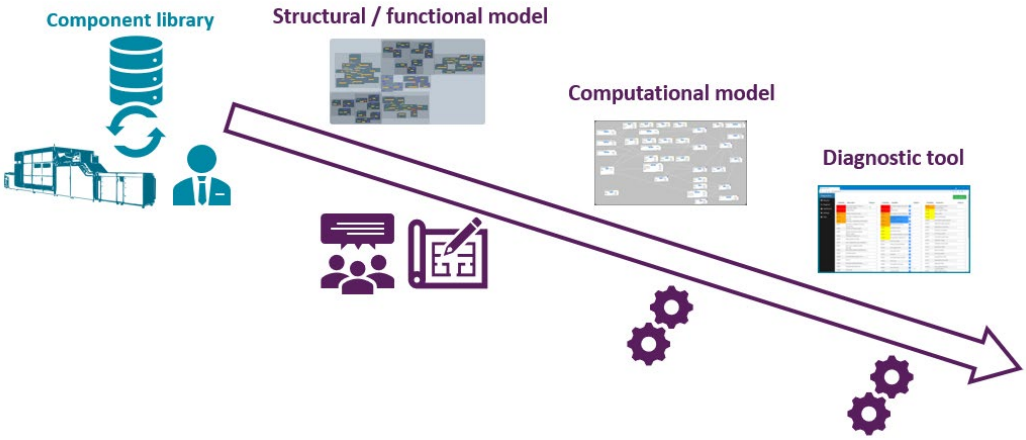


Figure 2 Overview of proposed way of working

The proposed way of working centers around a computational diagnostic model. With such a model, supporting tools can:

At design time

1. Capture failure modes and effects in a structured way. System designers are guided to think about the consequences of known component failure modes.
2. Assess diagnosability, i.e. assess which failure modes are indistinguishable, or provide a metric indicating the level of diagnosability.

At service time

3. During root cause analysis present the most likely root-causes.
4. Suggest the most cost-effective next test to do for further diagnosis.

Instead of building a fixed step-wise procedure or decision tree upfront, the diagnostic tool will interactively suggest the best test to do and use the results for the next step.

The diagnostic model is constructed in 3 step approach:

1. From existing technical information such as design models or technical drawing, create a first model version. Depending on source of information this can be partially automated.
2. Manually augment model to a full diagnostic model
3. Assess component prior failure probabilities from operation, reliability figures, or expert judgement

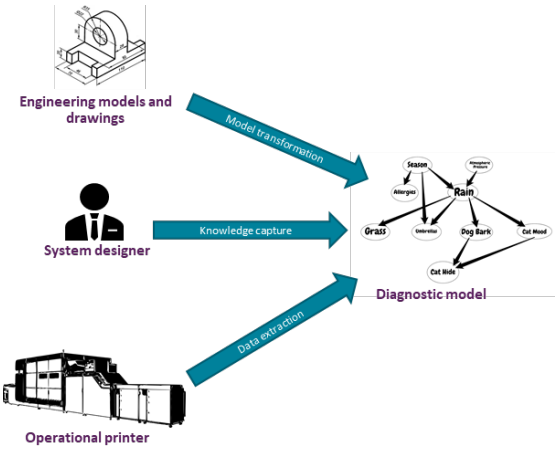


Figure 3 Three different sources of information are fused into the diagnostic model

To give an idea on how this looks like we will quickly show some of early prototypes we developed thus far.

Design tool to build a system description:

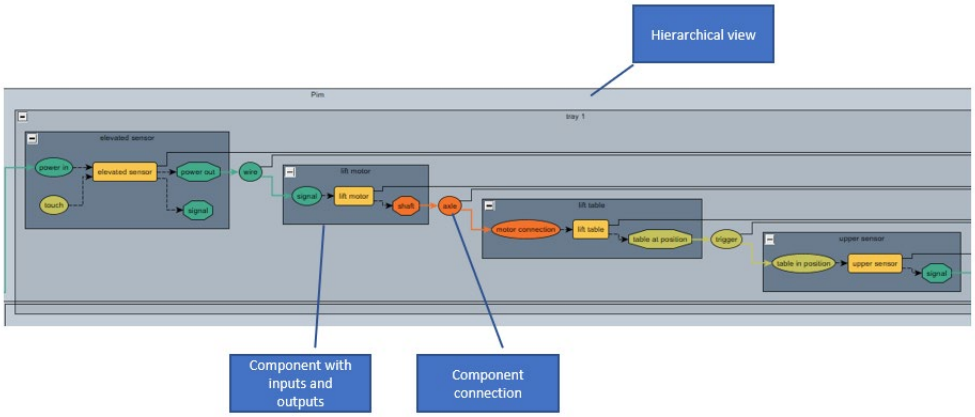


Figure 4 Hierarchical view that resembles system composition. Shown is a part of a printer paper input module (PIM) that contains a tray which in turn contains a sensor, a motor, a lift table, and another sensor. Details in chapter 5.

Service tool for interactive root cause analysis:

Figure 5 shows a screenshot of a prototype of such a tool demonstrating the principles.

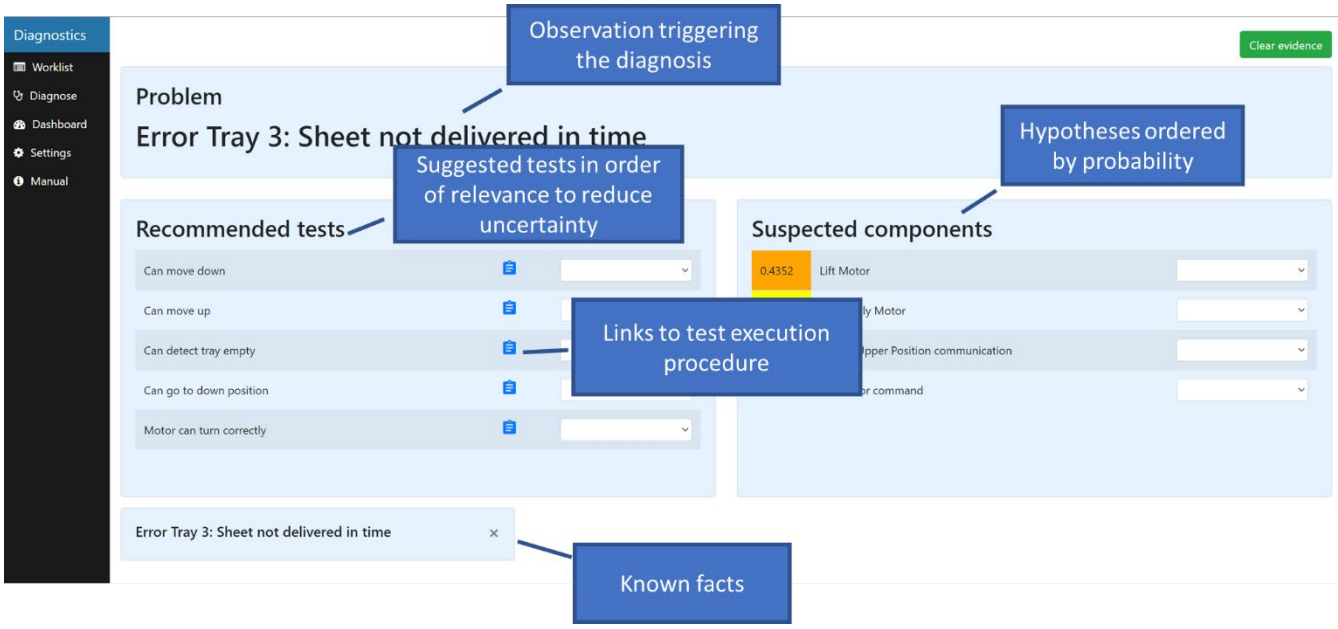


Figure 5 Prototype service interface demonstrating the principles. Details in chapter 6.

3 Bayesian Belief Networks

The core reasoning engine of the diagnostic tool is a Bayesian Belief Network (Pearl, 1988), often referred to as Bayes Net (BN). In this chapter we will describe the fundamental working of BN's necessary to understand the rest of the report. Readers familiar with Bayes Nets can skip this chapter.

3.1 Notation and probabilistic rules

As Bayesian Belief Networks are based on probability theory, we will introduce the basic concepts and the notation used in the remainder of this chapter.

We denote random variables by capital letters (A, B) and their possible values by small letters (a_1, a_2, b_1, b_2). The probability that variable A has value a_1 is then $P(A = a_1)$, or, if the exact value of A does not matter abbreviated to $P(A)$.

The joint probability, i.e. the probability that $A = a_1$ and $B = b_1$ is then $P(A = a_1, B = b_1)$ or $P(A, B)$ in short.

If we know the joint probability, i.e. the probability of all possible combinations of all variables, we can compute the probability of a variable by marginalizing or summing out:

$$P(A) = \sum_b P(A, B = b)$$

This looks easy enough, but if there are more variables this quicky becomes intractable:

$$P(A) = \sum_b \sum_c \cdots \sum_z P(A, B = b, C = c, \cdots, Z = z\ )$$

And typically we need to calculate this for every variable. The Bayes Net tools take into account the independence between variables to use smart algorithms that calculate these numbers efficiently.

The conditional probability, i.e. the probability that A=a1 given that B=b1 is written as $P(A = a_1 \mid B = b_1)$ or $P(A|B)$ in short. The conditional probability is defined as

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

When we reverse the roles of A and B and noting that $P(A, B) = P(B, A)$ we can derive Bayes Rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

which gives the Bayes Nets their name.

3.2 Principles

Bayesian Belief Networks (BN) are used to represent the uncertainty of the world by using probabilities. In general, describing a full joint probability, i.e. define the probability for every possible value combination for all variables, is very difficult and computationally intractable. BN's make this possible by explicitly describing probabilistic dependencies between the variables and smart algorithms to reason.

In a BN, an edge between two nodes indicates that the two variables are conditionally dependent, and, more importantly, a missing edge means that the two variables are conditionally independent. The latter is key to the practical applicability of probabilistic reasoning.

Formally, if a BN has variables $X_1 \dots X_n$, then $P(X_1, X_2, \dots, X_n) = \sum_i P(X_i \mid pa(X_i)$ where $pa(X_i)$ is the set of parents of X_i. The product of conditional probabilities is in general much easier to define and calculate than the full joint probability, but it does require explicit independence assumptions on the domain.

Example:

Suppose we have a fan consisting of a motor with a blade connected with an axle. We say that correct working of the fan is determined by the correct working of the motor, the axle, as well as the blades.

Figure 6 System used in the example is a fan consisting of a motor, blades, and an axle connecting the motor to the blades

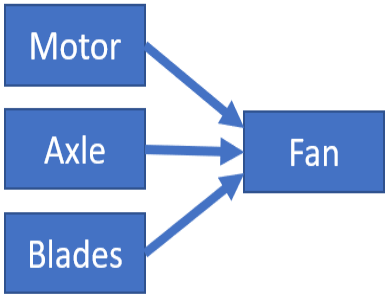


Figure 7 Model of the fan system showing that the condition, or *health* of the fan depends on the condition of the Motor, the Axle, and the Blades. These three conditions are independent.

We assume that a component is either working normally, or is completely broken. So, the component variables have two possible values, typically called *states*: *"normal" and "broken"*.

Failure modes
In this work we typically model a component with states "Normal" and "Broken" if we are not interested in different failure modes. Failures modes will be discussed in chapter 5.

A typical diagnostic question "if we observe that the fan does not function, what is the probability that the motor is the culprit" then translates into calculating $P(motor = broken \mid fan = broken)$. The Bayesian Network can provide this answer if we feed it with some numbers.

For the motor, axle and blades we provide *prior probabilities*. These are independent of each other (according to this model). The condition of the fan as a whole depends on motor, axle, and blade as depicted by the arrows, so we have to provide the conditional probability of the fan, i.e. $P(fan \mid motor, axle, blades)$. In our approach, these *conditional probability tables* (CPT's) are typically very simple: typically only containing zeros and ones, effectively encoding a logical "AND"-
relation.

Motor = normal	Motor = broken
0,99	0,01

Axle = normal	Axle = broken
0,98	0,02

Blades = normal	Blades = broken
0,95	0,05

Figure 8 Specification of the prior probabilities of the component health states in this example

Motor	Axle	Blades	Fan = normal	Fan = broken
normal	normal	normal	1	0
normal	normal	broken	0	1
normal	broken	normal	0	1
normal	broken	broken	0	1
broken	normal	normal	0	1
broken	normal	broken	0	1
broken	broken	normal	0	1
broken	broken	broken	0	1

Figure 9 Specification of the conditional probability table of the health of the Fan, given the health of its constituent components. Here, the relation is deterministic (only 0's and 1's)

Discrete vs Continuous
For ease of explanation we only consider discrete variables. In general, continuous variables can also be modelled in Bayesian Networks. Continuous variables are assumed to be normal distributed, or a mixture of Gaussian distributions, and cannot be the parent of discrete nodes. Continuous variables can always be discretized, thereby approximating the probability density function.

To reason with this model we provide it with some *evidence*, e.g. we observe that the fan does not function. We illustrate this using a tool called Bayes Server. See appendix A for a discussion on tools.

Scenario 1: No evidence.

Here we modelled the system as a Bayes Net, and did not provide evidence yet, that is, we do not have any information of the components or of the working of the fan as a whole. The node "Fan" shows the probability of the fan system being broken, calculated based on the prior probabilities of the component failures and its conditional probability table.

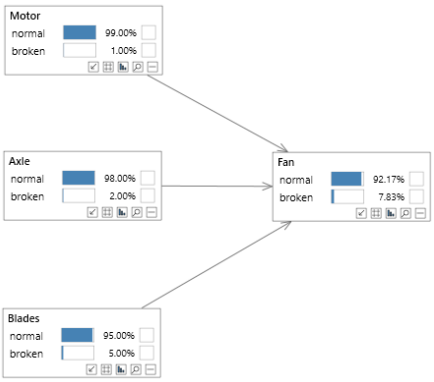


Figure 10 Probability of being broken for every component and the Fan as a whole. Numbers are specified in the tables as shown in Figure 8 and Figure 9.

Scenario 2: Fan does not function.

Suppose we know that the fan is broken, e.g. we do not feel the air flow. What is wrong?

We add the observation that the fan is broken as evidence in the network. As a result, the network will recalculate the probabilities of all other nodes in the network, the so-called *belief update* or *inference*.

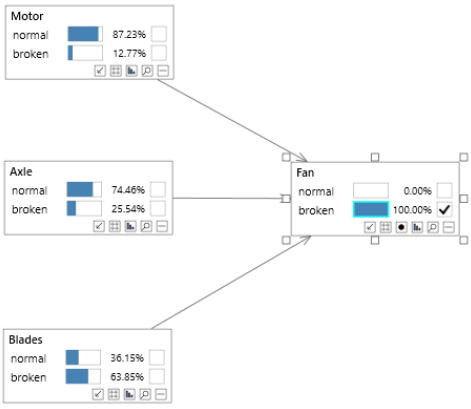


Figure 11 Inserting evidence that the fan is broken (checkmark in the Fan node) changes the probabilities of the components being broken

From the network we can see that the probability of failure of the motor, axle, and fan increased. In particular we see that the Blades are the most likely component to have failed.

Noisy-OR

The scenario sketched here is so common that many tools support this by so-called Noisy-OR nodes. This greatly eases specification of the CPT and allows for belief update tricks so that even nodes with hundreds of parents can be handled with ease.

Scenario 3: Fan does not function but motor spins.

Let us assume that the fan does not move the air but we can clearly hear the motor spinning. So, we have additional evidence that the motor is working correctly.

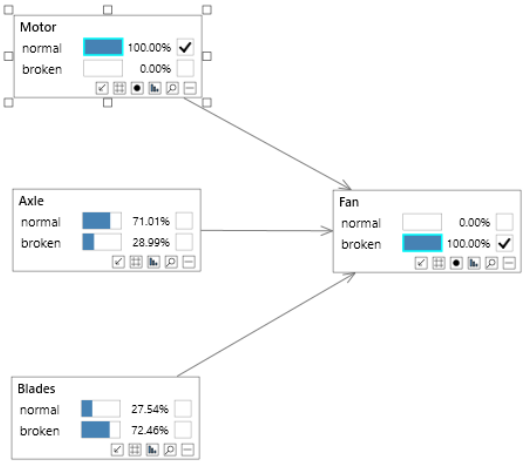


Figure 12 When adding evidence that the Motor is fine, probability of failure of the other components increases

After the belief update, we see that the probability of failure of the other components increases.

Note that although the failure probability of motor and blades were independent, by adding evidence they have become dependent: if we now would decrease the probability that the motor failed, the probability of the blades failing would go up (intuitively: something must be wrong!)

Explaining away
The scenario described above is commonly referred to as *explaining away:* if one cause is ruled out, other causes for the observed effect become more likely.

Scenario 4: Motor functions correctly.

Figure 13 When the Motor is fine, the probability that the fan is broken decreases. This is to illustrate forward reasoning, compared to backward reasoning in previous scenarios. Bayes Nets can handle evidence at root nodes, leaf nodes, or nodes in the middle.

It is important to understand that the arrows do not indicate the direction in which information flows. Bayesian Networks can be used to reason in all directions. Any node can be fed with evidence and all other nodes will be updated accordingly. The direction does play a role when several nodes are connected, as explained in the next section.

Soft evidence
One can even provide evidence that is not 100% certain, the so-called soft evidence. We will not discuss that in this introduction.

3.3 Causal Networks

If we have a cause and an effect that we want to model as a Bayes Net, we have two options, as shown in Figure 14. Probabilistically these models are equivalent, as we can easily prove using basic probability theory

$$\boldsymbol{P(cause, effect)} = \boldsymbol{P(cause \mid effect)} \cdot \boldsymbol{P(effect)}$$
$$= \boldsymbol{P(effect \mid cause)} \cdot \boldsymbol{P(cause)}$$

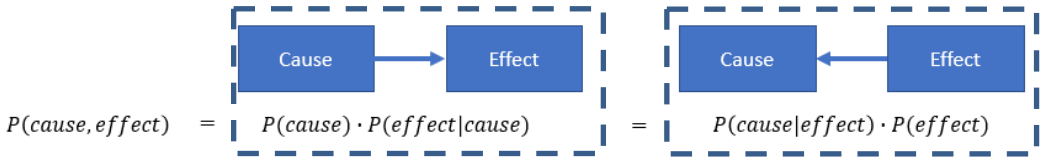


Figure 14 Two equivalent networks

In practice there is a difference since in general it is easier to estimate prior probability of a cause and how the effect is related to the cause than the other way around. But, there is no fundamental difference.

If we expand this network to 3 nodes A,B, and C, then some networks equal (in the sense that they represent the same joint probability distribution) while others are not. See Figure 15. The network on the right represents a different world.

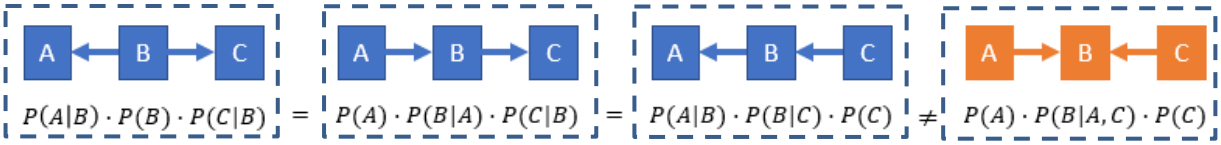


Figure 15 Four different ways to build a network of three variables. Three of them are equivalent, the fourth is not

When networks become larger, it becomes increasingly difficult to model the world as intended. To verify correctness one would have to add different sets of evidences throughout the network and verify the effect on the other nodes. However, as a rule of thumb one can say that if you always add arrows from cause to effect, then the network will behave as expected. Such a network is a *causal network*[1].

Causal networks have an additional advantage that we will exploit when we talk about tests: they can deal with *interventions*, i.e., they behave correctly when we actively change something in the system. This could be used to obtain more information, or to explore what-if scenarios. Refer to section 5.10 where we deal with interventions.

3.4 Building a model

To build a Bayesian Network the following steps are required:

1. Define variables
2. Define states of all variables (and the discretization if the value is continuous)
3. Define network structure
4. Define probability tables

In general, all of these steps can be learned from data or done by hand using domain knowledge.

[1] One could also systematically draw arrows from effect to causes. That would lead to a diagnostic network. But as stated in the text, the probabilities to be specified in such a network are in general hard to obtain.

3.4.1 Learning from data

In the ideal world, one would like to generate a model by just providing it with data. Unfortunately, there currently are some limitations.

Step 4, learning probability tables, is a common technique but gets more difficult if more variables are unobserved (*latent variables*). What is very well achievable is to learn the prior probabilities of the component failures. Chapter 4 is dedicated to that.

For step 3, aptly called *structure learning,* algorithms do exist but require a lot of data, more than we typically have in diagnostic systems where failures are not abundant. To find *causal* networks is even harder.

For steps 1 and 2 the algorithms are basically in a research stage and not readily applicable.

3.4.2 Building by hand

Building a model by hand requires domain knowledge and good understanding of the Bayesian modelling principles. A major downside of Bayes Nets and corresponding tools is that they do not support composition or hierarchy, resulting in very large flat networks. As we want to model an entire system up to serviceable component level, and a system has thousands of serviceable components, such model will have thousands of nodes. A flat model would be very hard to build and maintain.

To make creating the models easier, we have devised a way of working that allows system modeling in a compositional way and then to generate the (flat) Bayes Nets automatically. Chapter 5 is devoted to that.

4 Estimating component failure probabilities

One of the elements required to build the Bayes Nets is an estimation of the prior probabilities of its variables, that can either be provided by knowledge experts, extracted by historical data, if available, or a combination of the two.

In our specific use case the problem translates to calculate the failure probabilities of different components of the Printer Input Module (PIM).

A first estimation of the prior failure probability can be obtained based on the visit dataset. This dataset contains the log of all service visits, with reference to the time of the visit and the serial number of the system, and a list of parts that have been ordered by the operator concurrently to the visit. We stress the fact that the dataset is filled in manually, and that it does not contain detailed information about which component has been repaired or replaced during the service.

Furthermore, it is necessary to take into consideration that some of the parts mentioned in the visit dataset can be used also for other components outside the PIM.

The prior probabilities of a part failure are therefore calculated from the service visit data according to the following steps. For each part:

- Compute the number of visits for which it was ordered;
- Retrieve on how many components in the PIM it is used;
- Retrieve on how many locations inside or outside the PIM it is used;
- Apply the ratio between [3] and [4] to estimate from [1] the number of visits in which the part is replaced in a PIM component;
- Normalize the result of [4] to express it as a probability.

Other options to enrich the Bayes Net with data driven information have been investigated:

- Linking the component failure probability to the presence of specific (series of) errors.
- Linking the component failure probability to the way the system is used (e.g. how frequently the printer switches between trays).
- Include in the Bayes Net information about media type, and how this impact on the occurrence of certain errors.

The first direction has been explored mainly in 2020. By using the visit dataset described above and a dataset containing all the logs of warnings and errors, different models have been trained to predict which part was ordered given that specific (pattern) of errors occurred. The success of this analysis has been very limited, mainly because lot of relevant information is still not captured in any of the available dataset. More details are presented in [7].

The second and third approach are currently being explored, by exploiting the data available, describing the physical properties of the media used for each print.

5 System Modeling

Building a Bayesian Belief Network by hand requires specific expertise and is error prone. Additionally, BN's do not have the hierarchy or composition, so maintenance and reuse is very difficult.

Our approach is to enable a system designer to model the system in a friendly way, supporting composition and reuse, and generate the BN automatically from that model.

In this chapter we describe how this approach looks like.

5.1 Governing principles

To build a system model efficiently, we use the following approach:

1. Reuse component and subassembly models
    - The idea is to have a library of component models that can be used to assemble a system model
2. Build according to the system structure
    - Relatively easy
    - Typically, structural models are available: think of electrical diagrams, CAD drawings of mechanical structure, etcetera. These models could be used to (partially) build the diagnostic models.
3. Use a compositional approach
    - By using a compositional approach, the models remain manageable. At the same time, this enables subassembly model reuse across system variations and product families.
4. Use a functional description and composition where structural composition is not applicable. Control loops are a typical example of this: from the structure of a loop one cannot determine the behavior. As loops, and especially control loops, are ubiquitous in high tech equipment, we devote an entire section to them (section 5.6).

To illustrate the approach, and especially the difference between a structural and a functional approach, we will use the fan example introduced in chapter 3.

5.2 Structural model

Note that the code and graphs in this document and particular in this chapter are used for the feasibility study and are an implementation detail. Final representations will be developed by Canon development, fitting in their way-of-working, e.g. in the MPS-based model based system engineering framework.

For the structural model the steps are as follows:

- For a component we
  - define the inputs and outputs
  - define the failure modes
  - define the relation between inputs and outputs for the normal behavior and for every failure mode
- For an assembly we define
  - the components that are part of the assembly
  - how the components are connected (structurally)

For the fan, a model would then look like Figure 16.

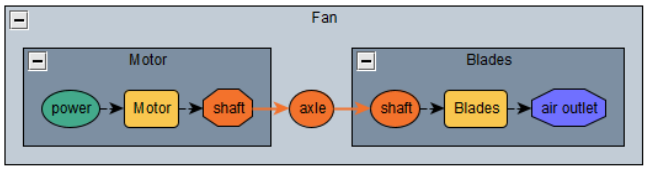


Figure 16 Structure of the Fan assembly. The shaft of the Motor (output) is connected to the shaft of the Blades (input) through an Axle.

The structural graph shows that the shaft of the motor is connected to the shaft of the blades through an axle. Behavior of components such as the motor, axle, and blades are defined in a library that is used in the assembly composition. Based on the individual component behaviors in combination with the structural description, the behavior of the assembly (fan in this example) is automatically derived.

5.2.1 Behavior specification

To define the behavior of a component, one has to specify the input-output relations for the component for its normal behavior and each of its failure modes.

Without a formal definition we introduce some terminology based on the example above.
A *component* (Motor) has an *input port* (power) and an *output port* (shaft). Every port has a *modality* associated with it. In the example the power port is associated with Power. A modality has *properties*.

In the example, the motor has a power socket as input and a shaft as output and transforms modality Power (capital P) into the modality Movement. Here we are only interested in whether there is Power and Movement or not, so for both entities we define the property “Present”, that can be Yes or No.

An example of the motor is shown in the tables in Figure 17 and Figure 18.

NORMAL	
power (Power)	*shaft (Movement)*
Present	Present
Yes	Yes
No	No

Figure 17 Table specifying the normal behavior of a motor. The shaft moves if and only if there is power supplied to the motor.

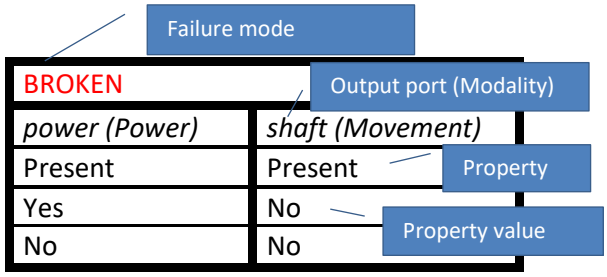


Figure 18 Table specifying the general failure mode "broken". The shaft will not rotate, regardless of the power.

To elaborate on the behavior specification, let's take a more complicated example.

Suppose we have an electrical heater that heats water flowing through when it supplied with power.

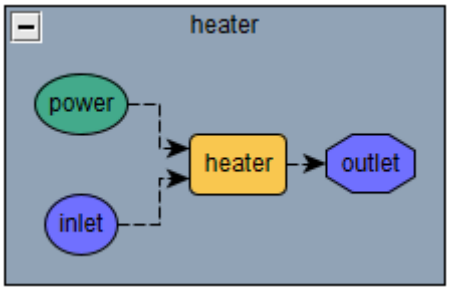


Figure 19 Structure of a Heater component that increases water temperature when powered.

For the Power, we are only interested whether there is power or not, and for the water we are interested in its flow rate (Low or High) and its temperature (Cold, Warm, or Hot). The full specification for the normal behavior would then look like Table 1 below.

NORMAL				
power (Power)	*inlet (Water)*		*outlet (Water)*	
Present	**FlowRate**	**Temperature**	**FlowRate**	**Temperature**
Yes	Low	Cold	Low	Warm
Yes	Low	Warm	Low	Hot
Yes	Low	Hot	Low	Hot
Yes	High	Cold	High	Warm
Yes	High	Warm	High	Hot
Yes	High	Hot	High	Hot
No	Low	Cold	Low	Cold
No	Low	Warm	Low	Warm
No	Low	Hot	Low	Hot
No	High	Cold	High	Cold
No	High	Warm	High	Warm
No	High	Hot	High	Hot

Table 1 Full specification of the heater in normal operating condition. For every input combination (2x2x3=12 combinations) the corresponding output is specified.

In principle one should define the behavior for every input combination. For components with many inputs or with input types that have many properties, or properties that have many states, such a table would get very large and be cumbersome to specify. In practice, a component will affect a limited number of properties so the complexity of the specification can be reduced drastically. See Table 2 for some ideas.

NORMAL				
power (Power)	*inlet (Water)*		*outlet (Water)*	
Present	**FlowRate**	**Temperature**	**FlowRate**	**Temperature**
Yes	*	Cold	inlet	Warm
Yes	*	Warm	Inlet	Hot
Yes	*	Hot	inlet	Hot
No	*	*	inlet	Inlet

Table 2 Shortened specification of the same component. Here, * (star) means "don't care" and *inlet* means that the value is identical to the value of the inlet input.

Other abbreviations that we use extensively in our modeling are the ELSE and ALWAYS keywords.

NORMAL			
In1 (X)	*In2 (Y)*	*In (Z)*	*out (X)*
Present	**Present**	**Present**	**Present**
Yes	Yes	Yes	Yes
ELSE			No

Table 3 Use of the ELSE keyword. Many times a component only has a successful output if all of its inputs are present (AND relation). In all other cases the component cannot do its function.

BROKEN			
In1 (X)	*In2 (Y)*	*In (Z)*	*out (X)*
Present	**Present**	**Present**	**Present**
ALWAYS			No

Table 4 Use of the ALWAYS keyword. This is often used for failure modes where there is never an output, regardless of inputs. Note that this could also be expressed by using * (don't care) for every input.

5.2.2 Specification language

For practical purposes we need tooling that allows to quickly specify a system, supporting both the structure (graph like) and behavior rules (table like). Most important, however, is that it fits the systems designers way of working and ideally merges with the (model-based) development tooling already in use. For our proof of concept we specify the structure graph and behavior tables in the Python programming language as an internal domain specific language.

Important aspects of the specification language is that it allows for instantiations and composition. E.g. we can define a Tray with all its components and then define a Paper Input Module that has 4 trays by instantiating 4 trays.

5.2.3 Translation to Bayes Net

From a structural model we use an automated process, implemented as Python scripts for now, to generate a Bayes Net.

For every component we create nodes for the inputs and outputs and create a node representing the components health.

The health nodes have the states “OK” denoting normal behavior, and an additional state for each failure mode. As health nodes do not have parent nodes, we have to specify the prior probabilities. These probabilities can be obtained from generic reliability information (“Motors of this brand have 0.01% failure probability”), from historic data (“transport belt motors have 0.02% probability of failure”), or even user specific data (“This motor in this printer is used in heavy

load situations and therefore has 0.05% probability of failure"). Chapter 4 is devoted to this subject.

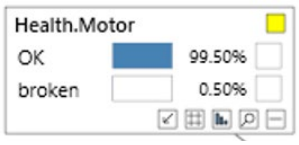


Figure 20 Health node of the Motor component in the Bayes net. Here, the Motor has one generic failure state "broken".

The outputs causally depend on the inputs and the health of the component, so links are added from the input nodes and the health node to all output nodes.

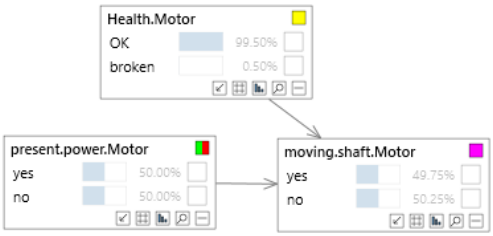


Figure 21 Nodes of the Motor component. The output node has the input node and the health node as parents.

The conditional probability table is created according to the component's input-output relation table. Note that because of the deterministic relationship between the nodes, the conditional probability table only has 0's and 1's. In general CPT's can have any probability values describing probabilistic behavior.

present.power.Motor	Health.Motor	moving.shaft.Motor = yes	moving.shaft.Motor = no
yes	OK	1	0
yes	broken	0	1
no	OK	0	1
no	broken	0	1

Figure 22 Conditional probability table of the Shaft output node, i.e. P(moving.shaft.Motor | present.power.Motor, Health.Motor)

The output node of a component is then linked to the input node of another component representing the connection between them. But as connections can also fail (a wire can get disconnected, an axle can break), an additional Health node is introduced representing the failure modes of the connection.

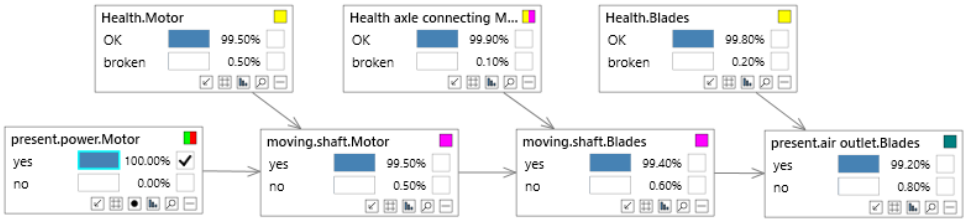


Figure 23 Bayes net generated from the structural definition of the fan. The shaft of the motor is connected to the shaft of the Blades but as that connection (an axle) can fail too, a node for the health of the axle is added.

5.3 Functional model

In contrast to the structural model, the functional model takes a functional decomposition as starting point. Figure 24 shows a somewhat contrived example. Instead of following the sequential causal steps of energy transportation, the functional model has a parallel structure where the function to blow air depends on three different subfunctions, each realized by a component.

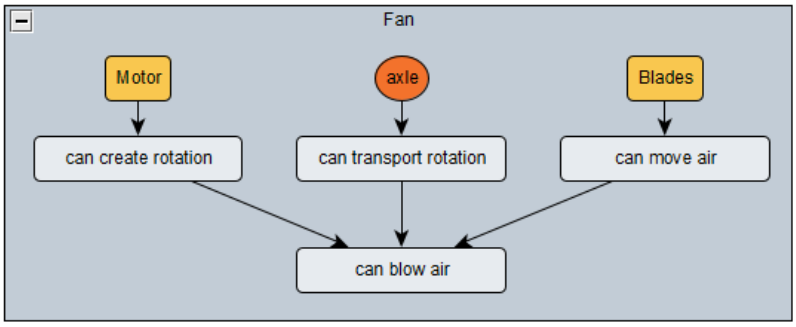


Figure 24 Functional view of the Fan. Components provide a function, the fan needs all functions to work correctly.

Functions, here also referred to as capabilities, do not indicate that the function is actually active. In the example, although the Fan can blow air, it needs power to actually blow air. This is depicted in Figure 25, where it shows that there is only air if there is power and the fan is capable of doing its functions. This connects functions to actual observations.

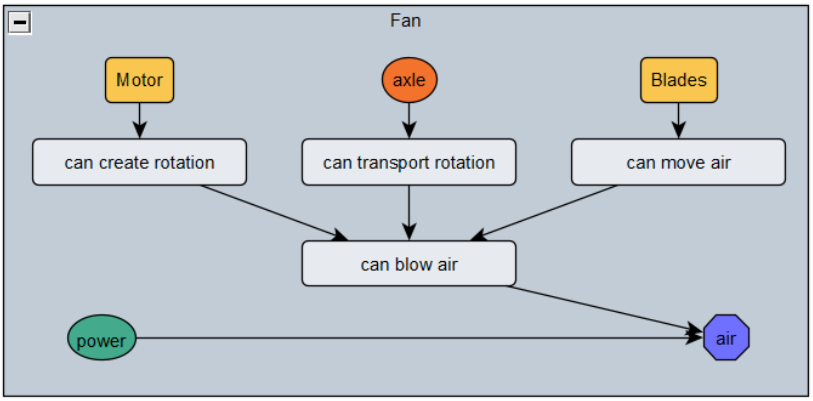


Figure 25 Fan functional model including actuals

Note that it is a design decision to exclude the power from the motor capability. Here power is considered exogenous to the fan.

5.3.1 Translation to Bayes nets

Translating a functional model to a Bayesian belief network is relatively straightforward: the required subfunctions are parent nodes to a function node, typically in a deterministic “and”-function: the function is only active if and only if all its subfunctions are active.

5.4 Comparison structural versus functional modeling

In the previous sections we presented two different modeling approaches: structural and functional. Here we will compare the two approaches.

5.4.1 Structural approach

The structural approach has the big advantage that one can take the physical layout and almost one-on-one model the system by connecting the component models according to this layout. If the physical layout is available in some sort of formal description, such as standardized electrical schema or CAD drawing, this can even be (partly) automated.

Another advantage of the structural modeling is that the inputs and outputs of the components are directly observable (measurable). Connection to data (such as logging) is relatively easy because data are measurements on physical entities that have a one-on-one mapping to the structural model elements.

Major limitation of the structural approach is that it does not work for loops, including control loops. To solve that we need additional abstraction mechanism that is explained in the next section.

Another disadvantage is that is it more difficult (compared to the functional models) to associate tests with the model as tests typically test a particular function.

5.4.2 Functional approach

The functional approach has the benefit of large modeling freedom. At the same time this could be a disadvantage as it requires expertise to construct a model. Although functional descriptions could help in constructing a model, these are typically not exact enough to be easily converted in a model for diagnostics. Human competence is important here. When functions are chosen wisely, the connection to available tests is straightforward.

Inserting data from sensor readings such as functional logging into the functional model is not trivial. To be able to use these data requires additional effort. Direct observation for a functional model could be present in an event log if function failures are logged. This requires a one-on-one relation between the definition of functions in the model and the event log, something to consider in the functional decomposition.

5.4.3 What to choose?

Our proposed strategy is: Use structural approach where possible, use functional approach when needed. The best balance will emerge when we start modeling bigger parts of the printer. Important is to understand that there is an option to choose between them.

5.5 Property dependencies

5.5.1 Problem statement

In our modeling framework we reason about the properties of modalities. The examples in this document are purposely simple and usually only consider a single property for a modality but in general there are more.

This leads to the following complications:

1. The Bayes nets get very complicated because for every property we generate a node.
2. Not all combinations of property values are possible. E.g. if there is no water, then what is its temperature?

We address these issues in the following subsections.

5.5.2 Multiple variables per node

Although the Bayesian belief networks are not directly visible to end-users, for understanding and debugging it is beneficial to have a Bayes net that follows the system structure. See the straight forward system of Figure 27, consisting of three components in series.

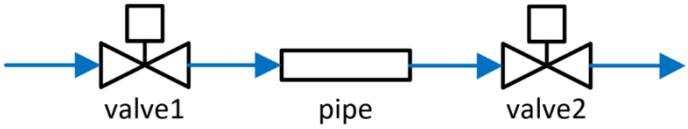


Figure 27 Example system of two valves connected through a pipe transporting water from left to right

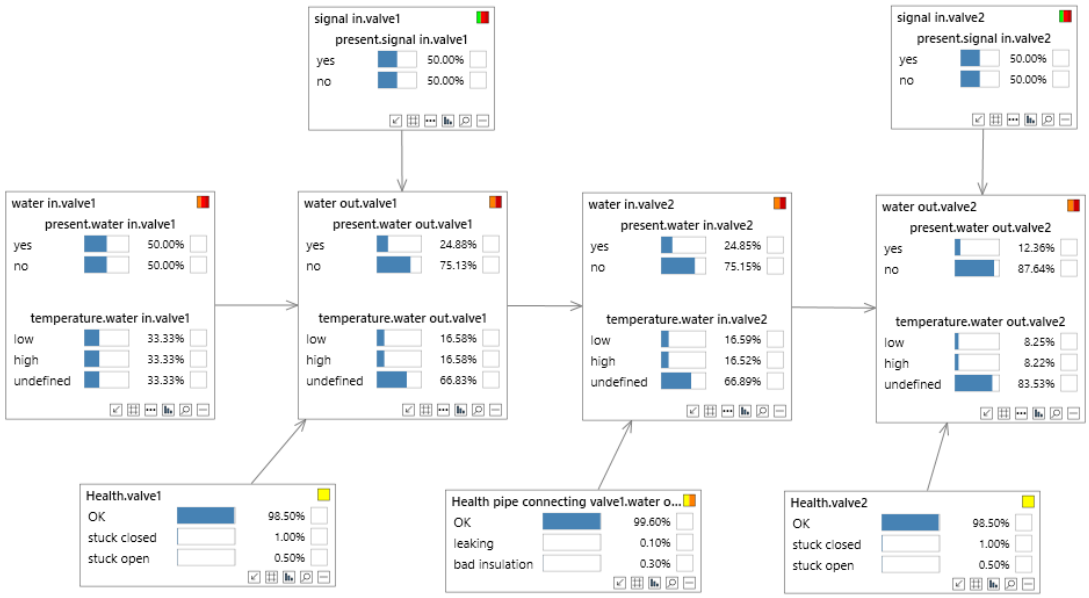


Figure 26 Network with multiple variables per node.

When interested in both the presence and temperature of water, we can model both properties of the modality water as variables, but aggregate them in a single node that represents water as in Figure 26.

5.5.3 Constraints

When ”impossible” combinations are still possible in the model, the probabilistic inference can give erroneous results. Consider the previous example when we measure Low temperature at the output of Valve 2.

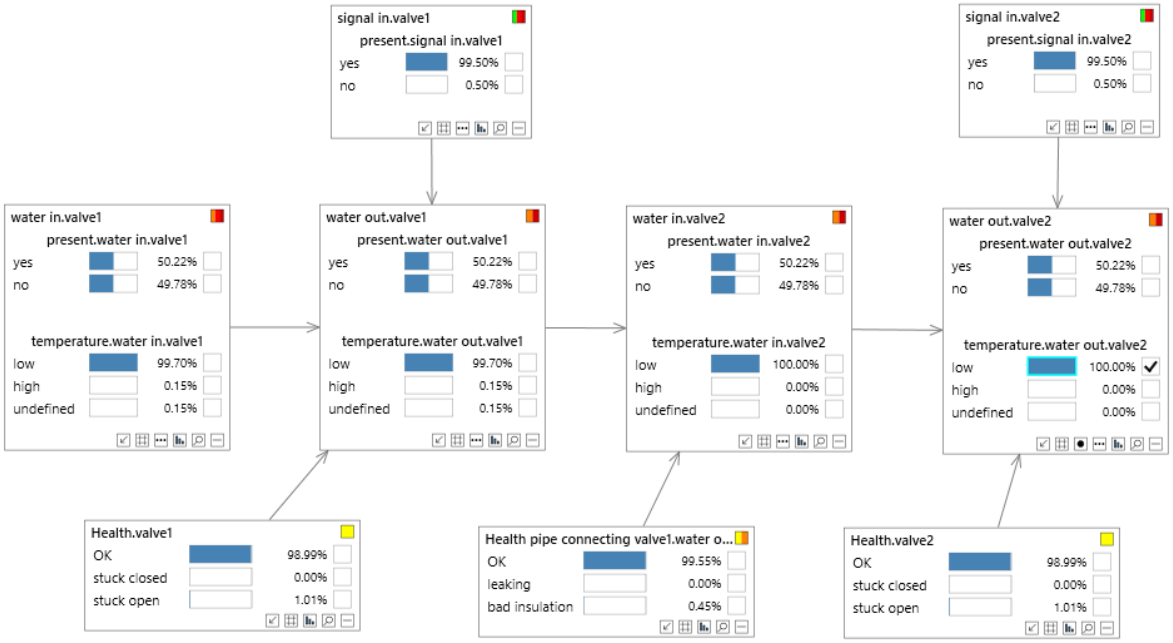


Figure 28 Network without constraints. Even though we measure a low temperature at the output, the network infers that there could be no water.

Clearly, there must be water present throughout the whole system but everywhere we see the possibility of no water being close to 50%.

To tackle these problems we introduce *Constraints* that can be applied to properties. In this case we can specify that having water and an undefined temperature is impossible, and that no water with Low or High temperature is impossible.

Technically, in the resulting network the input nodes then get a prior probability distribution that has zeros at the impossible combinations such as shown in Table 5.

temperature.water	present.water in.valve1 = yes	present.water in.valve1 = no
low	0,333333333333333	0
high	0,333333333333333	0
undefined	0	0,333333333333333

Table 5 Prior probability distribution of the water input node reflecting the fact that certain water property combinations are impossible.

In the example, now with the constraints applied, the network correctly concludes that measuring a low temperature at the output implies that there is water everywhere in the system. See Figure 29.

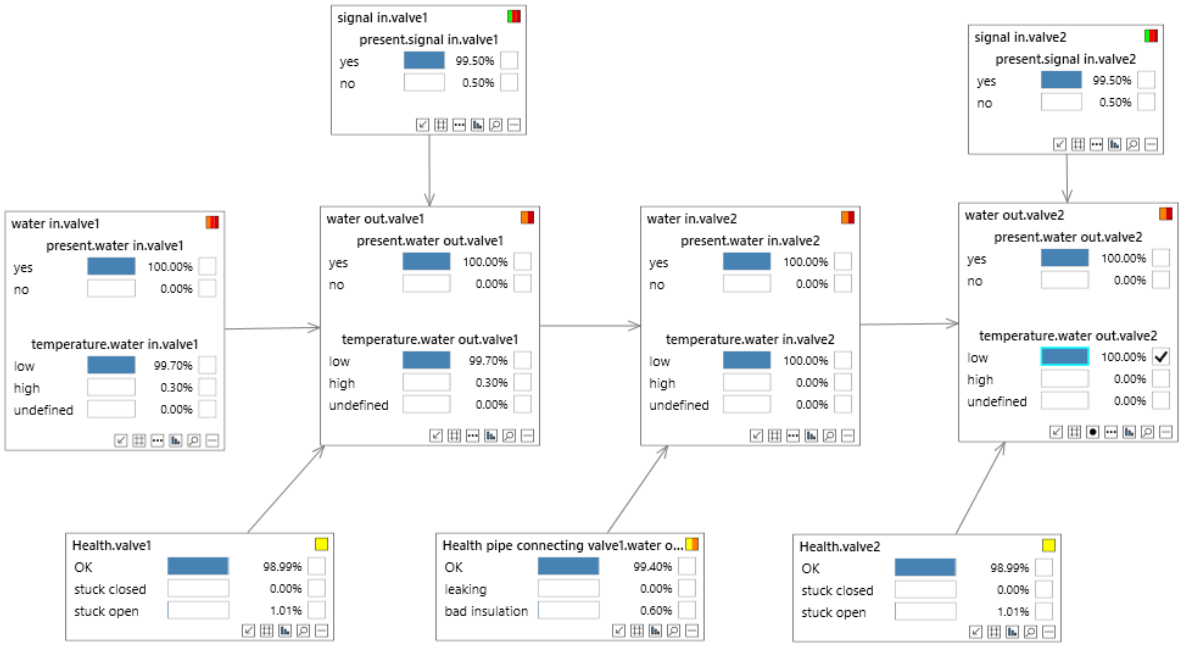


Figure 29 Network with constraints applied correctly infers that low temperature at the output implies water everywhere.

5.6 Loops

5.6.1 Problem statement

In many situations we have a choice between structural and functional modeling. Structural modeling has the advantage of strong correspondence with the system structure, so existing design documentation can be used as starting point for such model. However, there is one particular area we encountered during our investigation in which structural models have shortcomings and that is when there are control loops.

The issue is that the direct translation from a structural model with a loop to a Bayesian belief network as outlined in Section 5.2 will not work because a Bayesian belief network cannot have loops, it is a directed *acyclic* graph.

5.6.2 Example

As an example we take a house heating system consisting of a furnace, a sensor and a thermostat controlling a valve.

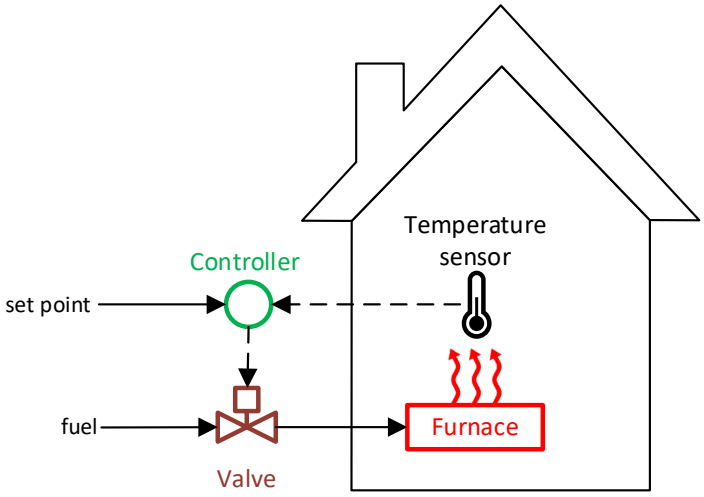


Figure 30 Schematic of the house heating example

Whenever the furnace is burning, the temperature in the house rises and the thermostat will close the valve causing the furnace to stop burning and as a consequence the temperature will drop.

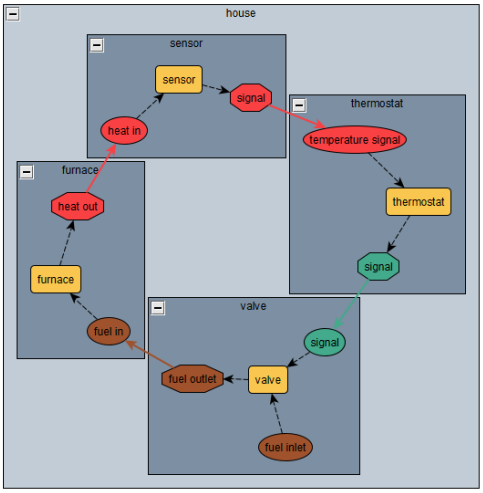


Figure 31 Structural model of house heating example with the loop clearly visible.

Figure 32 House heating generated "Bayes net" with the loop. The Bayes Server tool warns that this is not a directed acyclic graph and is therefore not a valid Bayes net.

5.6.3 Solution

Key observation is that the loop only exists because we do not take time into account. The seemingly obvious solution to model time explicitly will result in very complex models. See Appendix C for some alternatives to the method described here.

The solution is to first regard the loop as a single entity that eventually reaches a stable situation. If diagnosis indicates that the loop has a fault, the components that are part of the loop are further examined by breaking the loop.

Example

In the house heating example this approach would lead to the following structure:

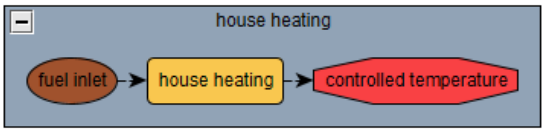


Figure 33 House heating loop represented as a single component

Note that the output of this loop component is “controlled temperature”. This captures the fact that there is time involved. In particular this implies that if we want to add evidence to this node, we will have to see if the temperature is indeed “controlled”. This is more complicated than merely measuring a temperature. It requires to see that if room temperature is below setpoint, then eventually the temperature is stable. The exact meaning of “eventually” and “stable” depend on the control loop characteristics.

The previous discussion centered around control loops as they are abundant in mechatronics systems but the same principles apply to other loops such as coolant fluid flowing in a loop through the system.

Steps

The approach is then described as follows:

1.	Designer	Construct model by connecting primitive components according to system description.
2.	Tool	Check for loops. When a loop is detected, the designer is warned that a loop description is required.
3.	Designer	Introduce a component that represents the loop with a higher level concept describing the output of the loop.
3a.	Designer	Add behavior of this new component based on combination of failure modes of every component in the original loop.
3b.	Designer	Connect inputs and outputs of the new component to the rest of the system.

Our prototype implementation now supports the infrastructure for these models by introducing a “Super Component”, which captures the loop behavior as described above. Validation of the method on real-world cases is still ongoing.

5.7 Upstream effects

5.7.1 Problem statement

The models discussed up to now assume that a failure mode only affects components “down the chain”. However, this not always true. For example a water pipe that is blocked at its exit, causes a pressure increase and flow decrease at its entry. Another example is a short-to-ground failure mode in an electrical circuit that will drain power from all components attached to the circuit. We will use the latter example to explain how we handle such situations.

5.7.2 Example

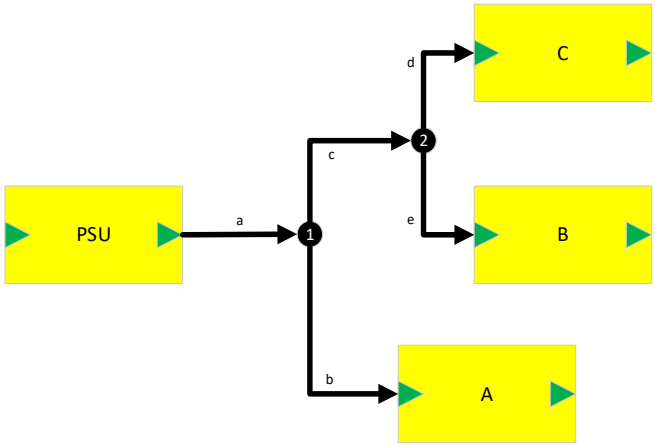


Figure 34 Example electrical system

In the example of Figure 34 , we have a Power Supply Unit (PSU) that is connected to three components A, B, and C through the wires a-e. The wires are joined using wire joints 1 and 2 and together form a cable tree.

If one of the wires, e.g. wire c would have a short to ground, all components, including A, would stop functioning. This is different from a failure mode such as a broken wire that would only affect components B and C in case wire c is broken, as shown in Figure 35.

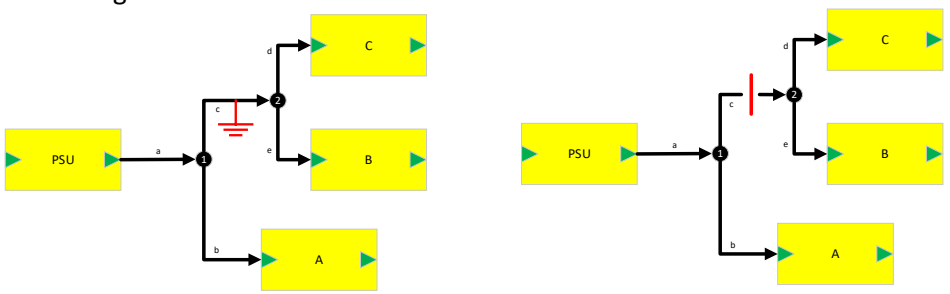


Figure 35 Failure modes of wire c. Left: short to ground that has an effect on components A,B, and C; Right: broken that only effects components B and C.

To recall, for diagnosis, the structural description is translated into a Bayesian Belief Network according to the following rules:

1. Every wire gets its own “Health”-node representing the failure modes of a wire: ok, broken, short to ground.
2. Every wire joint gets a “Health”-node just as every other system component. The normal behavior is that when there power in its input, both outputs have power and if there is no power on the input, neither output has power.

Such a network, however, would not capture the correct behavior of a short-to-ground failure of a wire as it would not propagate the error from that wire upward the chain. In other words, the failure mode does not only causally change the output of a wire, but also its input.

5.7.3 Solution

Extending the model with a short-to-ground can be done in different ways. Here, we show the preferred way. For alternative solutions refer to Appendix B.

The basic principle is to introduce a node that captures upstream effects. In the wire example we introduce an “Upstream short-to-ground” node. This node is then an *input* to the node *preceding* it in the structural flow.

As a consequence, every node that has a Power output must now also specify its behavior when the output experiences a short-to-ground from the *following* component. See Table 6.

NORMAL		
power in(Power)	***Power out Upstream (Power)***	***Power out (Power)***
Present	**Effect**	**Present**
Yes	Ok	Yes
Yes	Short	No
No	Ok	No
No	Short	No

Table 6 Specification of the output given its inputs, including the input from the following component

Additionally, we now need to define the upstream behavior as a result of the upstream effect of the *following* components. In the example of a wirejoint (see figure below) this means that we need to define the wirejoint upstream effect based on the upstream effects of its successors. See Table 7 for the Normal situation.

NORMAL		
Output 1 upstream	***Output 2 upstream***	***Input Upstream***
Effect	**Effect**	**Effect**
Ok	Ok	Ok
Ok	Short	Short
Short	Ok	Short
Short	Short	Short

Table 7 Specification of the Upstream effect of the wirejoint

The generated Bayes net for a wire joint in its immediate environment then looks like Figure 36.

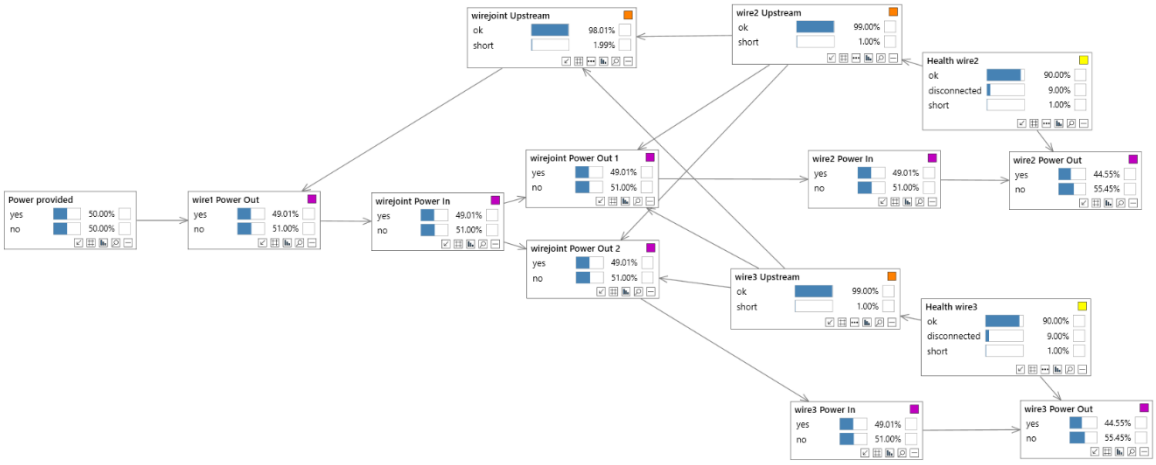


Figure 36 Generated Bayes Net of a wirejoint splitting wire1 into wire2 and wire3

The example shows a wire joint, that cannot fail, connected to two wires that can fail by being disconnected or having a short to ground. It is constructed according to the following principles:

1. Because a wire has a failure mode that causes an upstream effect, it generated a node to that effect (see wire2 Upstream).
2. Because the wire joint is connected to something that has upstream effects, all its output nodes take these upstream effects as parents.
3. And, it defines a node that captures the combination of upstream failures to propagate this to the components upstream.

Note that although we do see many edges point in the backward direction, this approach does not introduce cycles in the Bayes Net.

In a scenario when the is no power at the end of wire3 as shown in Figure 37. the probability of having a short to ground failure of both wire3 and wire 2 increased.

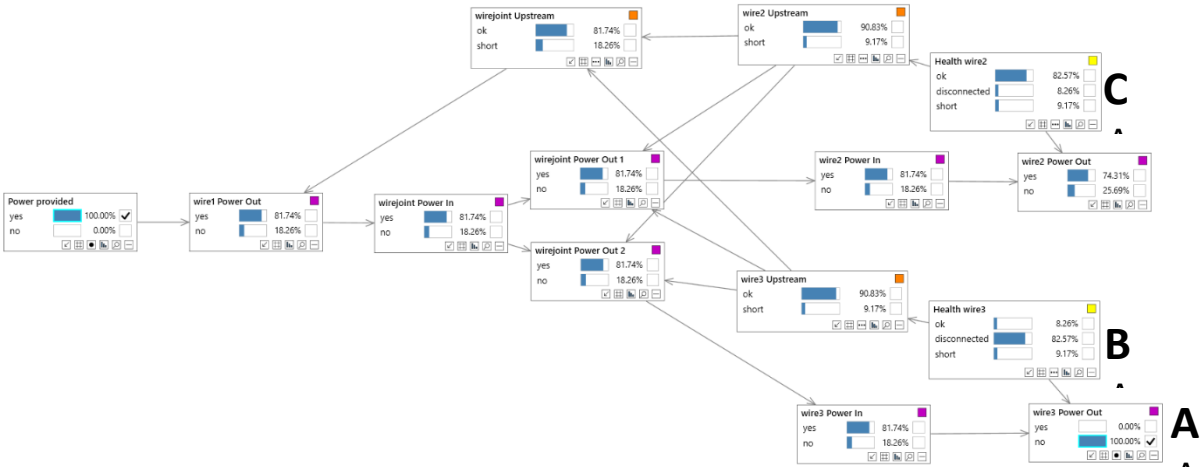


Figure 37 Wirejoint scenario where we observe no power at the output of wire 3 (A). This increases the probability of a short to ground in wire 3 (B) as well as wire 2 (C).

In a scenario where we know wire3 has a short to ground failure, we know for sure there is no power at the output of wire2 and the short to ground failure is propagated to the inputs of the wire joint (output of wire1). See Figure 38.

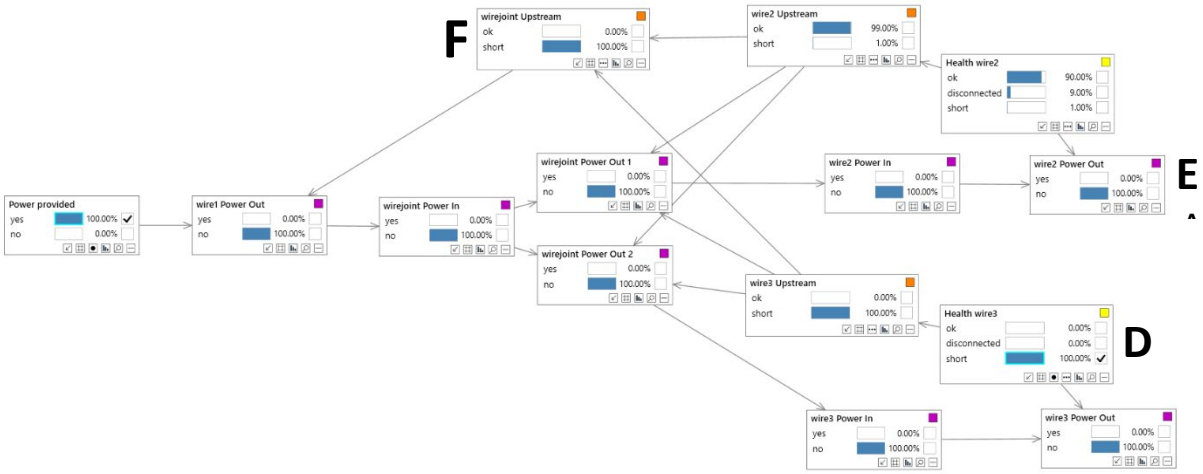


Figure 38 Wirejoint scenario where we know wire3 has a short to ground failure (D). As a result, there can be no power at the other output (E), and the short-to-ground effect is propagated upstream to earlier components in the chain (F).

Currently (end 2022), this way of working is not yet supported by our prototype modelling language and tools. However, we have considered multiple design and implementation issues that could arise. For one, the upstream nodes in the previous example look to have a failure mode as state but this is actually incorrect. What they describe is a property of the modality (in this case Power).

To further illustrate the approach let's look at a more complicated example of an electrical driven water valve that has an indicator light output.

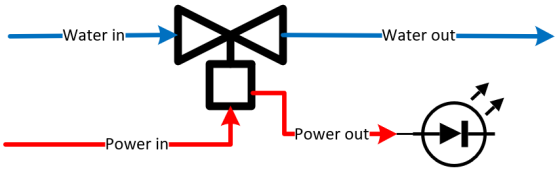


Figure 39 Schematic of a powered water valve with indicator light

In this example if power is provided to the valve, it will let the water pass through. When powered, it can show that state through an LED connected to its power output. If that light output has a short to ground, the valve closes and therefore stops the flow of water.

To gently introduce the resulting Bayes Net, we first show how it looks like without considering upstream behavior in Figure 40.

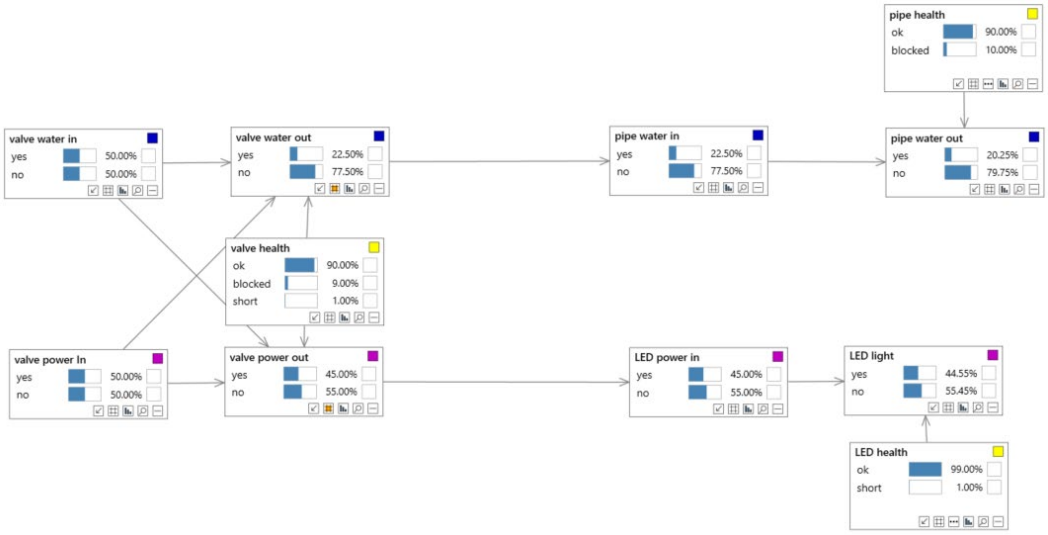


Figure 40 Bayes Net of the electrical valve example without considering upstream behavior

To model the upstream behavior, we connect the upstream properties of the component down the line (water pipe and LED in the example) to

1. All outputs of the previous component
2. The node that captures the upstream effect

Although the resulting Bayes Net looks complicated, it is can be generated automatically according to the rules above. See Figure 41.

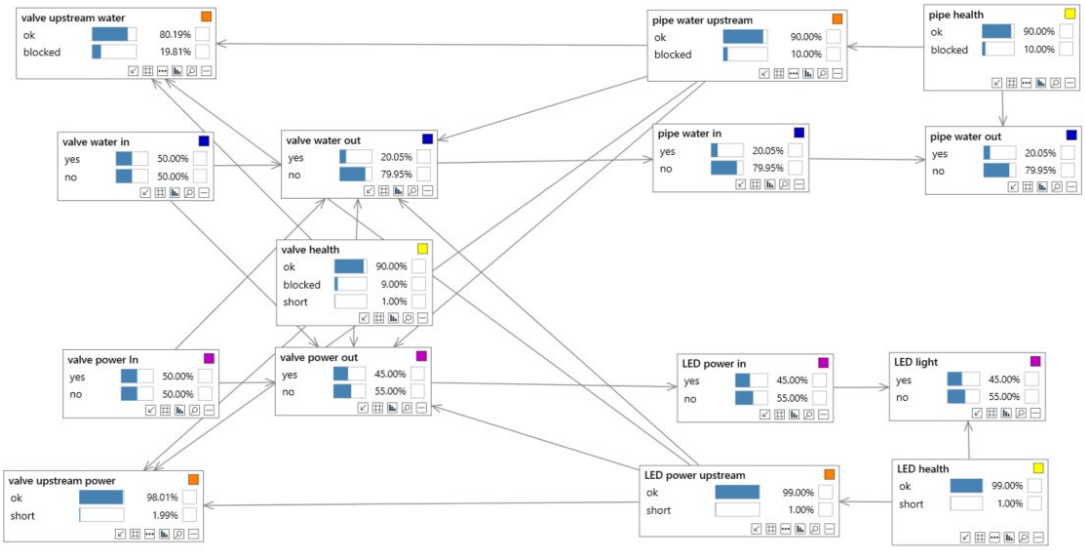


Figure 41 Bayes Net of the electrical valve example including upstream behavior

Note that there is an asymmetry in the upstream effects between water and power: a short in the power circuit causes an upstream water flow block, but a water stream block does not cause an upstream power loss. This is not visible in the picture but is specified in the conditional probability tables. Both scenarios are shown in the figures below.

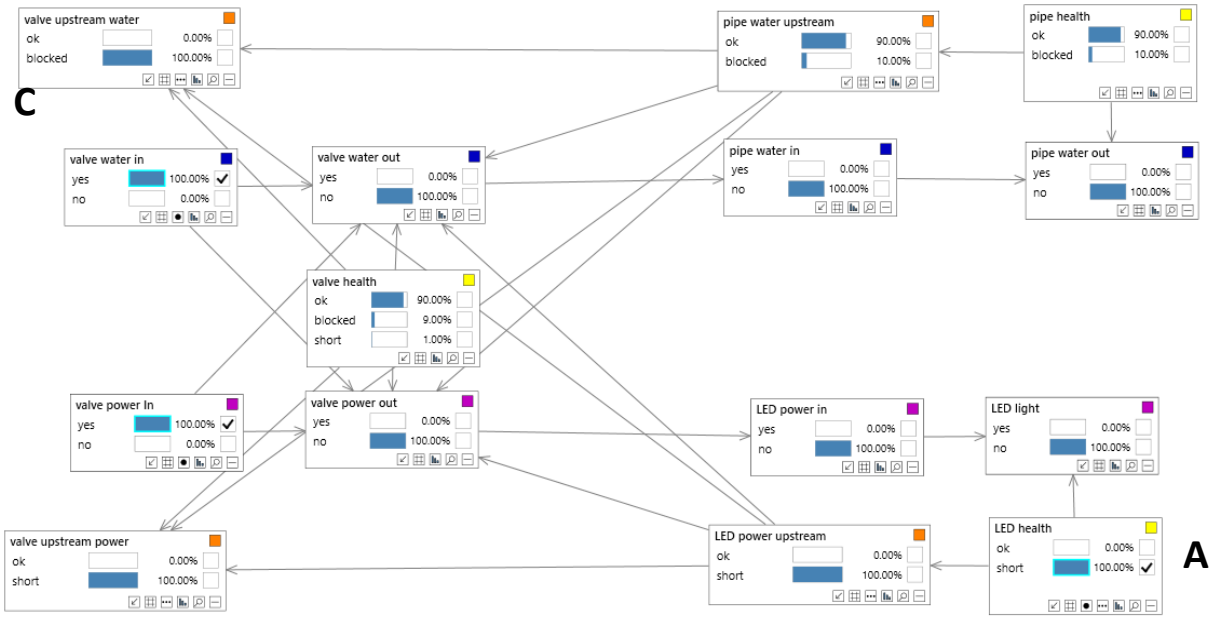


Figure 42 Scenario where a short in the LED (A) leads to a no water out situation (B) and a water blockage effect upstream (C)

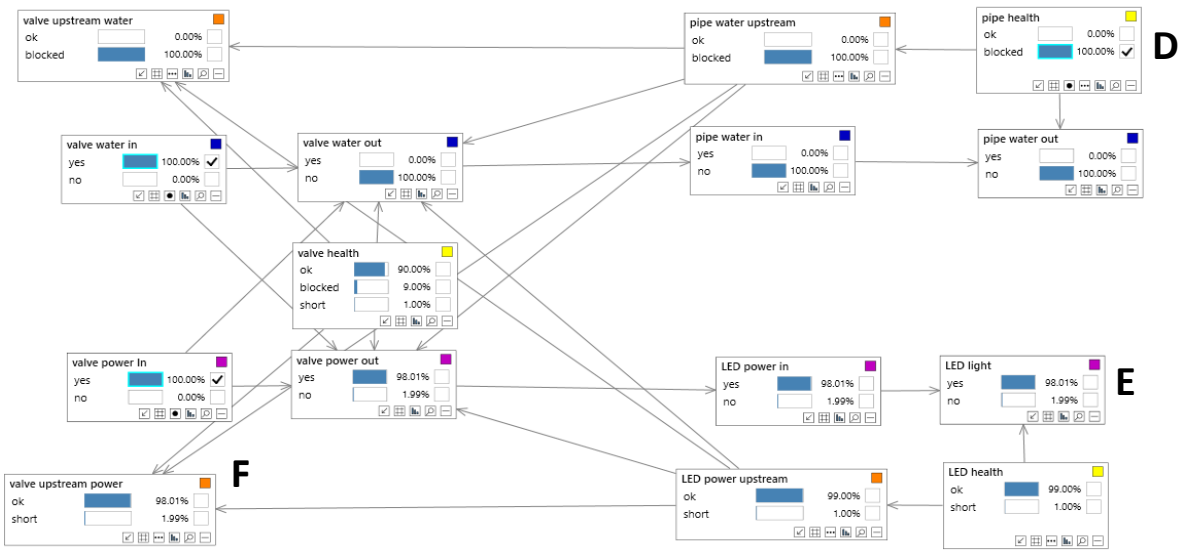


Figure 43 In contrast to the previous scenario, a blocked pipe (D) does not cause the LED to go out (E) nor cause an upstream power failure (F). Although the figure looks symmetrical, the behavior as specified in the conditional probability tables is not.

5.8 Fuses, interlocks and other safety related components

Fuses, interlocks and other safety related components need special consideration. Let us take a fuse as an example.

5.8.1 Multi-step diagnosis

When diagnosing a system without power, the cause might be a fuse that is blown. However, this is probably not the root-cause of the failure: replacing the fuse is not the solution. Although the fuse explains why several components are without power, we are looking for an explanation of why the fuse was blown. The fuse is both a cause *and* an effect. Root-cause analysis requires a “second why”-step.

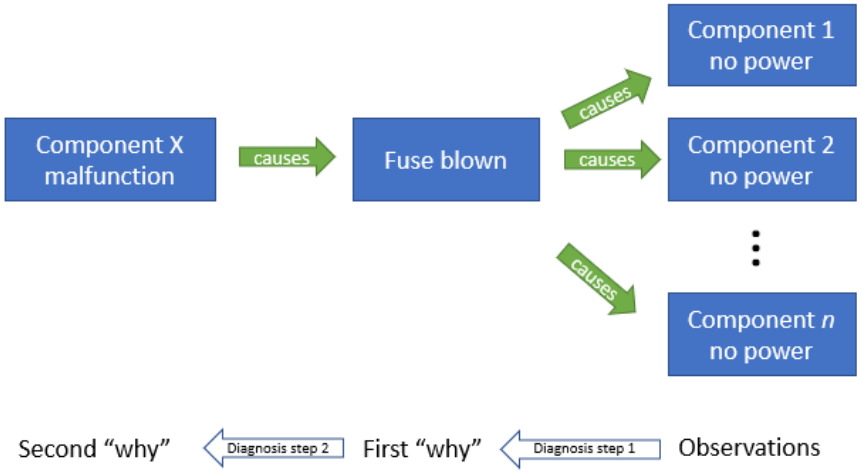


Figure 44 Diagnosing root cause where a safety component licked in is a two-step approach.

The general approach to finding the second why in these cases is to detach components from the circuit and test. This is an intervention (system change, then observe). Interventions are subject of section 5.10.

5.8.2 Bayesian Belief Network

The Bayesian Belief Network with two identical components connected to a single fuse looks like Figure 45 (simplified, wires are not modelled).

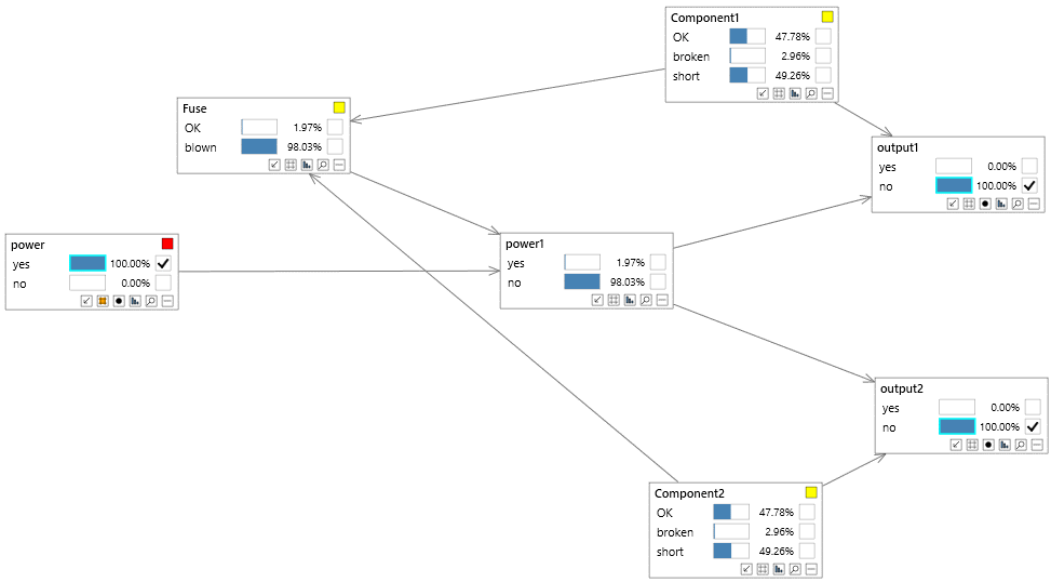


Figure 45 Bayes net of two components connected to a single fuse. Depicted is the diagnostic scenario in which power is provided but neither component produces output. See text for discussion.

If power is provided but neither component produces output, the most likely "component" that has failed is the Fuse (the alternative that both Components are broken at the same time is possible but much less likely).

When the Fuse is verified to have blown, the next diagnostic step is to identify which component has caused the short in the first place. As stated before, this will typically require an intervention such as disconnecting one component and trying to powerup the system with a fresh fuse.

Note the similarity with the upstream effect discussed in the previous section. The difference is that here we regard the Fuse as a component, be it that its "failure" is not independent but rather fully dependent on the components it protects[2].

5.8.3 Conclusion

We can handle fuses and other safety related components using the approach we developed so far. In practice this means that:

1. System designers can design their system in a natural way
2. The Bayes Net generating algorithm has to treat these components in a special way

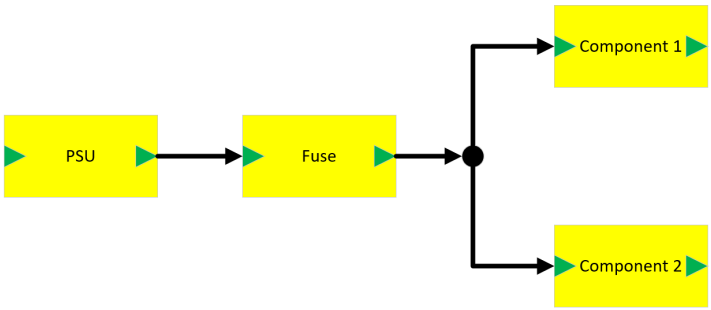


Figure 46 System specification of an electrical system of two components protected by a single fuse. The specification resembles the system structure.

5.9 Modeling software

The diagnosis task is not to identify a particular bug in the software. We assume the software is correct and does not deteriorate over time. What we do need is a model of its (normal) behavior in order to diagnose the hardware components.

Up to now this report used small examples to illustrate the basic concepts. To illustrate the modeling of software we will use part of the Paper Input Module as example.

[2] If a fuse or other safety device can fail itself then that would introduce another failure mode. The Bayes net itself would not change.

The printer behavior is not fully defined by its mechanical or electrical components. The control software is an important part too. So, to describe system behavior in order to diagnose when it misbehaves, a model of the software behavior is required.

An example is shown in Figure 47 where a message if the tray is empty is based on the status of two sensors, the Position Up sensor and the Tray Empty sensor. The logic is captured in an *Empty Tray Logic*-component. When signaled, a message is generated if the required *computing resource* is available. This computing resource is provided by a physical component, in this case the CPU of the PBA Board. This models the fact that if the PBA board fails, or has no power, there will be no Tray Empty message.

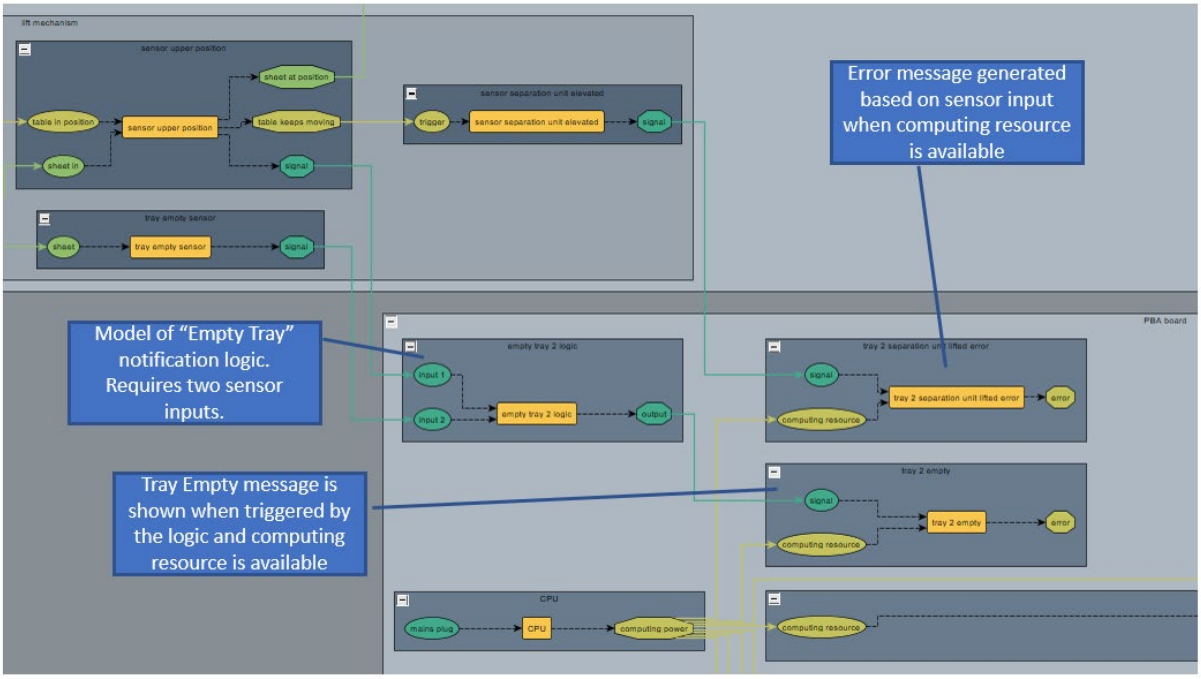


Figure 47 Part of the PIM model showing how the sensors in a tray eventually lead to an error or message on the user interface

5.9.1 Software modeling alternatives

Because a software function does not really have a physical location we have some design freedom of where to locate it in the model. Taking the Tray Empty logic example again, we could take

1. The structural view: the function runs on the PBA-board CPU and therefore belongs to the PBA-board.
2. The functional view: it is part of the Tray (there is one instance for every tray) so it belongs to the tray. Instantiating a PIM with multiple trays would then automatically add the necessary functions.

Although the second view sounds appealing, it has a serious drawback. If the cable between sensor and PBA-board were broken, the first approach would correctly deduct that the Tray Empty message cannot be displayed, while the second view would not capture this.

5.9.2 Software modeling process

Up to now we have modeled the software behavior based on an understanding of the behavior of actual software implementation. This in a way is inefficient as the software itself is a formal description of the behavior. Other approaches could be to either extract the rules from the software code itself (difficult), or tap into the model based software engineering approach (when possible).

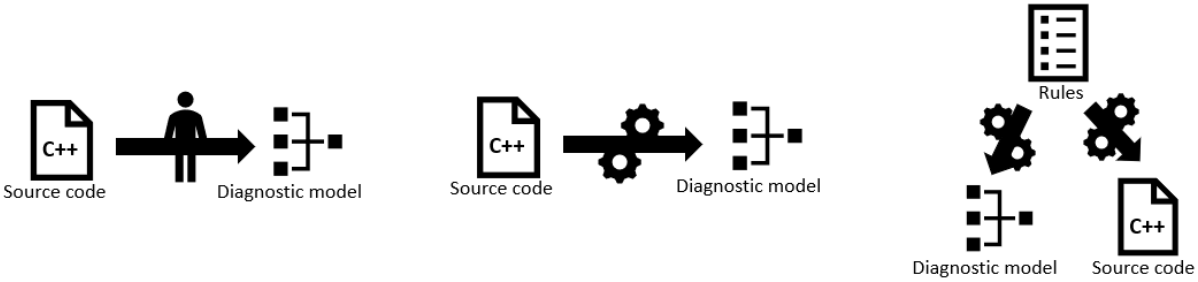


Figure 48 Different ways of capturing software behavior in a diagnostic model. Left: manual analysis; Middle: automatic extraction; Right: model based software engineering

5.10 Interventions

5.10.1 Do-operations

Interventions change the system or the environment in which it operates. Examples include trying a different system function to see whether that works, temporarily disconnecting a connector, and triggering a sensor by hand to see if at least the sensory circuit works as expected.

Evidence obtained after an intervention cannot be treated the same way as we do when merely observing additional facts. This is best illustrated with an example.

Re-using the Fan example of sections 3.2 and 5.2, where we diagnose the fact that there is no wind. The additional *observation* that the motor shaft is rotating indicates that the motor is OK and that power must be present. In contrast, consider the test that we turn the motor shaft by hand (instead of using the motor). If we now observe that there is no wind blowing, we obviously do not gain information on the motor health or power status

Figure 49 Compare the passive observation that the motor shaft is turning (top) versus the active intervention of turning the motor shaft (bottom, with red checkmark). The probability of having power and the health of the motor is different.

The intervention on a node does not change the marginal probabilities of its parents. In the Bayes Server tool, interventions can be entered using the “do(…)” evidence context menu. The terminology is based on the do-calculus coined by Pearl (Pearl, 2000).

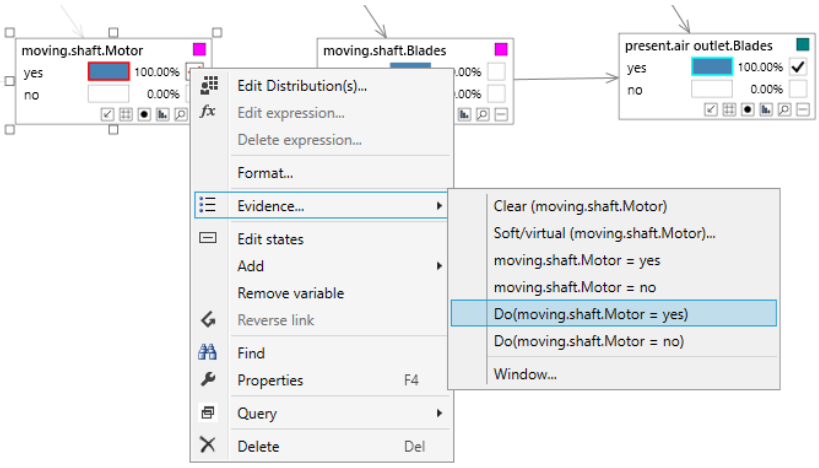


Figure 50 Context menu in Bayes Server to enter interventions

The notation here is *P*(*Health motor* | ***do***(*motor shaft moving*), *Air present*)

If we do an intervention, or possibly multiple interventions, we need to combine the information we have for each of these scenarios because a single scenario only

provides one part of the puzzle. It is important to realize that doing an intervention might change certain observations but we still need to take into account what we observed before the intervention. Therefore, we cannot have just a single instance of node for a certain phenomenon. The solution is to copy large part of the Bayes Net to accommodate for every scenario.

5.10.2 Example

Suppose we have 3 electrical components C1, C2, C3, in a row. All components must be OK for power to come out at the end.

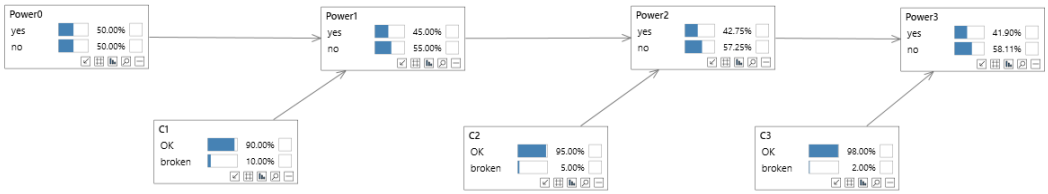


Figure 51 Model of 3 electrical components in series

We know there is power at the input of C1 but we see no power at the output of C3.

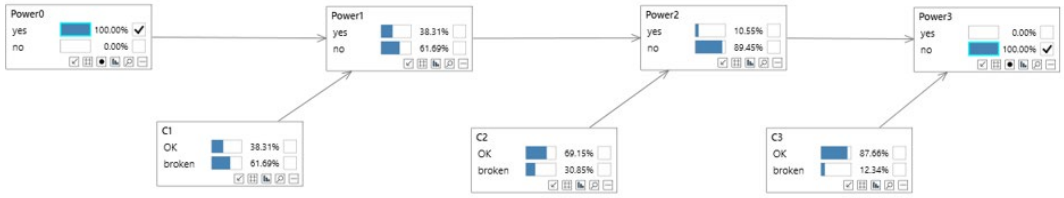


Figure 52 Scenario where there is power at the input but no power at the output

How can we diagnose which component is broken?

5.10.3 Extending the network

Suppose we can test what happens if we apply power (intervention!) to the input of C2 and observe that now we *do* have power at the output. If we merely insert the intervention we get the following result:

Figure 53 Scenario where we insert power in the middle if the network and observe power at the output. Note the red checkmark indicating a do-operation. The incoming arrows are displayed in light gray color because the do-operation effectively blocks the upstream information flow.

We can conclude that C2 and C3 are OK, but the network tells us that the probability of C1 being broken is only 10%. This is incorrect, because from the previous scenario we know that at least one component must be broken, so in this case the only explanation is that C1 is definitively broken.

To have the Bayes Net calculate this correctly we extend the network as shown below. Because in both scenarios (with and without intervention) we are using the same physical components and assume the health of these components do not change between these scenarios, we reuse the health nodes of the components.

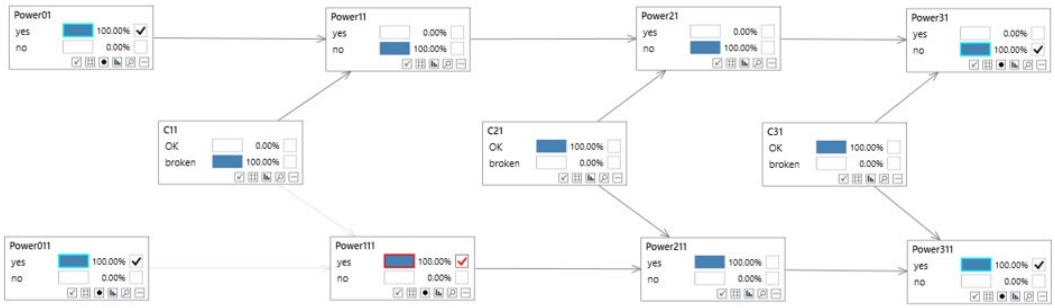


Figure 54 Combing the two scenarios into a single network, with the component healths as connecting nodes

The top row shows the first scenario, the bottom row the second (with intervention). Now, the network takes the information of both scenarios into account and correctly concludes that C1 must be broken.

Note that because we duplicate the nodes we have the possibility to add different observations for the same phenomenon. In this case we can express that we observe no power at the output of C3 in scenario 1 while there is power in scenario 2.

5.10.4 Replacing a component

Another type of intervention is to replace a specific component and then observe the effect. We model this by (again) extending the network but in this case we also "duplicate" the replaced component because it is in fact another item with its own health. In the example we assume that the new component is 100% OK but this is not required for the method to work.

Building on the previous example, assume that the first intervention of adding power was not successful:

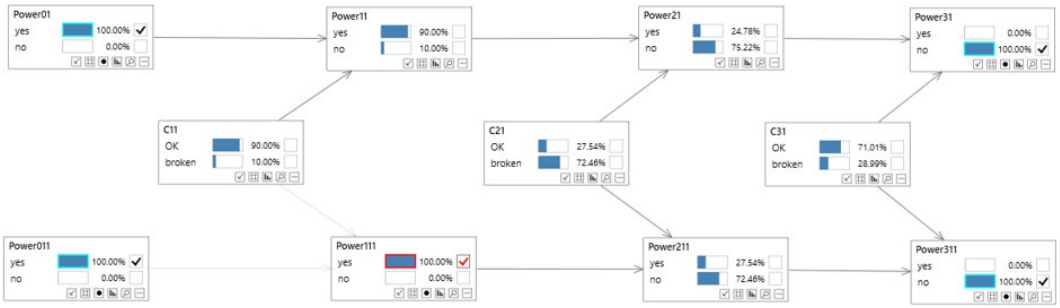


Figure 55 Alternative scenario compared to Figure 55. Here we observe no power at the output after inserting power.

We now have a high suspicion that either C2 or C3 is broken. Let's see what happens if we replace C3. See Figure 56.

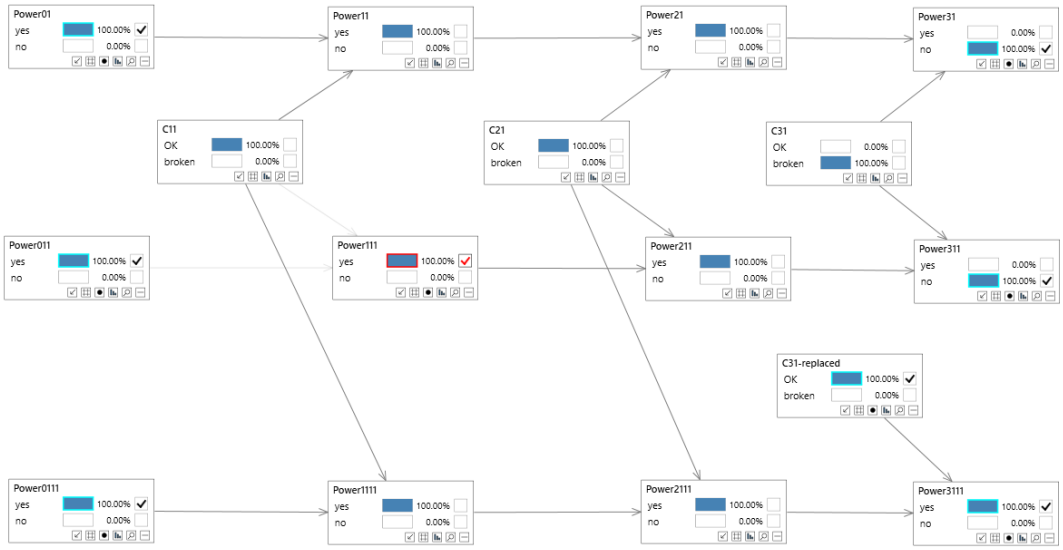


Figure 56 Adding information that replacing C3 solved the problem. The network concludes that C3 must have been the problem and that C1 and C2 are definitively OK.

In this scenario we do have power at the output so the network correctly infers that C3 must have been broken and that C1 and C2 are OK.

Should we still have no power, as in the next figure, the most likely cause is C2 being broken.

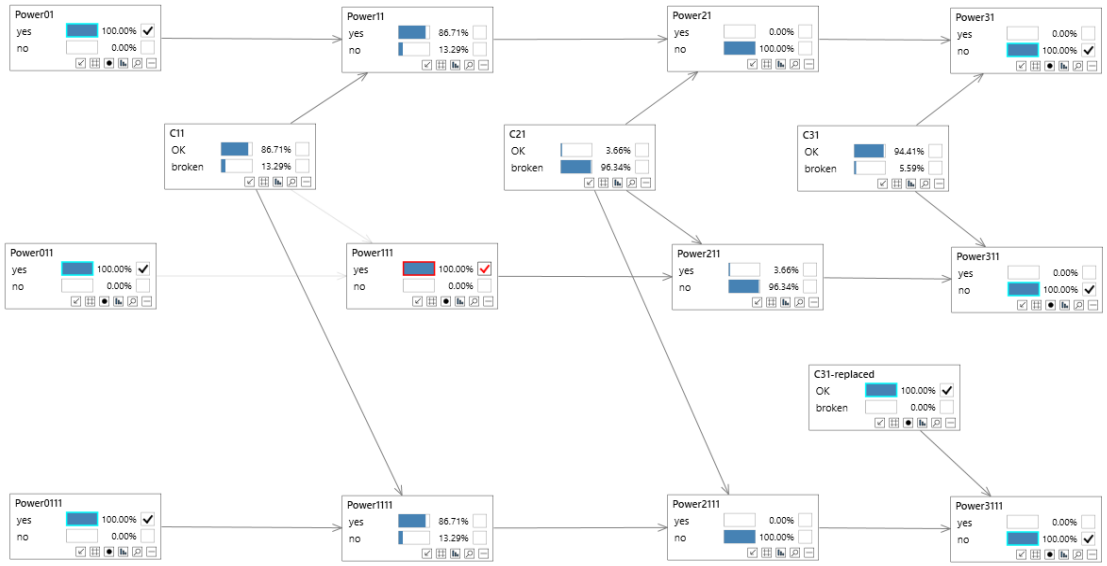


Figure 57 Network when replacing C3 still did not solve the problem. Now C2 is by far the most likely candidate. For discussion why see text in the next section.

5.10.5 Multiple faults

Note that we are not 100% sure because it could be, although less likely, that both C1 *and* C3 are broken. This information cannot be understood immediately from the picture above but can be obtained by looking at the joint probability distribution $P(C1, C2, C3)$ as shown in Table 8.

C11	C21	C31 = OK	C31 = broken
OK	OK	0	0
OK	broken	0,84971	0,017341
broken	OK	0	0,036609
broken	broken	0,094412	0,0019268

Table 8 Joint probability distribution over the three components after at the end of three scenarios. Example: the probability that C1 and C3 are broken and C2 is OK, is 0.036609 (last column, second row from below).

5.10.6 Calculating the best intervention to do

Not only do we need to assess the information after an intervention, we also have to determine which intervention is the best to do. This topic is addressed in section 6.8.

5.10.7 System inputs

In the previous sections we have not discussed if we should duplicate the nodes that represent system inputs or not. The answer is: it depends.

1) If we do not observe the inputs they are effectively part of the root cause analysis (the system does not work because there is no power). In this case we do not duplicate the input nodes, assuming that the input does not change between interventions, just as we assume the component healths do not change.

2) If we set an input (as an intervention) we duplicate the node, and set the evidence accordingly using the "do-action"[3].

3) If we observe an input, we do not duplicate the node and set the evidence. This assumes that the input did not change between interventions. Even if we observe an input in a later scenario, adding it as evidence then also influences the beliefs in earlier scenarios.

5.10.8 Alternative: Posteriors as the new priors

Instead of extending the network we also considered an alternative approach: for a new scenario we insert the posterior probabilities (the new beliefs after the previous scenarios) as the prior probabilities. Intuitively this makes sense. E.g. when we concluded in scenario 1 component C1 is OK we fix its state to being OK in the next scenario. However, we encountered several cases in which this approach yields the wrong outcome. See for example the next diagnostic scenario.

We have an all-in-one printer that can copy and print. Copying requires a scanner component, printing requires an image processing component and both functions require an ink jetter component.

[3] Since by definition an input has no parents nodes, using do-action or setting regular observational evidence is actually computational equivalent.

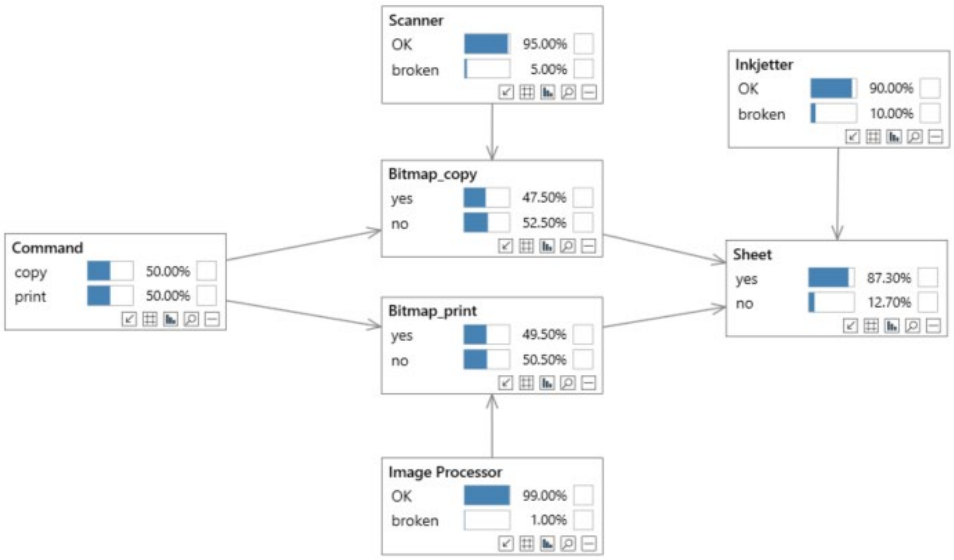


Figure 58 Modeling an all-in-one printer that can copy and print

Suppose we see that copying does not work.

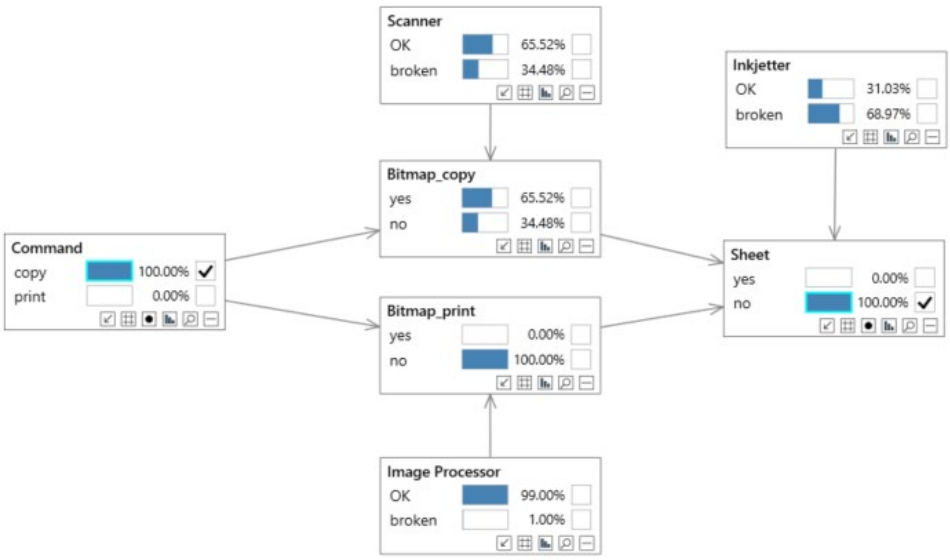


Figure 59 Scenario where we see that copying does not produce a sheet. Conclusion is that either Scanner or Ink jetter are likely failing

To further diagnose, we try the printing functionality. Suppose that printing succeeds. We model that by carrying over the posteriors of the previous scenario and add evidence that printing succeeds.

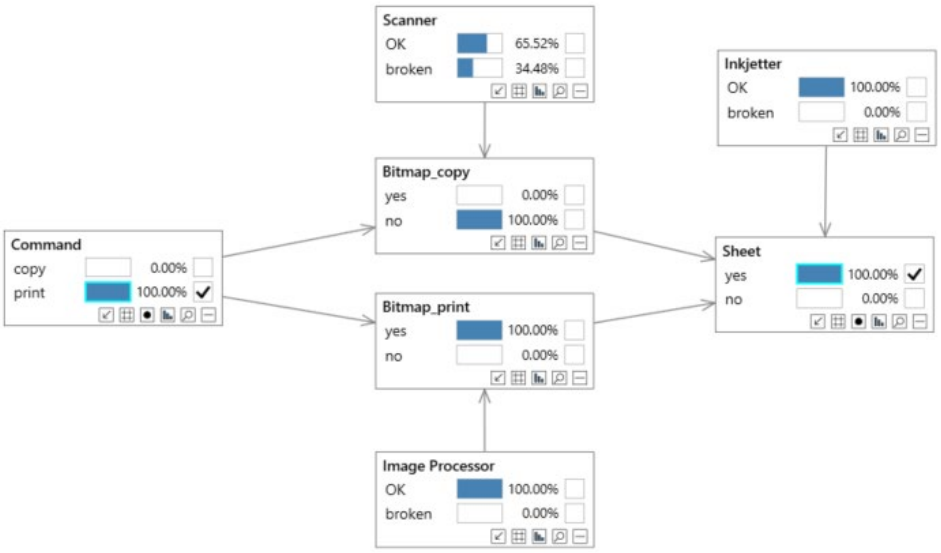


Figure 60 Second scenario where printing succeeds. The Scanner health probability is still at the probability from the previous scenario. Apparently, not all information from the previous is carried over as we know the Scanner must be failing.

The network correctly infers that the image processor and ink jetter are OK but fails to conclude that the scanner *must* be the failing component.

This example shows that although using the posteriors as the new priors transfers knowledge from one scenario to the next, it does not transfer *all* available knowledge.

5.10.9 Implementation consequences

Supporting interventions requires a change in the tool chain: instead of pre-computing a single Bayes Net that is used at diagnostics time, we now have to generate new Bayes Nets on the fly as interventions are planned and executed.

5.11 Diagnosing wrong inputs

5.11.1 Problem statement

In all examples up to now we assumed that all system inputs were known and are correct. The only task left is then to diagnose a failing component.

In reality not all inputs are known and the reason for a system failure could be a wrong input. The most obvious example is that there is simply no power provided to the system, but in complex systems it is not at all obvious which inputs are required for a specific system function.

5.11.2 Solution

As a small example consider a water supply system that can provide hot or cold water. Opening the valve will give water (if water is supplied) and pressing the switch will activate the heater (if power is supplied).

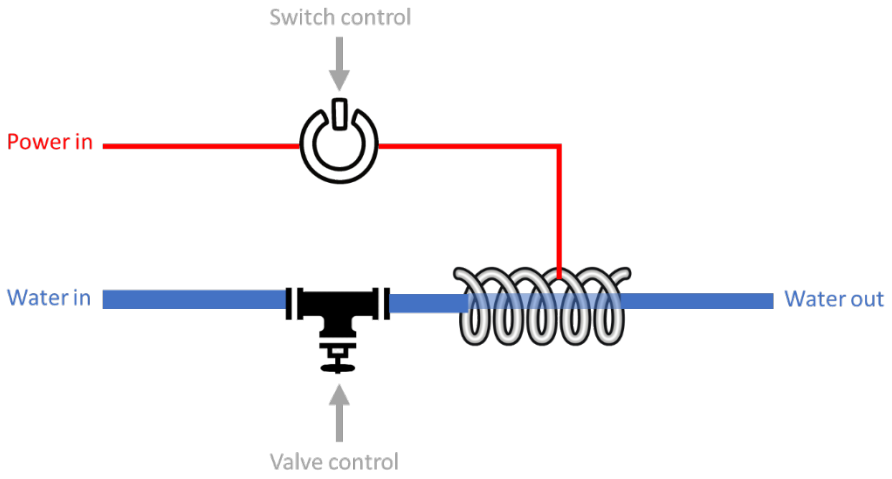


Figure 61 Schematic of a system that can supply water, either hot or cold, depending on user control settings

For model simplicity we leave out failing components and focus on the inputs.

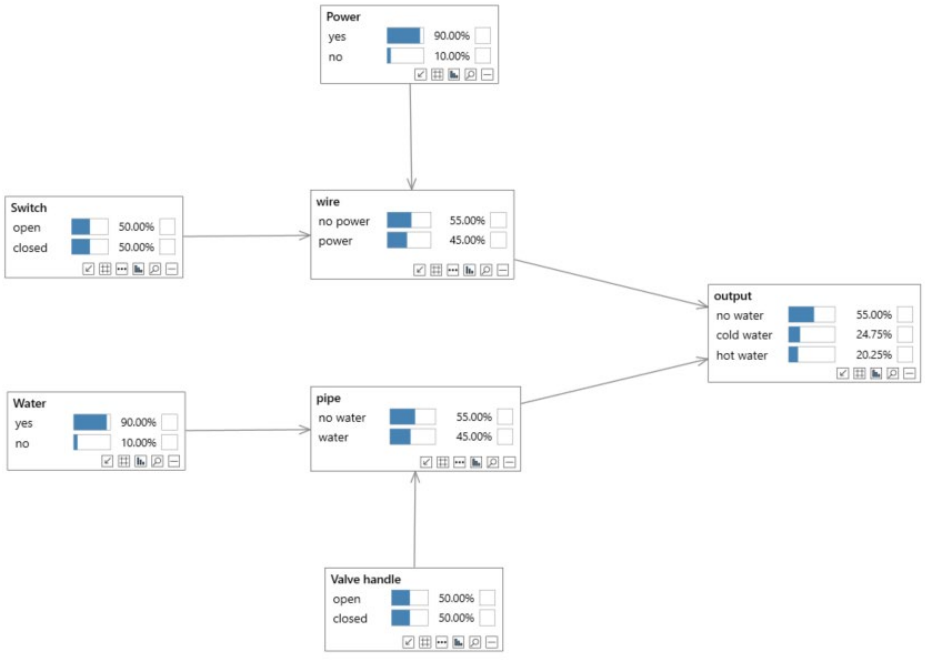


Figure 62 Bayesian Belief Network modeling the water supply system

If we observe that there is cold water coming out, the diagnosis depends on our expectation:

1. If we expected no water, then we need to check the valve knob position
2. If we expected hot water, then we need to check the switch position as well as the power input.

Thus, the “correct” diagnosis suggestion depends on our *expectation*.

To handle this in our diagnostic framework we make the users’ expectation on the system explicit. First we set the expectation and then we provide the actual situation. Then a diagnosis corresponds to the comparison between the actual situation and the expected situation. This comparison can be done in the BN by first setting the base evidence (expectation) and subsequently calculating for every node either the difference in probability or the quotient (called *lift*) given the actual evidence.

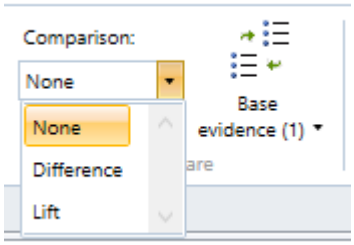


Figure 63 Setting base evidence and calculating deviation from the base by either Difference or Lift in the Bayes Server tool.

Scenario 1: we expect no water, but we see cold water

Step 1: setting expected situation by indicating no water at the output and capturing the results as base evidence.

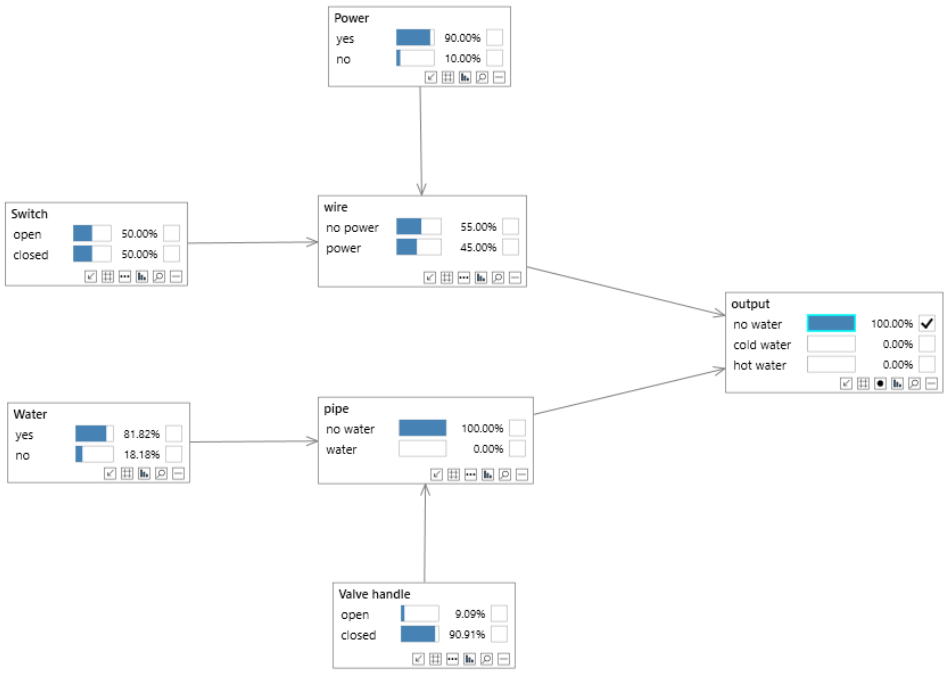


Figure 64 Setting the expectation "no water" on the output. This is then set as base evidence.

Step 2: setting cold water as observation and calculating the lift.

Figure 65 Setting the observation "cold water" and calculate the lift (quotient) compared to the previously set base evidence. Lifts larger than 1 (increase) are indicated with a yellow upward pointing arrow.

We see that the lift (indicated by the arrows in Figure 65) is very high on the Valve handle input. That is the input we should check first.

Scenario 2: we expect hot water, but we see cold water

Step 1: setting the expectation by indicating hot water at the output and capturing the results as base evidence.

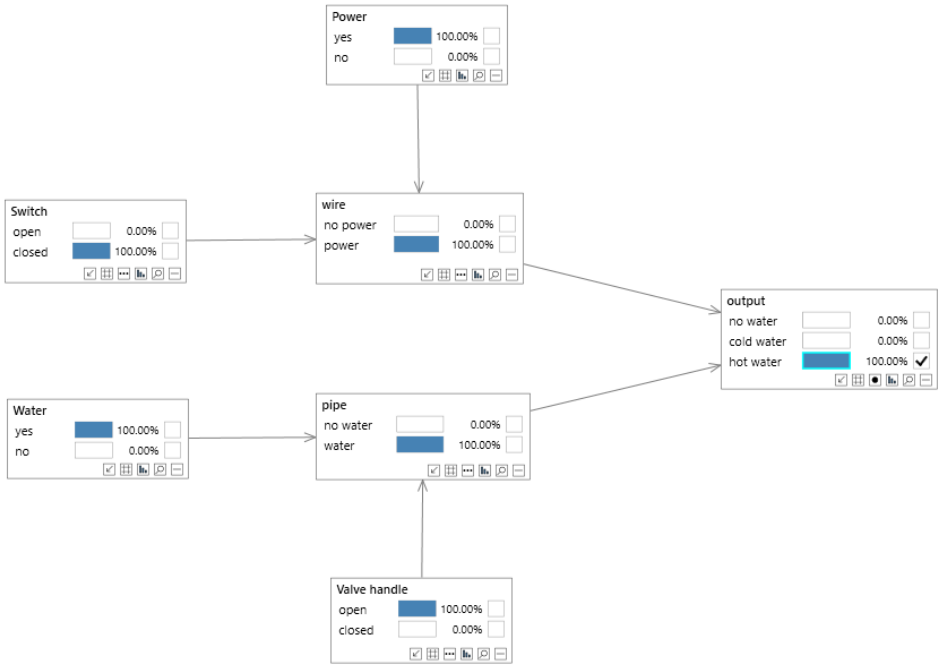


Figure 66 Setting the expectation "how water" on the output. This is set as base evidence.

Step 2: setting cold water as observation and calculating the lift.

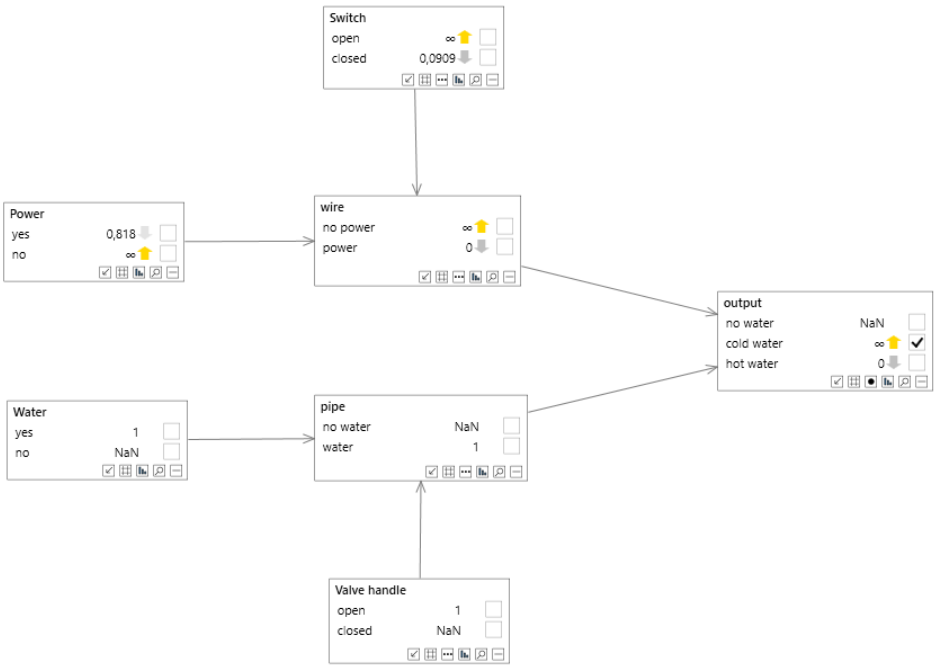


Figure 67 Setting the observation "cold water" and calculating the lift. In contrast to scenario 1, now we see that lift is very high in the power section of the model: either the switch is open or the input power is not available.

Now we see that the lift is very high on the power switch and on the power input. These are the items we need to check first. Although the evidence (cold water) is the same in both scenarios, the outcomes are different because the expectation was different.

5.11.3 Conclusion

By setting the expectation and calculating the difference or lift compared to the base situation we can diagnose wrong or unknown inputs. This is not much different from diagnosing component failures except that in that case we have the implicit expectation that components are working correctly.

For inputs it is not obvious what is the right and what is the wrong input, as that depends on the expected system function, and therefor has to be set explicitly. This could be done by either explicitly indicate the correct value for every input, or as we have shown in this section, to only set the expected outcome, and have the model calculate the corresponding expected inputs.

This approach is not yet implemented in our diagnostics software.

6 Service tool

The service tool is created from the model and supports the service engineer in their diagnostic task.

6.1 Workflow

The workflow would be like this:

1. Optionally, the tool reads all available machine data and adds these observations as evidence in the model
2. Service engineer selects the problem (observation as evidence, e.g. an error code)
3. Tool calculates:
    a. Which components are most likely to have failed, including
        i. Confidence in this outcome
        ii. Explains why these components are suspected to have failed
    b. Which inputs are important to know for a good diagnosis
    c. Which tests are the most valuable to do to get a better or more specific diagnosis
4. Service engineer selects and executes a test and enters the outcome of the test in the tool
5. Go to step 3, until the engineer decides to do a service action.

To illustrate these steps we developed a web-based prototype. See Figure 68.

Figure 68 User interface showing the capabilities of the diagnostic tool for the service engineer. Starting from the error message or at the top, the tool lists suspected components and suggests tests for further diagnoses. Evidence entered is shown at the bottom

This chapter addresses the steps mentioned above in more detail.

6.2 Connecting to data

Machine data that is available can be automatically entered as evidence in the Bayesian belief network. This supplies the tool with as much information as possible that is available “for free”. Data can be sensor data or event data from the event logs.

When setting up such an automated data pipeline there are two potential pitfalls:

Event processing

To correctly interpret the events logged in an event file it is important to understand the event generating mechanism. As an example, a system process might log its starts and ends with a JobStart and JobEnd events. However, if the last job event in the event log is a JobStart, this does not mean that the system is running a job. It could be that the system stopped processing the job due to an error.

Sample frequency

Typically, sensors are sampled with a certain sample frequency. This means that at any point in time you actually don’t know the current situation, only the situation

at the last measurement. This can lead to a situation in which an effect can be observed *before* the cause is observed. In a naïve modeling approach, this will also lead to inconsistent evidence or wrong diagnosis.

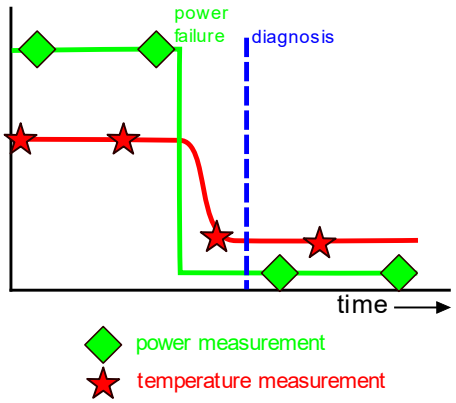


Figure 69 When doing diagnosis at the time indicated by the dashed line, we would conclude that the temperature is low while the power is on (last known measurements), indicating a heater problem. However, it could be that the temperature is low because the power went off and therefore not a heater failure.

6.3 Observations and inputs

Typically, a diagnosis starts with the observed problem, such as an error message. This observation is entered into the tool. Other “known” facts are also entered to get the best possible diagnosis. This includes the model inputs (for systems typically things like power supply: if the power plug is not connected to the wall socket or the building electricity supply fails then the system will not work). If not provided and relevant for the diagnoses, the test recommendation (see below) will include a request to observe an input (“is the plug connected to the wall socket?”).

6.4 Hypotheses and explanation

At any point in time the service tool will list the most likely hypotheses for the root cause. Showing multiple hypotheses (especially when their likelihoods are similar) helps in understanding “what the tool is thinking”, can immediately trigger the service engineer for either positive or negative conclusions, and helps in understanding why certain tests are recommended (see next section).

In addition to a list of hypotheses the tool also shows the certainty of these hypotheses. In the beginning of a diagnosis session the certainty will be low and then gradually increase as the session progresses. How the uncertainty should be conveyed to the user is still subject of research.

Providing an explanation with the failure hypotheses will

1) increase the trust of the engineer in the tool
2) help the engineer in selecting the best next action
3) educate the engineer while doing diagnosis.

This is still subject to research on how to do this effectively. Current prototypes do not yet have the explainability feature.

6.5 Test recommendation

6.5.1 Types of test

To further pinpoint the root cause, additional diagnosis steps are proposed.

These steps include:

- Add additional observations (probing)
- Make a change in the system and observe the result (intervention)

6.5.2 Value and cost of a test

The tool recommends the best actions to take based on its diagnostic value (how much will the test result reduce the uncertainty in the diagnosis?) as well as the amount of effort required to execute a test (checking an indicator light might be much easier than using measurement equipment, although the latter could provide more detailed information).

Technically, the reduction of uncertainty is based on some information theoretic value such as the value of information of potential new evidence or entropy reduction over a set of hypothesis. The amount of effort is encoded in a *cost*. The tool will then determine the tests with high value of information and low cost. An obvious way to include cost is to simply divide the expected entropy loss by the cost, which would give us per test candidate the number of bits per Euro to rank them but there might be better alternatives. This is part of the ongoing research.

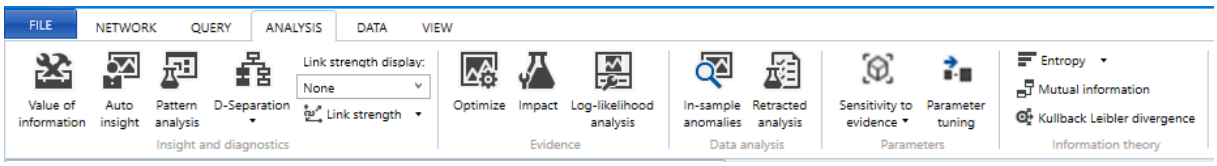


Figure 70 Bayesian network tools provide analysis of networks, such as Value of Information (left) and Entropy (right). Picture from Bayes Server.

As with the hypotheses, the tool recommends multiple actions sorted according to its effectiveness for diagnoses.

6.5.3 Decision networks

As an alternative to using entropy or other information-theoretic metric to calculate the best test to do, we also investigated the use of decision networks for this purpose.

Decision networks are an extension to Bayesian Belief Networks by adding decision nodes and utility nodes in addition to the random variable nodes.

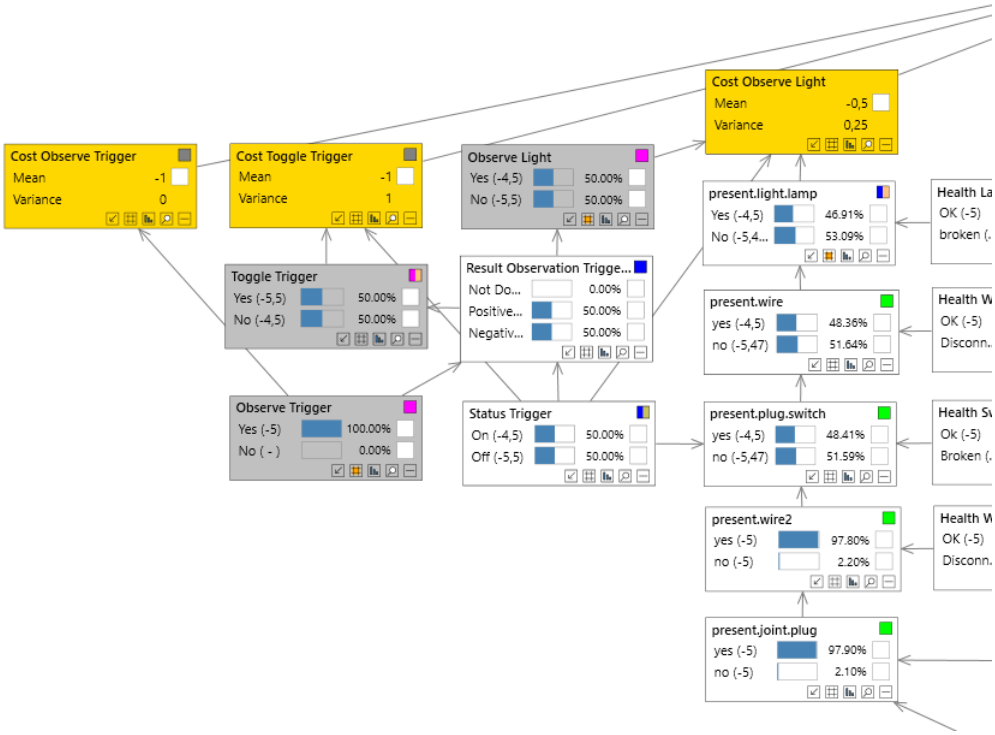


Figure 71 Part of a decision network in the Bayes Server tool. The yellow nodes are utility nodes representing costs and the dark gray nodes represent decisions. Every decision is associated with a cost. Example taken from [8]

As we found out, using decision networks quickly runs into practical issues of memory consumption when scaling up model size. We experimented with approximations to circumvent these obstructions [8] but have not yet arrived at a satisfactory solution. Further research on this topic is planned for the near future.

6.6 User Interface

We have described the capabilities of the service tool and built a prototype showing the principles of the tool but have not designed an actual interface for use by the service engineer as such a tool should fit in the system manufacturers service ecosystem.

6.7 Prototype implementation

In our prototype implementation, we implemented a strategy that shows the principles but definitively needs refinement towards a real tool.

All the nodes in the Bayesian network that drive the tool are divided into 3 categories:

1. Inputs and observations
2. Capabilities and functions
3. Components, including connections

Given all the evidence we calculate the posterior probability for every state of every node, assume that the Normal state (or correct behavior) is the first state in the network, and sort the nodes per category. The probability displayed is the sum of all abnormal state probabilities, or 1 minus the Normal state probability.

The tests to recommend are calculated as follows:

- All components that have a failure probability larger than 0.2 are considered potential failures
- For every available capability/function (category 2), we calculate the entropy of the set of potential failures, given the capability.
- The top *N* capabilities that have the lowest entropy (intuitively: that gives the most information among the set of potential failures) are recommended as tests.
- To show the test when a user clicks the clipboard icon, we have manually connected the capability to a test description in a csv file.

6.8 Intervention prototype

The prototype described above only implicitly deals with interventions. In particular, it assumes that a function can be tested without explicitly modeling what this test entails. This approach does not work when using a more structural oriented model. There we need to specify what the intervention actually is and what we then physically observe.

The intervention-aware prototype is depicted in Figure 72.

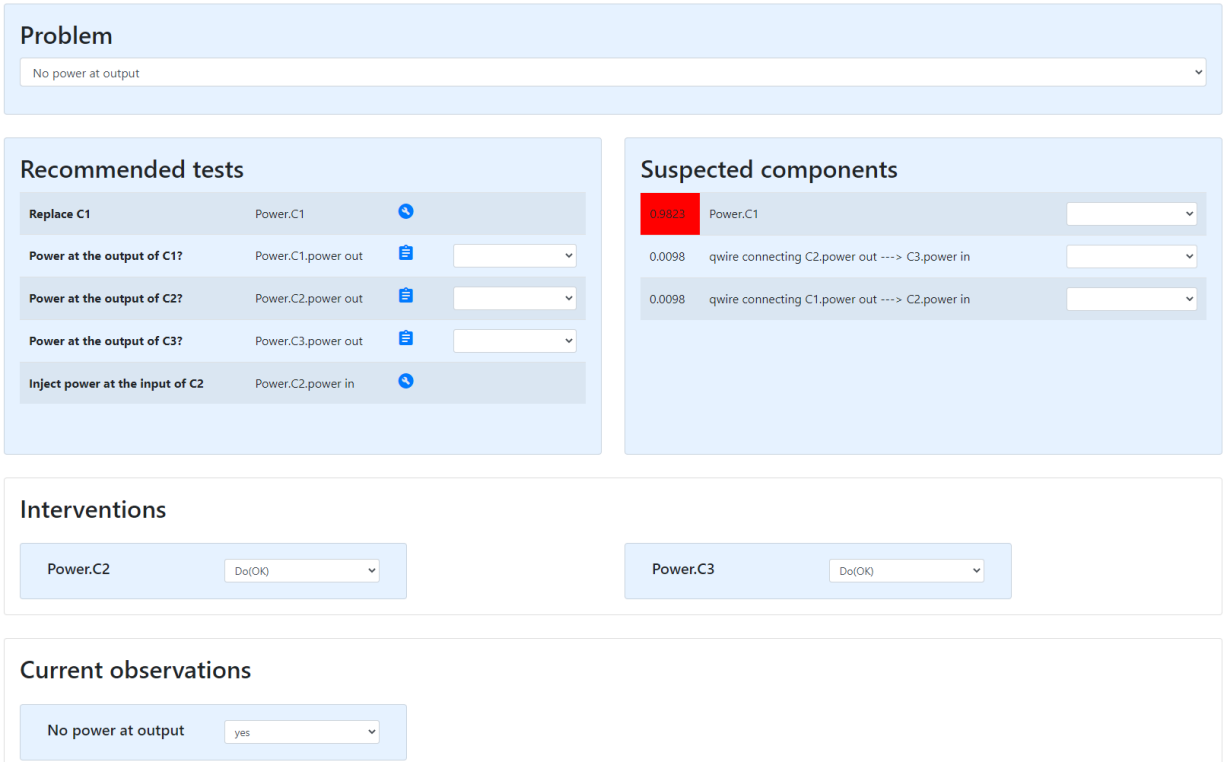


Figure 72 User interface prototype supporting interventions. the section "Current observations" refers to situation in which components C2 and C3 have been replaced as indicated in the Interventions section.

We augmented the earlier prototype with the following elements:

1) In the list of tests we specifically indicate interventions and an associated intervention.
2) We show in the left columns different icons for interventions (wrench) and probing tests (clipboard).
3) To calculate the expected information gain after intervention we extend the network and calculate the entropy decrease given the observation associated with the intervention.
4) Previous interventions are shown in the new Interventions segment of the user interface.
5) If there was already an intervention done, the question is, is that intervention still applicable. Currently (end 2022) we deal with this in the following way but this is subject to further research:
    a. To calculate the expected value of an intervention we calculate the maximum value over all possible combinations of earlier interventions.
    b. When an intervention is done, the user is asked whether earlier interventions are still applicable. Previous interventions turn yellow until the user has either confirmed or rejected them. Only

then the probability distribution over the health variables is recalculated.

7 Conclusion

In this report we have outlined an approach towards faster diagnostics.

It includes

1. A method to systematically describe system normal and abnormal behavior;
2. A way to rank root-cause hypotheses, using a generated Bayesian Belief Network;
3. An outline to recommended the most efficient next step in a diagnosis task;
4. A way to methodically reason with interventions, i.e. dealing with multiple scenario's;
5. Suggestions for a service engineer facing user interface.

The method mentioned in 1. is not only beneficial for root-cause analysis but also:

- Supports design for diagnosability;
- Streamlines the FMEA process.

8 References

[1] P. Kruizinga, E. van Gerwen and L. Tealdi, "Guided root cause analysis of machine failures - the Carefree project," Canon Production Printing, 2021.

[2] L. Barbini, "Canon Performance Diagnosis, *to be published*, 2023.

[3] L. Barbini, "Approaches to performance diagnostics and prediction with Bayesian networks", *to be published*, 2023.

[4] Korteling, Wouter, "Carefree 2022 – Using data science to relate media properties to printer errors," TNO-ESI 2022-10158 , 2022.

[5] L. Barbini, C. Bratosin and E. van Gerwen, "Model Based diagnosis in complex industrial systems: a methodology," in *PHM Society European Conference, 2020*, 2020.

[6] E. van Gerwen, T. Nägele and L. Barbini, "Integrating system failure diagnostics into model-based system engineering," *INCOSE Insight,* 2022.

[7] L. Tealdi, C. Grappiolo and M. Borth, "Carefree 2020 - Data-based Approach," TNO-ESI 2020-10188, 2020.

[8] J. Beurskens, "Sequential Testing for the Diagnosis of High-tech Systems," https://research.tue.nl/en/studentTheses/sequential-testing-for-the-diagnosis-of-high-tech-systems, 2022.

[9] J. Pearl, Probabilistic Reasoning in Intelligent Systems - Networks of Plausable Inference, Morgan Kaufmann, 1988.

[10] J. Pearl, Causality: Models, Reasoning and Inference, Cambridge University Press, 2000.

A. Bayesian Belief Network tools

There are several tools available that support Bayesian Belief Networks. They typically have a computation engine with API to include in custom made software, as well as a GUI for experimenting and validation. All tools have a trial version with limitations.

There are slight differences in the capabilities of the different tools, but all support dynamic networks, continuous variables, and decision networks to some degree.

Tools include

Tool	Website	Remarks
Hugin	www.hugin.com	Expensive. Old-fashioned GUI.
Netica	www.norsys.com	Old-fashioned GUI. Very good tutorial section to get started.
Bayes Server	www.bayesserver.com	Easy connection to databases. Examples in this report are created using this tool.
Bayes Fusion	www.bayesfusion.com	Also has web-based query interface.

There are also free libraries available in various programming languages. Implementing the inference algorithms is not an easy task as dealing with thousands of small numbers requires attention to efficiency and numeric stability.

B. Upstream effect handling alternatives

In section 5.7 we discussed the handling of upstream effects in the Bayes net generation. There are alternatives and variations to that approach that we considered during development. These alternatives are listed here for completeness.

All these alternatives have one characteristic in common: the Bayes Net generating algorithm needs to accommodate for exceptions based on specific failure modes and components. In these examples the short to ground failure mode in combination with wires and other components that handle power there is an exception to the otherwise general Bayes Net generating rules. As a consequence, every time a new power related component is added to the library, the Bayes Net generating algorithms needs to be updated.

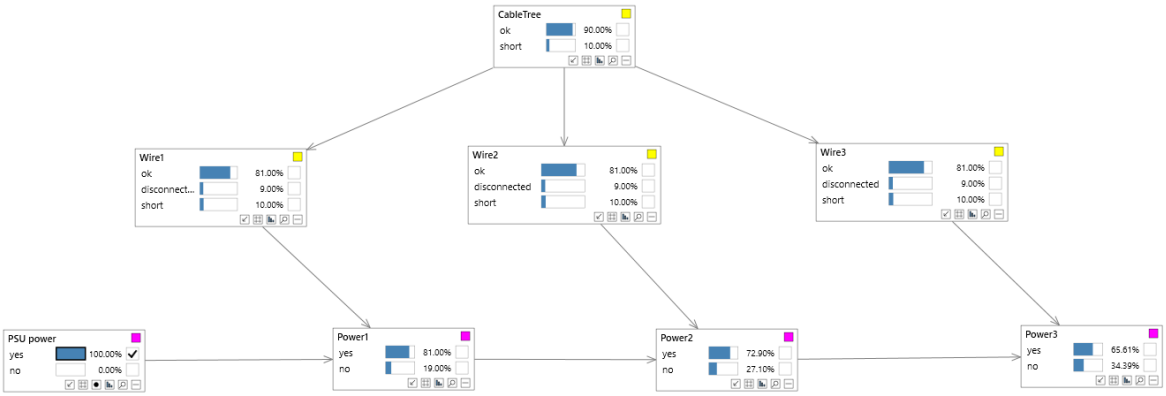
The solution described in the main text requires a more elaborate component definition but does not require changes to the Bayes Net generating algorithm.

For clarity we reduce the size of the example so we can simplify the Bayes nets. In particular, we leave out the WireJoints and consider a cable of several segments.

Figure 73 Simplified model of 3 wires in series

Alternative 1 — Top node

Basic idea is to introduce a cable tree node representing the state short-to-ground of the cable tree We then model that a wire has a short-to-ground if the cable tree has a short to ground.

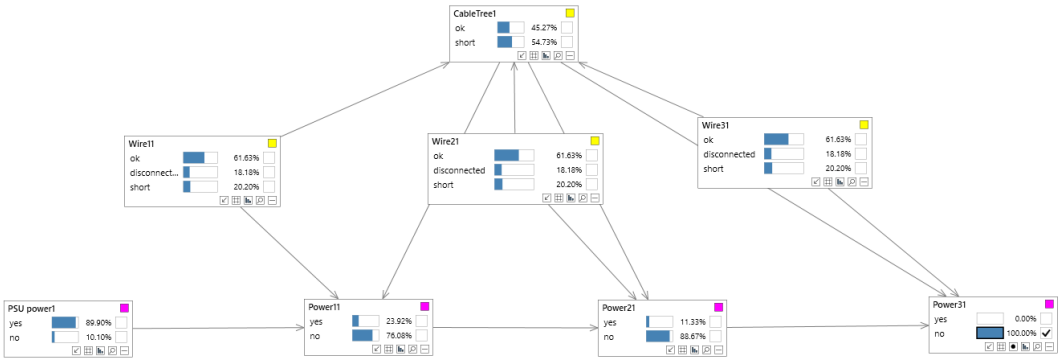


Although this encodes the behavior well some disadvantages are:

1. If a wire is diagnosed as short-to-ground, the cable tree must be short-to-ground and then all other wires in the cable tree are also short-to-ground. This is not what one would like to see when looking at the components individually.
2. If somehow you could assess the state of a single wire as being OK (not short-to-ground), then in this model no other wire could be shorted. Note that in general you cannot find individual wires to be OK without disconnecting them from the system.
3. In this model one has to specify the a-priori probability of the cable tree to short, which would actually be a result of the probability of the individual wires.

Alternative 2 — Reverse causality

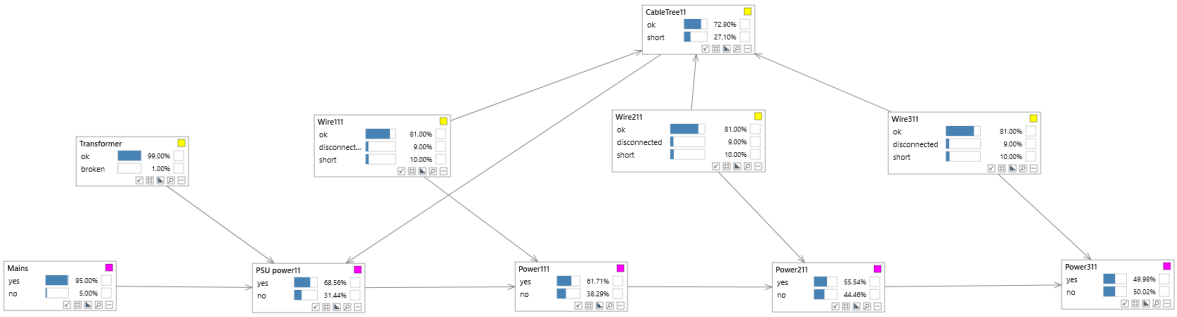
The difference compared to Alternative 1 is that instead of encoding that the cable tree “causes” the wire to short, we encode that a short in the wire causes the cable tree to short, which in turn inhibits power to flow.



This model fixes all disadvantages of option 1, at the expense of more complexity of the Bayes net, but since this is generated from the structural description anyway this is not a huge disadvantage.

Alternative 3 — Reduce complexity

In alternative 2, all arrows from the CableTree node to the individual Power nodes are superfluous. If there is no power at the beginning of the cable tree, then no other points on the wire will have power. This is expressed in the following model:

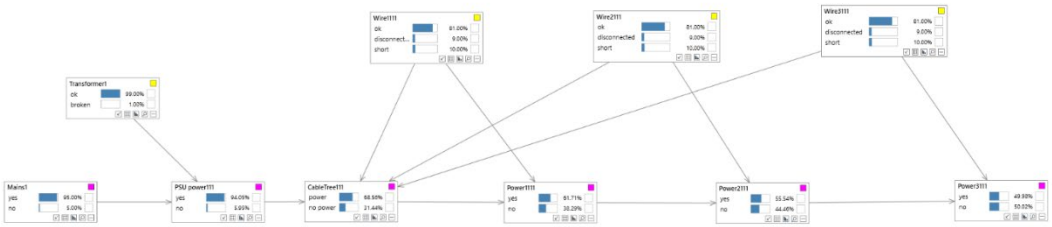


This works fine but has the slight downside that the PSU output node has to change when it is connected to a cable tree: it gets an additional parent. Although technically possible, this feels uneasy as we would like to see the Input-Health-Output triple as a building block of the PSU component. A similar argument can be made if we connect the CableTree to the first Wire output (Power111 in the picture).

Alternative 4 — Add blocking node at the start of the flow

This was previously our proposed solution but has been superseded by the solution described in the main text.

The general idea is to add a node at the start of the flow that blocks downstream flow when an upstream component has a specific failure. The CableTree node in the picture stops the flow whenever one of the wires has a short-to-ground failure.



Alternative 5 — Reduce number of links

To reduce the number of links in the solution as described in Section 5.7, we can move the short-to-ground failure mode from the wires into the new CableTree, effectively making it a component that sits inside the energy flow.

Despite the appeal of simplicity this has two disadvantages:

1. A component node in a physical flow is different from the standard model approach
2. Setting the probability of a short-to-ground must be computed from the individual short-to-ground probabilities of the constituting wires, while these wire nodes do not show a short-to-ground failure mode. This feels messy.

C. Loop handling alternatives

In section **Error! Reference source not found.** we described how loops in the structural model are handled. We investigated some alternatives and these are described here.

Alternative 1 — Cut the loop

The basic idea of "cutting the loop" is to take a node in the loop, duplicate it, and have one copy take all incoming arrows and the other copy provide all outgoing arrows. Then, we instantiate these copies for different observations and for each observation calculate the health for all components. If we repeat this process by duplicating another node in the network, we end up with set of equations we can solve.

Again, this is best illustrated with our house heating example.

The diagnoses scenario is that we know there is fuel available, yet the temperature is low.
In Figure 74 we duplicated the Thermostat node into an a and b copy.

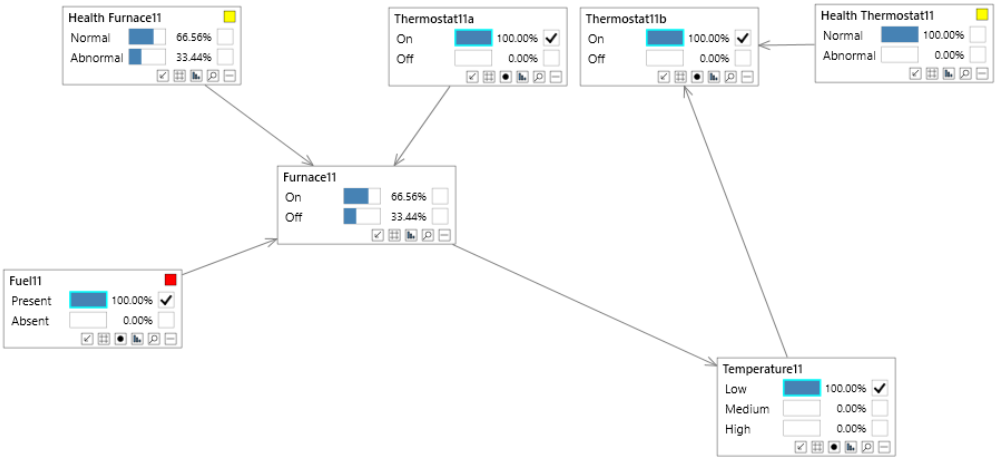


Figure 74 House heating network where the Thermostat node is split into two nodes a and b to break the loop. In addition to the observations, the evidence added here is Thermostat=On. Similarly we compute the marginal probabilities for Thermostat=Off.

From probability theory we know that[4]

$$P(thermostat\ abnormal) =$$
$$P(thermostat\ abnormal|Thermostat = On) \cdot P(Thermostat = On) +$$
$$P(thermostat\ abnormal|Thermostat = Off) \cdot P(Thermostat = Off)$$

The Bayes Net provides the numbers marked in yellow if we set the condition on both copies of the Thermostat node.

[4] All probabilities in this example are conditioned on Fuel=Present and Temperature=Low, but these are left out in the text for readability.

Similarly we know

$P(furnace\ abnormal) =$

$$P(furnace\ abnormal|Thermostat = On) \cdot P(Thermostat = On) +$$
$$P(furnace\ abnormal|Thermostat = Off) \cdot P(Thermostat = Off)$$

Key insight is that we do not have to break the loop by splitting the Thermostat node, we can do the same exercise splitting the Furnace node as in Figure 75.

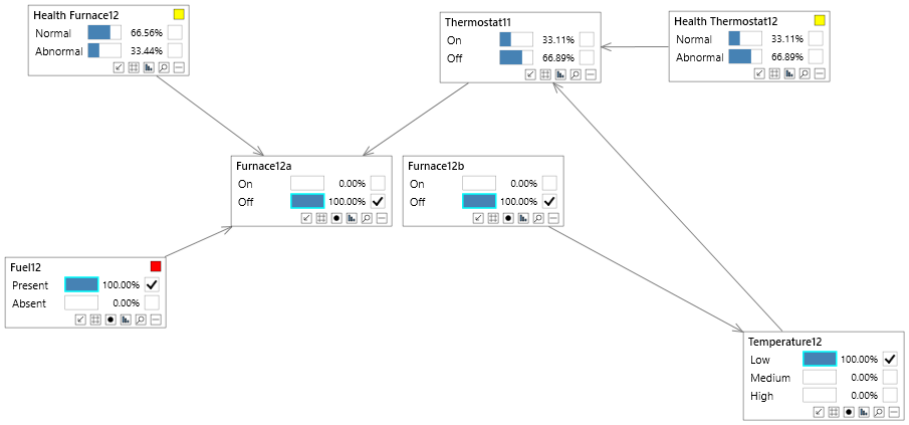


Figure 75 House heating network with Furnace node split into a and b to break the loop. Shown here is the case where Furnace=Off.

This leads to following set of equations:

$P(thermostat\ abnormal) =$

$$P(thermostat\ abnormal|Furnace = On) \cdot P(Furnace = On) +$$
$$P(thermostat\ abnormal|Furnace = Off) \cdot P(Furnace = Off)$$

$P(furnace\ abnormal) =$

$$P(furnace\ abnormal|Furnace = On) \cdot P(Furnace = On) +$$
$$P(furnace\ abnormal|Furnace = Off) \cdot P(Furnace = Off)$$

We now have two expressions for $P(thermostat\ abnormal)$ and $P(furnace\ abnormal)$ so we can combine these equations. Let $\boldsymbol{p} = P(Thermostat = On)$ and $\boldsymbol{q} = P(Furnace = On)$, and replace the known yellow terms given by the Bayes Net with $c_1,\ldots,c_8$. Then combining the four equations above gives us

$$\begin{cases} \boldsymbol{c_1} \cdot \boldsymbol{p} + \boldsymbol{c_2} \cdot (1 - \boldsymbol{p}) = \boldsymbol{c_3} \cdot \boldsymbol{q} + \boldsymbol{c_4} \cdot (1 - \boldsymbol{q}) \\ \boldsymbol{c_5} \cdot \boldsymbol{p} + \boldsymbol{c_6} \cdot (1 - \boldsymbol{p}) = \boldsymbol{c_7} \cdot \boldsymbol{q} + \boldsymbol{c_8} \cdot (1 - \boldsymbol{q}) \end{cases}$$

This is a set of two linear equations that we can solve for $\boldsymbol{p}$ and $\boldsymbol{q}$. Substituting $\boldsymbol{p}$ and $\boldsymbol{q}$ then gives:

$$\begin{cases} P(Thermostat\ abnormal) = 0.403 \\ P(Furnace\ abnormal) = 0.201 \end{cases}$$

This is outcome of our diagnosis. The ratio of 2:1 corresponds to the component prior probability ratio (0.01 and 0.005 respectively).

Note that the sum of the probabilities is not 1.0. This is due to the non-deterministic conditional probability tables that this approach requires. See point 2 in the discussion below.
If we have evidence for a node that is part of the loop, the calculation is much simpler because then we can cut the loop at the node with the evidence and the Bayes Net will give the answer directly, as shown in Figure 76.

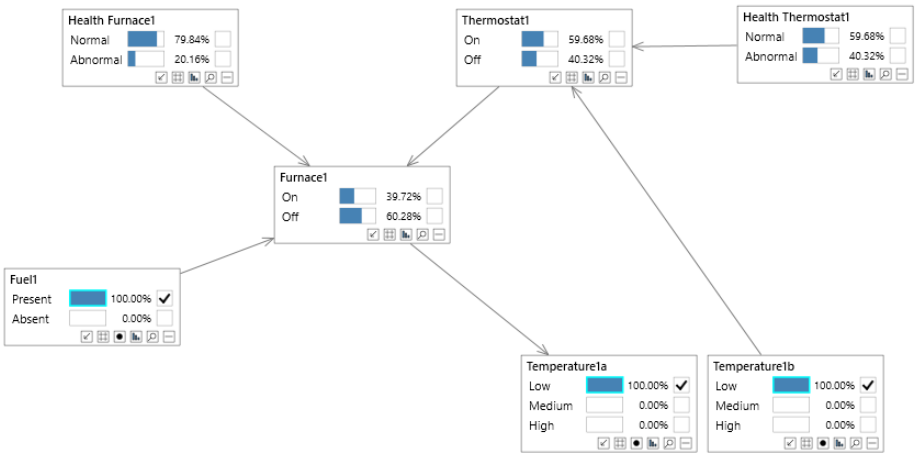


Figure 76 Cutting the loop at the node with evidence. The computed health for Furnace and Thermostat correspond to the values calculated earlier

This approach has two major drawbacks.

1. The math looks convincing but there we cannot justify the method of duplicating a node and setting both versions to the same value. This would need theoretical underpinning.
2. It requires some “fuzziness” in the conditional probability tables. The CPT of the Temperature node is shown in Figure 79 where we encode the possibility that the temperature is low when the furnace is on. This is the moment that the temperature just dropped below the furnace activation threshold of the thermostat. This (small) probability has an impact on the diagnoses outcome but is difficult, if not impossible, to assess[5].
   On a more fundamental level, this actually demands that the conditional probability table is constructed based on the knowledge of the control loop of which it is part of. This defeats the purpose of the approach in which the behavior of the larger system should emerge from its constituent parts.

[5] In fact we considered many variations this CPT and all have their shortcomings.

Furnace11	Temperature11 = Low	Temperature11 = Medium	Temperature11 = High
On	0,01	0,98	0,01
Off	1	0	0

Figure 77 Conditional probability table of the Temperature node

On these grounds this alternative was rejected in favor of the approach discussed in the main text.

Alternative 2 — Model time explicitly

All earlier loop handling methods abstract from time in the loop. A different approach is to model time explicitly. This technique is known as Dynamic Bayesian Belief Networks (DBN).

In a DBN, some links have a *delay*. Nodes that have an incoming delayed link are *temporal nodes* (time-dependent) and so are all its descendants. Figure 78 shows the house heating example with the temporal nodes Furnace, Room Temperature, and Thermostat. Note the self-loops with delay 1 for the latter two nodes.

The model assumes that the outside temperature is cold, i.e. if the furnace is not burning the room temperature will decrease because of the heat loss.

The room temperature has 4 states:

- Cold: Temperature is far below setpoint.
- Little cold: temperature is a little below the setpoint but still acceptable.
- Little hot: temperature is a little above the setpoint but still acceptable.
- Hot: Temperature is far above setpoint.

This encodes the small interval around the setpoint where the actual temperature fluctuates because of the switching behavior of the system.

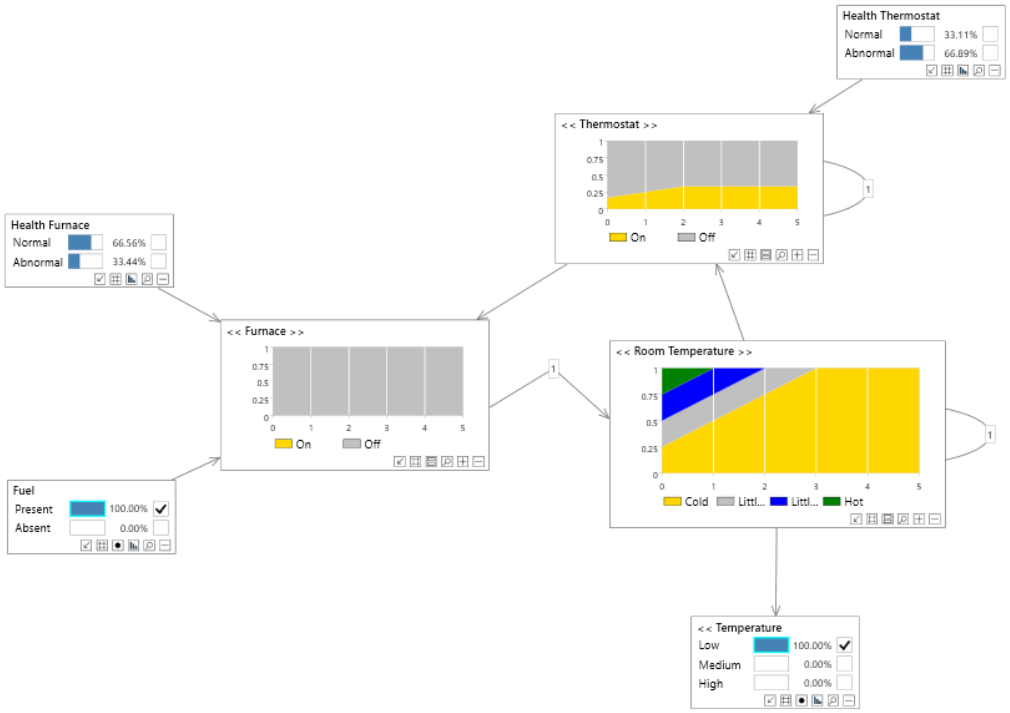


Figure 78 Dynamic Bayesian Network for the house heating example

The Room Temperature is connected from the Furnace indicating that it takes some time for the room to heat up if the Furnace is on. The “loop” Furnace -> Room Temperature -> Thermostat -> Furnace is then not a cycle because it contains a delay link making all its nodes temporal. The figure shows the probability distribution of these nodes in the time stamps t=0 to t=5. The Temperature node is a *final node* indicating (in this case) the temperature at t=5.

To model the dynamics of the house heating example correctly we also added *self-loops* for the Room Temperature and Thermostat. This makes their value dependent on their previous value, effectively turning them into a stateful node. For example, the Room Temperature at time=t then depends on both the Furnace and the Room Temperature at time=(t-1) as shown in Figure 79.

Furnace[t-1]	Room Temperature[t-1]	Room Temperature[t] = Cold	Room Temperature[t] = LittleCold	Room Temperature[t] = LittleHot	Room Temperature[t] = Hot
On	Cold	0	1	0	0
On	LittleCold	0	0	1	0
On	LittleHot	0	0	0	1
On	Hot	0	0	0	1
Off	Cold	1	0	0	0
Off	LittleCold	1	0	0	0
Off	LittleHot	0	1	0	0
Off	Hot	0	0	1	0

Figure 79 Conditional Probability Table (CPT) of the Room Temperature node on the Dynamic Bayesian Belief Network. Note the dependency on the previous time-step denoted by [t-1].

The time variable `t` is not absolute time but advances from `t-1` to `t` when a state change occurs. Consequence is that we cannot just combine multiple loops in one system model as all loops will have different timings and cannot share the same `t`. Instead, we have to connect the final loop result when it reaches a stable state (is at equilibrium, in this example the final node Temperature) to the rest of the system.

In contrast to the alternative presented earlier, all CPT's in this model are deterministic. The only "probabilistic numbers" are the priors at t=0 but since all transitions are deterministic and we wait until the loop reaches an equilibrium, these priors are irrelevant. This is a major benefit over the alternative presented earlier. However, we consider building these detailed dynamic loops models too complicated for our purposes and rejected this alternative in favor of the solution mentioned in the main text.