HUGIN Graphical User Interface Documentation

Release 9.4

HUGIN EXPERT A/S

Jun 13, 2023

CONTENTS

The HUGIN Graphical User Interface is an interactive tool enabling you to use the facilities of the HUGIN Decision Engine. It can help you construct models that can be used in other applications.

The HUGIN Graphical User Interface is also ideal for educational purposes. When introducing the concept of Bayesian networks to a group of students, they will be very motivated if they can model and test Bayesian networks using an easy-to-use tool.

CHAPTER

ONE

INTRODUCTION

1.1 The Origin of HUGIN

During an EU sponsored research project (under the ESPRIT program) on diagnosing neuromuscular diseases, the Bayesian network MUNIN was constructed. A research group at Aalborg University worked on developing correct and efficient computation methods for the diagnosis problem. Some results had at that time been obtained by American researchers, but a very obstinate problem still remained, which prevented Bayesian networks from being used in the construction of expert systems. The problem was known as the rumour problem: you may hear the same story through several different channels; but still the story may originate from the same source. Without knowing whether or not your channels are independent, you cannot combine the statements correctly.

In Bayesian networks, the rumour problem appears when a cause can influence the same event through different paths in the network.

The problem was solved and general methods were made available to be used in any domain which can be modeled by a Bayesian network.

The methods were programmed into a general development and runtime system, which was easy to use for anyone wishing to construct an expert system based on Bayesian networks. The system was called HUGIN. Over the years the system has been extended in various ways (e.g. (limited-memory) influence diagrams (LIMIDs), continuous variables, structure learning, adaptation, object-oriented specification of Bayesian networks and LIMIDs, etc).

1.2 Basic Concepts

Before you can use the HUGIN Graphical User Interface , you should at least understand the concept of *Bayesian Network* (page 19) which is described in the *Tutorials* (page 15) section. This section also contains a step-by-step description of how to *construct a Bayesian network* (page 26) using the HUGIN Graphical User Interface.

The extension of Bayesian networks with decision and utility nodes, known as influence diagrams, allows you to model decision scenarios explicitly. If you are not familiar with (limited-memory) influence diagrams (*LIMIDs* (page 37)), you can also learn about these in the *Tutorials* (page 15) section. There is also a step-by-step description of how to *construct a LIMID* (page 45) using the HUGIN Graphical User Interface.

Also, you can learn about the concept of *object-oriented networks* (page 51), which provides a very powerful mechanism for constructing models with repetitive patterns and for constructing models in a hierarchical fashion (top-down, bottom-up, or a mix of the two), making large models much more readable. Again, there is a step-by-step description of how to *construct an object-oriented network* (page 56),using the HUGIN Graphical User Interface.

1.3 The HUGIN Development Environment

The HUGIN Graphical User Interface is a component of *the HUGIN Development Environment* (page 5). The two other main components are the *HUGIN Decision Engine* (page 5) and the *HUGIN Application Program Interface* (page 6).

CHAPTER

TWO

THE HUGIN DEVELOPMENT ENVIRONMENT

The HUGIN Development Environment provides a set of tools for constructing model-based decision support systems in domains characterized by inherent uncertainty. The models supported are Bayesian networks (BNs) and their extension (limited-memory) influence diagrams (LIMIDs). The HUGIN Development Environment allows you to define both discrete domain variables and to some extent continuous domain variables in your models.

You have the opportunity to use the HUGIN Decision Engine (HDE) through the HUGIN Graphical User Interface – an easy-to-use graphical environment. You can also use the HDE through one of several APIs (Application Program Interfaces) which come as libraries for C, C++, .NET and Java, and as an ActiveX server. More information about the versions of the HUGIN Development Environment that are available can be found in the *Versions* (page 11) section.

The HUGIN Development Environment can be used to construct models as components in applications for decision support, data mining, and expert systems. The application communicates with the constructed component models through one of the *HUGIN APIs* (page 5).

2.1 Components of the HUGIN Development Environment

The HUGIN Development Environment has three main components: The HUGIN Decision Engine, a collection of Application Program Interfaces, and the HUGIN Graphical User Interface.

2.1.1 The HUGIN Decision Engine

The HUGIN Decision Engine (HDE) performs reasoning on a knowledge base represented as a Bayesian network or LIMID. The HDE performs all data processing and storage maintenance associated with the reasoning process. An important part of the HDE is the compiler, which transforms networks into efficient structures (known as junction trees), making possible to perform inference (reasoning) in the networks very efficiently. For small and medium sized networks, inference takes fractions of a second, and even for large networks it most often takes only a few seconds.

The HDE can be accessed through HUGIN Graphical User Interface, or it can be accessed from application programs using one of the HUGIN Application Program Interfaces.

2.1.2 Application Program Interfaces

The HUGIN APIs (Application Program Interfaces) allow programmers can build knowledge-based applications, utilizing the power of the HUGIN Decision Engine for reasoning. When used through one of the HUGIN APIs, the HUGIN Decision Engine functions as an ordinary program library, giving the application programmer total control of events.

The HUGIN APIs are currently available as C, C++, .NET/.Net Core, Java, Python, Web Service API and Visual Basic on Windows via a COM interface.

You can download the HUGIN API reference manuals from the HUGIN web site.

HUGIN also offers APIs for Android and IOS. Please contact HUGIN at hugin@info.dk for more information.

2.1.3 The HUGIN Graphical User Interface

The HUGIN Graphical User Interface is used to create and maintain network models as well as executing them (entering evidence and displaying resulting probability distributions and expected utilities). The HUGIN Graphical User Interface can operate in two different modes: Edit Mode and Run Mode. In Edit Mode, nodes can be created and linked, states and actions can be specified, and conditional probability tables and utility tables can be entered using a window-, menu- and mouse driven interface. In Run Mode, beliefs and utilities for individual nodes can be displayed. The user can enter evidence incrementally by selecting states/actions of individual nodes. The HUGIN Decision Engine can then be engaged, propagating the information to obtain revised probabilities and expected utilities.

2.1.4 Extra Features

For extra performance gains, the HUGIN Decision Engine and compiler features a facility for compressing sparse probability tables. This can save considerable space and likewise considerably increase performance. The compiler and inference engine also features an option for approximating the probability tables to increase their scarcity. Combined with the compression, this can have dramatic effects on performance, with negligible effects on end results.

2.2 Bayesian Network Technology in the HUGIN Development Environment

This text provides you with an overview of Bayesian networks, LIMID models, and networks with conditional Gaussian variables (also known as Bayesian networks). More detailed information on these and other issues can be found in specialized sections and *tutorials* (page 15)

Probabilistic graphical models (of which Bayesian networks and LIMIDs are excellent examples) are especially suitable for domains with inherent uncertainty. Domain models based on Bayesian network technology can be built using the HUGIN Graphical User Interface, an easy-to-use graphical user interface built on top of the Java API provided with the HUGIN Developer package. Stand-alone applications using network technology can be built using the *HUGIN APIs* (page 5), a series of C, C++, and Java libraries and an ActiveX server providing access to the facilities of the HUGIN Decision Engine (HDE) for the application programmer.

Bayesian Networks

Bayesian networks are often used to model domains that are characterized by inherent uncertainty. (This uncertainty can be due to imperfect understanding of the domain, incomplete knowledge of the state of the domain at the time where a given task is to be performed, randomness in the mechanisms governing the behavior of the domain, or a combination of these.)

Formally, a Bayesian network can be defined as follows:

Definition (Bayesian Network) A Bayesian network is a directed acyclic graph with the following properties:

- Each node represents a random variable.
- Each node representing a variable A with parent nodes representing variables B_1, $B2_2$,…, B_n is assigned a conditional probability table (CPT):

$$P(A|B_1, B_2, ..., B_n)$$

The nodes represent random variables, and the edges represent probabilistic dependences between variables. These dependences are quantified through a set of conditional probability tables (CPTs): Each variable is assigned a CPT of the variable given its parents. For variables without parents, this is an unconditional (also called a marginal) distribution.

Example 1

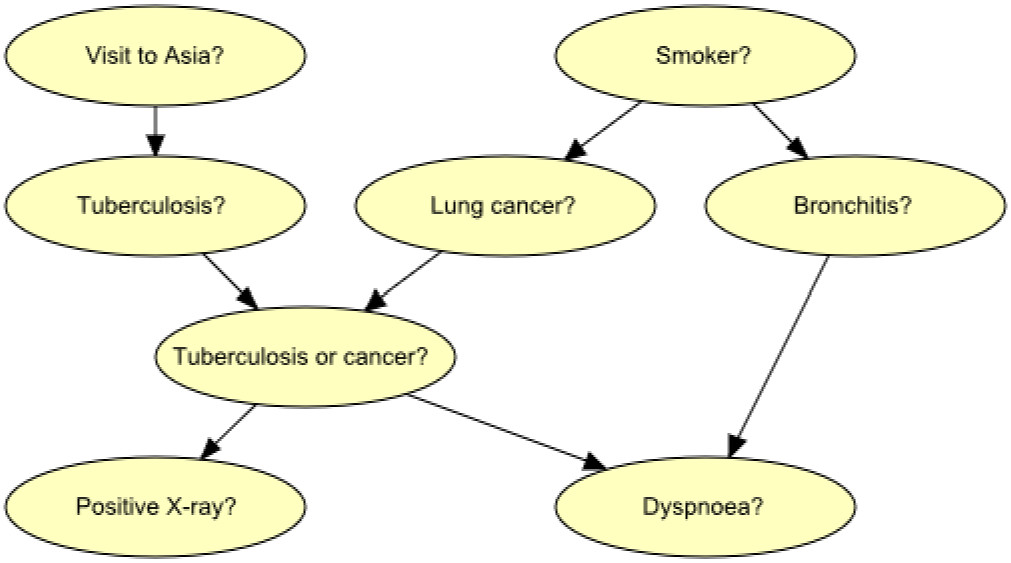


Figure 1: Graph representing structural aspects of medical knowledge concerning lung diseases.

Figure 1 shows a model for the following piece of fictitious medical knowledge:

> "Shortness-of-breath (dyspnoea) [d] may be due to tuberculosis [t], lung cancer [l] or bronchitis [b], or none of them, or more than one of them. A recent visit to Asia [a] increases the risk of tuberculosis, while smoking [s] is known to be a risk factor for both lung cancer and bronchitis. The result of a single chest X-ray [x] does not discriminate between lung cancer and tuberculosis, neither does the presence or absence of dyspnoea" *(Lauritzen & Spiegelhalter 1988)* (page 541).

The last fact is represented in the graph by the intermediate variable e. This variable is a logical-or of its two parents (t and l); it summarizes the presence of one or both diseases or the absence of both.

An important concept for Bayesian networks is *conditional independence*. Two sets of variables, A and B, are said to be (conditionally) independent given a third set C of variables if when the values of the variables C are known, knowledge about the values of the variables B provides no further information about the values of the variables A:

$$P(A|B, C) = P(A|C)$$

Conditional independence can be directly read from the graph as follows: Let A, B, and C be disjoint sets of variables, then

- identify the smallest sub-graph that contains A B C and their ancestors;

- add undirected edges between nodes having a common child;

drop directions on all directed edges.

Now, if every path from a variable in A to a variable in B contains a variable in C, then A is conditionally independent of B given C *(Lauritzen et al. 1990)* (page 541).

Example 2

If we learn the fact that a patient is a smoker, we will adjust our beliefs (increased risks) regarding lung cancer and bronchitis. However, our beliefs regarding tuberculosis are unchanged (i.e., t is conditionally independent of s given the empty set of variables). Now, suppose we get a positive X-ray result for the patient. This will affect our beliefs regarding tuberculosis and lung cancer, but not our beliefs regarding bronchitis (i.e., b is conditionally independent of x given s). However, had we also known that the patient suffers from shortness-of-breath, the X-ray result would also have affected our beliefs regarding bronchitis (i.e., b is *not* conditionally independent of x given s and d).

These (in)dependences can all be read from the graph of Figure 1 using the method described above.

Another, equivalent and very popular criterion for reading statements of conditional independence is *d-separation* (page 289) *(Pearl 1988)* (page 541).

2.2.1 Inference

Inference in a Bayesian network means computing the conditional probability for some variables given information (evidence) on other variables. This is easy when all available evidence is on variables that are ancestors of the variable(s) of interest. But when evidence is available on a descendant of the variable(s) of interest, we have to perform inference opposite the direction of the edges. To this end, we employ *Bayes' Theorem*:

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

HUGIN inference is essentially a clever application of Bayes' Theorem; details can be found in the paper by *Jensen et al. (1990(1))* (page 541).

2.2.2 Limited Memory Influence Diagrams

A *LIMID* is a Bayesian network augmented with decision and utility nodes (the random variables of a LIMID diagram are often called chance variables). Edges into decision nodes indicate *time precedence*: an edge from a random variable to a decision variable indicates that the value of the random variable is known when the decision will be taken, and an edge from one decision variable to another indicates the chronological ordering of the corresponding decisions. The network must be acyclic.

We are interested in making the best possible decisions for our application. We therefore associate *utilities* with the state configurations of the network. These utilities are represented by *utility nodes*. Each utility node has a utility function that to each configuration of states of its parents associates a utility. (Utility nodes do not have children). By making decisions, we influence the probabilities of the configurations of the network. We can therefore compute the *expected utility* of each decision alternative (the global utility function is the sum of all the local utility functions). We shall choose the alternative with the highest expected utility; this is known as the *maximum expected utility principle*.

When we have observed the values of the variables that are parents of the first decision node in the LIMID, we want to know the maximum expected utilities for the alternatives of this decision. The HUGIN Decision Engine will compute these utilities on the assumption that all future decisions will be made in an optimal manner (using all available evidence at the time of each decision). Similar considerations also apply to the remaining decisions.

The computational method underlying the implementation of LIMIDs in the HUGIN Decision Engine is described by *S. L. Lauritzen and D. Nilsson (2001)* (page 541).

Example 3

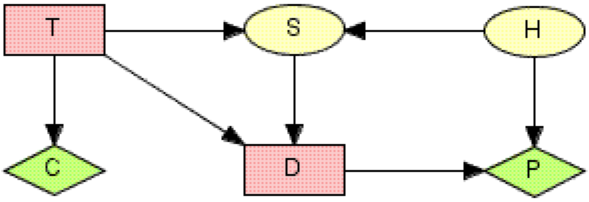
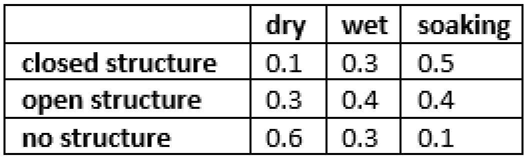


Figure 2: LIMID model for the oil wildcatters decision problem: *T* represents the decision whether or not to test; *D* represents the decision whether to drill or not to drill; *S* represents the outcome of the seismic soundings test (if the oil wildcatter decides to test); *H* represents the state of the hole; *C* represents the cost associated with the seismic soundings test; and *P* represents the expected payoff associated with drilling.

An oil wildcatter must decide whether or not to drill. He is uncertain whether the hole is *dry*, *wet*, or *soaking*. At a cost of $10,000, the oil wildcatter could take seismic soundings which will help determine the underlying geological structure at the site. The soundings will disclose whether the terrain below has closed structure (good), *open* structure (so-so), or *no* structure (bad).

The cost of drilling is $70,000. If the oil wildcatter decides to drill, the expected payoff (i.e., the value of the oil found minus the cost of drilling) is $-70,000 if the hole is *dry*, $50,000 if the hole is *wet*, and $200,000 if the hole is *soaking*; if the oil wildcatter decides not to drill, the payoff is (of course) $0.

The experts have estimated the following probability distribution for the state of the hole: *P(dry)=0.5*, *P(wet)=0.3*, and *P(soaking)=0.2*. Moreover, the seismic soundings test is not perfect; the conditional probabilities for the outcomes of the test given the state of the hole are:



	dry	wet	soaking
closed structure	0.1	0.3	0.5
open structure	0.3	0.4	0.4
no structure	0.6	0.3	0.1

Figure 3: CPT for the outcomes of the test

Figure 2 shows LIMID model for the oil wildcatters decision problem. Random variables are depicted as circles, decision variables are depicted as squares, and utilities are depicted as diamonds.

On the basis of this LIMID, the HUGIN Decision Engine computes the utility associated with testing to be $22,500 and the utility associated with not testing to be $20,000. So the optimal strategy is to perform the seismic soundings test, and then decide whether to drill or not to drill based on the outcome of the test.

[This example is due to *Raiffa (1968)* (page 541). A more thorough description of this example is found in the *Examples* (page 517) section.]

2.2.3 Networks with Conditional Gaussian Variables

The HUGIN Decision Engine is able to handle networks with both discrete and continuous random variables. The continuous random variables must have a *Gaussian* (also known as a *normal*) distribution conditional on the values of the parents.

The distribution for a continuous variable Y with discrete parents I and continuous parents Z is a (one-dimensional) Gaussian distribution conditional on the values of the parents:

$$P(Y|I=i, Z=z) = \mathcal{N}(\alpha(\rangle) + \beta(\rangle)^{\tau}\ddagger, \gamma(\rangle)))$$

Note that the mean depends linearly on the continuous parent variables and that the variance does not depend on the continuous parent variables. However, both the linear function and the variance are allowed to depend on the discrete parent variables. These restrictions ensure that exact inference is possible.

Note that discrete variables cannot have continuous parents.

Example 4

Figure 3 shows a network model for a waste incinerator *(Lauritzen 1992)* (page 541):

> "The emissions (of dust and heavy metals) from a waste incinerator differ because of compositional differences in incoming waste [*W*]. Another important factor is the waste burning regimen [*B*], which can be monitored by measuring the concentration of CO2 in the emissions [*C*]. The filter efficiency [*E*] depends on the technical state [*F*] of the electrofilter and on the amount and composition of waste [*W*]. The emission of heavy metals [*M:sub:`o`*] depends on both the concentration of metals [*M:sub:`i`*] in the incoming waste and the emission of dust particulates [*D*] in general. The emission of dust [*D*] is monitored through measuring the penetrability of light [*L*]."

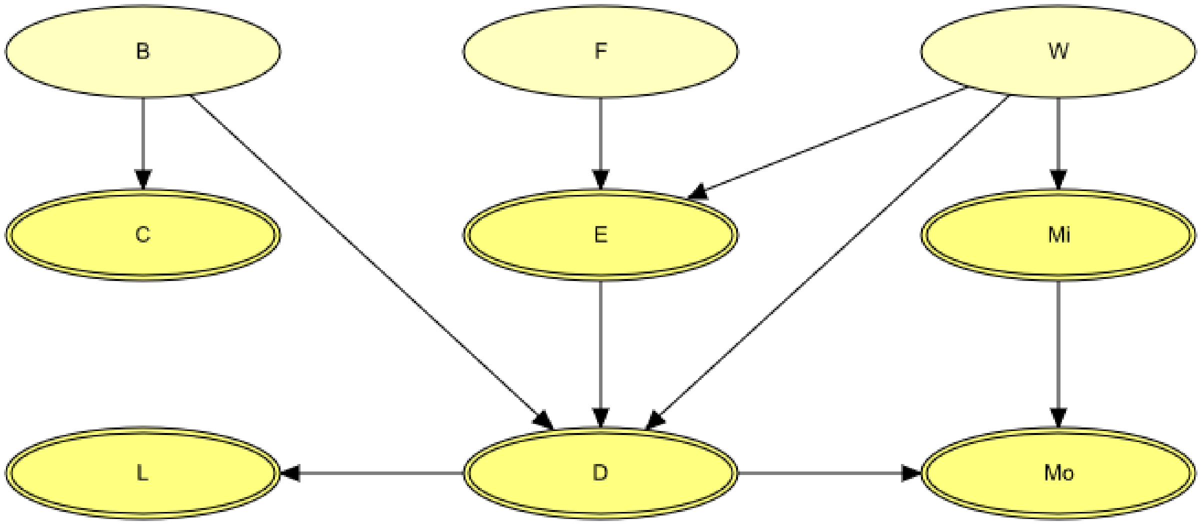


Figure 3: The structural aspects of the waste incinerator model: B, F, and W are discrete variables, while the remaining variables are continuous.

The result of inference within a network model containing conditional Gaussian variables is - as always - the beliefs (i.e., marginal distributions) of the individual variables given evidence. For a discrete variable this amounts to a probability distribution over the states of the variable. For a conditional Gaussian variable two measures are provided:

1. the mean and variance of the distribution;
2. since the distribution is in general not a simple Gaussian distribution, but a mixture (i.e., a weighted sum) of Gaussians, a list of the parameters (weight, mean, and variance) for each of the Gaussians is available.

Example 5

From the network shown in Figure 3 (and given that the discrete variables B, F, and W are all binary), we see that

- the distribution for C can be comprised of up to two Gaussians (one if B is instantiated);
- initially (i.e., with no evidence incorporated), the distribution for E is comprised of up to four Gaussians;
- if L is instantiated (and none of B, F, or W is instantiated), then the distribution for E is comprised of up to eight Gaussians.

2.3 Versions

There are various configurations[1] of the HUGIN Development Environment. Overall they can be divided into two categories: Commercial and Academic Licences. Furthermore, a popular free evaluation version with limited functionality can be downloaded from our web site and an OEM version can be achieved for deployment in production setups.

2.3.1 Commercial Versions

For Commercial users, two versions of the HUGIN Development Environment are available: HUGIN Explorer™[2], and HUGIN Developer™[3]. HUGIN Explorer comprises HUGIN Graphical User Interface that is an interactive Java-based tool built on top of the HUGIN Java API, and thus enables you to use the facilities of the HDE for constructing, manipulating, and evaluating models that can be used in other applications. HUGIN Developer™ comprises HUGIN Graphical User Interface and one of the available Hugin APIs, and it is intended for developers using the HUGIN Decision Engine (HDE) as part of custom-built applications. It provides access to all functions of the Decision Engine through an API. HUGIN Explorer™ and HUGIN Developer both allow construction of Bayesian networks of any size, limited only by the amount of virtual memory available. Neither version has limitations on the number of nodes or the number of states of the nodes.

2.3.2 Academic Versions

The following two versions of the HUGIN Development Environment are targeting academic users: HUGIN Educational™[4], and HUGIN Researcher™[5]. HUGIN Educational™ is targeting employees or students at an academic facility who wish to learn about and utilize Bayesian network technology. It comprises HUGIN Graphical User Interface that is an interactive Java-based tool built on top of the HUGIN Java API, and thus enables students and researchers to use the facilities of the HDE for constructing, manipulating, and evaluating models that can be used in other applications. HUGIN Researcher™ is targeting employees or students at an academic facility who using Bayesian network technology to build advanced applications for research purposes to be used for academic purposes. It comprises the HUGIN Graphical User Interface and the HUGIN APIs, and it is intended for students and researchers using the HUGIN Decision Engine (HDE) as part of their academic work. It provides access to all functions of the Decision Engine through an API. HUGIN Educational™ and HUGIN Researcher™ both allow construction of Bayesian networks of any size, limited only by the amount of virtual memory available. Neither version has limitations on the number of nodes or the number of states of the nodes.

[1] https://www.hugin.com/index.php/products/

[2] https://www.hugin.com/index.php/hugin-explorerhugin-educational//

[3] https://www.hugin.com/index.php/hugin-developerhugin-researcher/

[4] https://www.hugin.com/index.php/hugin-explorerhugin-educational/

[5] https://www.hugin.com/index.php/hugin-developerhugin-researcher/

2.3.3 Free Evaluation Version

HUGIN Lite™ is a free version free version[6] of HUGIN Developer™ for people who do not trust sales literature. HUGIN Lite™ contains all the basic functionality of HUGIN Developer™, except for loading, construction, and saving of models with more than 50 states, and a restriction to 500 cases when learning. HUGIN Lite™ is intended to give an idea of the possibilities and facilities in HUGIN Developer™ and HUGIN Researcher™.

2.3.4 Deployment Licences

For clients who want to incorporate HUGIN technology into products or services an OEM agreement[7] can be made allowing the client to deploy the *HUGIN APIs* (page 5) in production environments.

2.3.5 Special HUGIN API Versions

The HUGIN Decision Engine is highly portable. This means that versions of the HUGIN API can be produced for many types of machines and operating systems. If you have special requirements, or special machines you would like to see supported, please contact HUGIN Expert A/S at info@hugin.com. If possible, we will be happy to produce special HUGIN API versions on request.

2.4 System Requirements

To use the HUGIN Development Environment (if not a special version - see the *Versions section* (page 11)) you need:

- Windows: Microsoft Windows 10/11 (x86/x64)
- Linux: Red Hat Enterprise Linux 7, Ubuntu 18.04 LTS, Ubuntu 22.04 LTS (x64), and compatible distributions (x86/x64)
- macOS: macOS Monterey, macOS Ventura

The Java API and the Graphical User Interface requires Oracle's JVM to run. On macOS, Oracle's JVM is only available in 64 bit.

2.4.1 Memory Requirements

The complexity (both in terms of space and time) of making inference in a Bayesian network or a LIMID model can be quite large. For most applications, however, the complexity is moderate.

The time and memory required for making inference depends on the number of variables, the number of states of the variables, and the structure of the network. Among these three factors, it is the structure of the network that is the most important. As an example, an early version of the TREAT network (a decision support system for treatment of severe bacterial infections) developed at Aalborg University contains more than 1400 variables, but still requires less than 2 Mbytes of memory (using optimal triangulation). On the other hand, one can easily construct networks with less than 50 variables where the complexity of inference gets prohibitive. Note that by using the *compression* (page 125) and *approximation* (page 125) facilities of the HUGIN compiler, memory requirements can be vastly reduced. Also, using the *optimal triangulation feature* (page 126) when compiling a network can be very useful in minimizing the complexity of inference.

Please note that the memory requirements mentioned above assume that we are using the standard single-precision HUGIN API. (Using the double-precision version will double that amount.) If we are using the HUGIN Graphical

[6] https://www.hugin.com/index.php/hugin-lite/
[7] https://www.hugin.com/index.php/hugin-oem/

User Interface, there will also be a memory overhead from running Java and representing the variables as Java objects, etc.

In most cases, when our customers experience problems with lack of memory, the problems were caused by inappropriate structures in the models which could be solved by optimizing the representation.

See the section on *junction trees* (page 254) for a more detailed discussion of how inference is performed and what causes the potential high complexity.

2.4.2 Application Programming Interfaces

Application Programming Interfaces (APIs) for the HUGIN Decision Engine (HDE) are provided for the C, C++, Java, Python, and C# (.NET Core 2.0) programming languages for all supported platforms (Windows, Linux, and macOS). Additionally, a web service API is provided for all supported platforms.

For the Microsoft Windows platform, APIs for C# (.NET Framework 2.0 and 4.0) and Excel (COM) are also provided.

For macOS and iOS, library frameworks for the Swift programming language are provided.

For the Microsoft Windows platform, DLLs for C/C++ are provided for use with Visual Studio 6.0, Visual Studio .NET 2003, Visual Studio 2005, Visual Studio 2008, Visual Studio 2010, Visual Studio 2012, Visual Studio 2013, Visual Studio 2015, Visual Studio 2017, and Visual Studio 2019.

For the Linux platform, two software packages are available: One compiled on Red Hat Enterprise Linux 7 (RHEL7), and one compiled on Ubuntu 18.04 LTS (64-bit only). Most Linux software distributions should be compatible with at least one of these packages. If you experience problems, please contact HUGIN Expert A/S.

The packages have been compiled using the default system compiler. For RHEL7, this is gcc/g++ 4.8.5, and for Ubuntu 18.04 LTS, this is gcc/g++ 7.4.0. The APIs may not work with other compilers (including the version of the compiler). This is most likely to be the case for the HUGIN C++ API.

For macOS/iOS, all libraries have been compiled using Xcode 10.3.

The HUGIN Java API is compatible with Java 1.4.2 and newer versions from Oracle.

The HUGIN Web Service API requires Java 8 or newer on the server.

The HUGIN Python API works with Python 2.7 and Python 3.x.

If you have special requirements (e.g., operating systems, compilers, or hardware) you would like to see supported, please contact HUGIN Expert A/S. If possible, we will be happy to produce special HUGIN API versions on request.

Upon taking any version of the HUGIN software into use, you automatically agree to the terms of the HUGIN Software License Agreement (below).

2.5 HUGIN Software License Agreement

HUGIN Expert A/S, hereinafter called "*HUGIN*", and the organization/institution specified in the order form, hereinafter called "*LICENSEE*", agree that the following terms and conditions will apply to the use of the computer software product specified in the order form, hereinafter called "*HUGIN SOFTWARE*", a copy of which is being supplied by *HUGIN to LICENSEE.*

1. Grant. HUGIN provides *HUGIN SOFTWARE* to *LICENSEE* and grants to *LICENSEE* a non-exclusive and non-transferable right to use this software. No license, right or interest in any trademark, trade name or service mark of *HUGIN SOFTWARE* or any third party from whom *HUGIN SOFTWARE* has acquired such license rights are granted under this License. *HUGIN SOFTWARE* should be used by the *LICENSEE* only on a computer system owned, leased or operated by the *LICENSEE*. The maximum number of simultaneous users for *HUGIN SOFTWARE* shall be as specified in the order form.

2. Term. This Agreement shall become effective upon its acceptance by the authorized representatives of *HUGIN* and *LICENSEE*. The Agreement shall remain in effect in perpetuity, except that HUGIN may discontinue the license or terminate the Agreement if *LICENSEE* fails to comply with any terms or conditions thereof.

3. Charges. Within thirty (30) days of the effective date of this Agreement, *LICENSEE* shall pay *HUGIN* the charge specified in *HUGIN's* current Price Schedule. *LICENSEE* shall also pay sales taxes if applicable.

4. Restricted Use. *LICENSEE* agrees to maintain *HUGIN SOFTWARE* source code and object code in confidence and will not make *HUGIN SOFTWARE* or any derivative thereof available for any use whatsoever in any form to any other individual or firm. *LICENSEE* agrees to take appropriate action by instruction, agreement or otherwise with all persons permitted access to *HUGIN SOFTWARE* or any derivative thereof to satisfy LICENSEE's protection and security obligations under this Agreement. *LICENSEE* may take additional copies, in whole or in part, of *HUGIN SOFTWARE* as necessary and incidental to its use in compliance with this Agreement, such as for archival and back-up purposes, provided that each such copy, in whole or in part, shall remain subject to all terms of this Agreement. *LICENSEE* shall not disassemble or decompile the *HUGIN SOFTWARE*.

5. Non-Assignment. Under no circumstances shall this Agreement or any of the rights granted to *LICENSEE* hereunder be sold, assigned or sub-licensed, voluntarily or by operation by law, to any other person or entity, and any such purported sale, assignment or sub-license shall be void. The API that ships together with the *HUGIN SOFTWARE* has a tight link to the installation of these packages. It's not possible to use the API's on another computer unless the whole package is installed. To be able to distribute applications using the HUGIN API a HUGIN OEM License is imperative. Please contact HUGIN Expert A/S for further information on the HUGIN OEM license.

6. Warranty. *HUGIN SOFTWARE* is provided without warranty of any kind, either expressed or implied, including without limitation implied warranties or merchantability and fitness for a particular purpose. *HUGIN* disclaims any responsibility for ease of installation, accuracy, completeness or correctness of *HUGIN SOFTWARE*. HUGIN does not warrant that *HUGIN SOFTWARE* will meet *LICENSEE*'s requirements or that operations involving *HUGIN SOFTWARE* will be uninterrupted or error free.

7. Applicable Law. This Agreement shall be governed by, subject to and interpreted in accordance with Danish law, and *LICENSEE* and *HUGIN* shall submit to the jurisdiction of the Danish court.

CHAPTER

THREE

TUTORIALS

A number of tutorials are provided to help you getting acquainted with the HUGIN technology and with the HUGIN Graphical User Interface. There is one section of tutorials that introduce some basic concepts, and another that presents some more advanced features of the HUGIN Graphical User Interface.

3.1 Basic Concepts

- The *Paradigms Tutorial* (page 16) presents the three main paradigms for expert systems: Rule-based systems, Neural networks, and Bayesian networks.
- The *Bayesian Networks Tutorial* (page 19) describes the basic properties of Bayesian networks, and is recommended if you have no or little prior knowledge about Bayesian networks.
- The *How to Build BNs Tutorial* (page 26) provides a step-by-step guide to constructing a Bayesian network using the HUGIN Graphical User Interface.
- The *Limited Memory Influence Diagrams Tutorial* (page 37) describes the basic properties of limited memory influence diagrams, and is recommended if you have no or little prior knowledge about limited memory influence diagrams (LIMIDs).
- The *How to Build LIMIDs Tutorial* (page 45) provides a step-by-step guide to constructing a LIMID using the HUGIN Graphical User Interface.
- The *Object Orientation Tutorial* (page 51) describes the basic properties of object-oriented Bayesian networks and LIMIDs, and is recommended if you have no or little prior knowledge about this subject.
- The *How to Build OOBNs Tutorial* (page 56) provides a step-by-step guide to constructing an object-oriented Bayesian network using the HUGIN Graphical User Interface.

3.2 Learning More

- The *Node Table Tutorial* (page 63) explains the functionalities of node tables.
- The *Table Generator Tutorial* (page 73) shows how to specify simple expressions for large tables and then let the built-in table generator do all the hard work of filling in the numbers of the table.
- The *Case and Data File Formats Tutorial* (page 85) describes how data for learning may be specified as case and data files.
- The *Structure Learning Tutorial* (page 97) describes how Bayesian networks can be constructed automatically from data.
- The *EM Learning Tutorial* (page 105) describes how the probabilities (parameters) of Bayesian networks can be learned automatically from data.

- The *Adaptation Tutorial* (page 111) explains how the probabilities specified for Bayesian networks can be automatically updated from experience (i.e., evidence) such that, for example, the networks adapt to changing conditions its environment.
- The *Case Generator Tutorial* (page 86) explains how to generate simulated cases from a Bayesian network.

3.2.1 Paradigms of Expert Systems

Section author: By Finn V. Jensen, Dept. of Computer Science, Aalborg University, Denmark

This is a brief overview of the three main paradigms of expert systems.

Rule-Based Systems

A rule is an expression of the form

if *A* **then** *B*

where *A* is an assertion and *B* can be either an action or another assertion. For instance, the following three rules could be part of a larger set of rules for troubleshooting water pumps:

1. **If** pump failure **then** the pressure is low
2. **If** pump failure **then** check oil level
3. **If** power failure **then** pump failure

A *rule-based system* consists of a library of such rules. These rules reflect essential relationships within the domain, or rather: they reflect ways to reason about the domain.

When specific information about the domain becomes available, the rules are used to draw conclusions and to point out appropriate actions. This is called *inference*. The inference takes place as a kind of chain reaction. In the above example, if you are told that there is a power failure, Rule 3 will state that there is a pump failure and Rule 1 will then tell us that the pressure is low. Rule 2 will also give a (useless) recommendation to check the oil level.

Rules can also be used in the opposite direction. Suppose you are told that the pressure is low; then Rule 1 states that it can be due to a pump failure, while Rule 3 states that a pump failure can be caused by a power failure. You should also be able to use Rule 2 to recommend checking the oil level, but it is very difficult to control such a mixture of inference back an forth in the same session.

Uncertainty

Often the connections reflected by the rules are not absolutely certain, and also the gathered information is often subject to uncertainty. In such cases, a *certainty measure* is added to the premises as well as to the conclusions in the rules of the system. Now, a rule gives a function that describes how much a change in the certainty of the premise will change the certainty of the conclusion. In its simplest form, this looks like:

4. **If** *A* (with certainty *x*) **then** *B* (with certainty *f(x)*)

There are many schemes for treating uncertainty in rule-based systems. The most common are *fuzzy logic*, *certainty factors*, and (adapted versions of) *Dempster-Shafer belief functions*. Common to all of these schemes is that uncertainty is treated locally. That is, the treatment is connected directly to the incoming rules and the uncertainty of their elements. Imagine, for example, that in addition to Rule 4 we have the rule

5. **If** *C* (with certainty *x*) **then** *B* (with certainty *g(x)*)

If we now get the information that A holds with certainty a and C holds with certainty *c*, what is then the certainty of *B*?

There are different algebras for such a combination of uncertainties, depending on the scheme. Common to all these algebras is that in many cases they come to incorrect conclusions. This is because the combination of uncertainty is not a local phenomenon, but it is strongly dependent on the entire situation (in principle a global matter).

Neural Networks

(Only the so-called *feed-forward* networks are treated.)

A neural network consists of several layers of nodes: At the top there is a layer of *input nodes*, at the bottom a layer of *output nodes*, and in between these normally 1 or 2 *hidden layers*. Except for the output nodes, all nodes in a layer are in principle connected to all nodes in the layer immediately below. A node along with its in-going edges is called a *perceptron*.

A neural network performs *pattern recognition*. You could for instance imagine a neural network that reads handwritten letters. By automatic tracking, a handwritten letter can be transformed into a set of findings on curves (not a job for the network). The network will have an input node for every possible kind of finding and an output node for each letter in the alphabet. When a set of findings is fed into the network, the system will match the pattern of findings with equivalent patterns of the different letters.

Technically, the input nodes are given a value (0 or 1). This value is transmitted to the nodes in the next layer. Each of these nodes perform a weighted sum of the incoming values, and if this sum is greater than a certain threshold, the node fires downward with the value 1. The values of the output nodes determine the letter.

So, apart from the architecture of the network (the number of layers and the number of nodes in each layer), the *weights* and the *thresholds* determine the behavior of the network. Weights and thresholds are set in order for the network to perform as well as possible. This is achieved by *training*: You have a large number of examples where both input values and output values are known. These are then fed into the training algorithm of the network. This algorithm determines weights and thresholds in such a way that the distance between the set of outputs from the network and the desired sets of outputs from the examples gets as small as possible.

There is nothing preventing the use of neural networks for domains requiring the handling of uncertainty. If relations are uncertain (for example in medical diagnosis), a neural network with the proper training will be able to give the most probable diagnosis given a set of symptoms. However, you will not be able to read the uncertainty of the conclusion from the network, you will not be able to get the next-most probable diagnosis and - probably the most severe set-back - you will not know under which assumptions about the domain the suggested diagnosis is the most probable.

Bayesian Networks

Bayesian networks are also called *Bayes nets*, *causal probabilistic networks* (CPNs), *Bayesian belief networks* (BNs), or *belief networks*.

A Bayesian network consists of a set of *nodes* and a set of directed *edges* between these nodes. Edges reflect cause-effect relations within the domain. These effects are normally not completely deterministic (e.g. disease -> symptom). The *strength* of an effect is modeled as a probability:

6. **If** tonsillitis **then** P(temp>37.9) = 0.75
7. **If** whooping cough **then** P(temp>37.9) = 0.65

One could be led to read these statements as rules. They shouldn't. So, a different notation is used:

P(temp>37.9 | whooping cough) = 0.65

If 6. and 7. are read as '**If** otherwise healthy and..**then**..', there also needs to be a specification of how the two causes combine. That is, we need the probability of having a fever if both symptoms are present and if the patient is completely healthy. All in all you have to specify the *conditional* probabilities:

P(temp>37.9 | whooping cough, tonsillitis),

Where ‘whooping cough’ and ‘tonsillitis’ each can take the states ‘yes’ and ‘no’. So, you must for any node specify the strength of all combinations of states for the possible causes.

Fundamentally, Bayesian networks are used to update probabilities whenever information becomes available. The mathematical basis for this is Bayes’ theorem:

P(A | B) P(B) = P(B | A) P(A)

Contrary to the methods of rule-based systems, the updating method of Bayesian networks uses a global perspective, and if model and information are correct, it can be proved that the method computes the updated probabilities correctly (correctly regarding the axioms of the classical probability theory).

Any node in the network can receive information as the method doesn’t distinguish between inference in or opposite to the direction of the edges. Also, simultaneous input of information into several nodes will not affect the updating algorithm.

An essential difference between rule-based systems and systems based on Bayesian networks is that in rule based systems you try to model the expert’s way of reasoning (hence the name *expert systems*), while with Bayesian networks you try to model dependences in the domain itself. Systems of the latter type are often called *decision support systems* or *normative expert systems*.

Comparing Neural Networks and Bayesian Networks

The fundamental difference between the two types of networks is that a perceptron in the hidden layers does not in itself have an interpretation in the domain of the system, whereas all the nodes of a Bayesian network represent concepts that are well defined with respect to the domain.

The meaning of a node and its probability table can be subject to discussion, regardless of their function in the network. But it does not make any sense to discuss the meaning of the nodes and the weights in a neural network. Perceptrons in the hidden layers only have a meaning in the context of the functionality of the network.

This means that the construction of a Bayesian network requires detailed knowledge of the domain in question. If such knowledge can only be obtained through a series of examples (i.e., a data base of cases), neural networks seem to be an easier approach. This might be true in cases such as the reading of handwritten letters, face recognition, and other areas where the activity is a ‘craftsman like’ skill based solely on experience.

It is often criticized that in order to construct a Bayesian network you have to ‘know’ too many probabilities. However, there is not a considerable difference between this number and the number of weights and thresholds that have to be ‘known’ in order to build a neural network, and these can only be learnt by training. It is an enormous weakness of neural networks that you are unable to utilize the knowledge you might have in advance.

Probabilities, on the other hand, can be assessed using a combination of theoretical insight, empiric studies independent of the constructed system, training, and various more or less subjective estimates.

Finally, it should be mentioned that in the construction of a neural network the route of inference is fixed. It is decided in advance, about which relations information is gathered, and which relations the system is expected to compute. Bayesian networks are much more flexible in that respect.

3.2.2 Introduction to Bayesian Network

A *Bayesian network* (BN) is used to model a domain containing uncertainty in some manner. This uncertainty can be due to imperfect understanding of the domain, incomplete knowledge of the state of the domain at the time where a given task is to be performed, randomness in the mechanisms governing the behavior of the domain, or a combination of these.

Bayesian networks are also called *belief networks* and *Bayesian belief networks*. Previously, the term *causal probabilistic networks* has also been used. A BN is a network of nodes connected by directed links with a probability function attached to each node. The network (or graph) of a BN is a *directed acyclic graph* (DAG), i.e., there is no directed path starting and ending at the same node.

A node represents either a discrete random variable with a finite number of states or a continuous (Gaussian distributed) random variable. Throughout this document, the terms "variable" and "node" are used interchangeably. The links between the nodes represent (causal) relationships between the nodes.

If a node doesn't have any parents (i.e., no links pointing towards it), the node will contain a *marginal probability table*. If the node is discrete, it contains a probability distribution over the states of the variable that it represents. If the node is continuous, it contains a Gaussian density function (given through mean and variance parameters) for the random variable it represents.

If a node do have parents (i.e., one or more links pointing towards it), the node contains a *conditional probability table* (CPT). If the node is discrete, each cell in the CPT (or, in more general terms, the *conditional probability function* (CPF)) of a node contains a conditional probability for the node being in a specific state given a specific configuration of the states of its parents. Thus, the number of cells in a CPT for a discrete node equals the product of the number of possible states for the node and the product of the number of possible states for the parent nodes. If the node is continuous, the CPT contains a mean and a variance parameter for each configuration of the states of its discrete parents (one if there are no discrete parents) and a regression coefficient for each continuous parent for each configuration of the states of the discrete parents.

The following example tries to make all this more concrete.

The Apple Tree Example

The problem domain of this example is a small orchard belonging to Jack Fletcher (let's call him Apple Jack). One day Apple Jack discovers that his finest apple tree is losing its leaves. Now, he wants to know why this is happening. He knows that if the tree is dry (caused by a drought) there is no mystery - it is very common for trees to lose their leaves during a drought. On the other hand the losing of leaves can be an indication of a disease.

The situation can be modeled by the BN in Figure 1. The BN consists of three nodes: Sick, Dry, and Loses which can all be in one of two states: Sick can be either "sick" or "not" - Dry can be either "dry" or "not" - and Loses can be either "yes" or "no". The node Sick tells us that the apple tree is sick by being in state "sick". Otherwise, it will be in state "not". The nodes Dry and Loses tell us in the same way if the tree is dry and if the tree is losing its leaves, respectively.

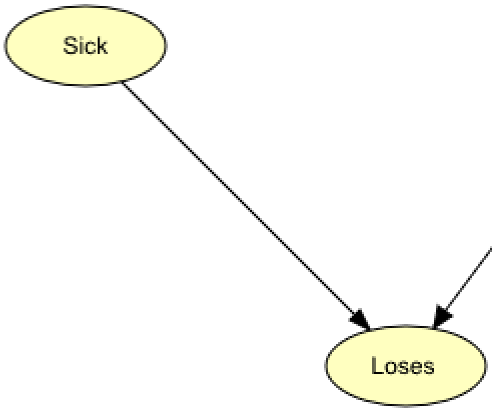


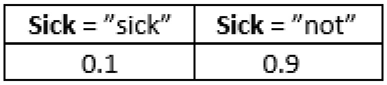
Figure 1: BN representing the domain of the Apple Jack problem.

The BN in Figure 1 models the *causal dependence* from Sick to Loses and from Dry to Loses. This is represented by the two links.

When there is a causal dependence from a node A to another node B, we expect that when A is in a certain state this has an impact on the state of B. One should be careful when modeling the causal dependences in a BN. Sometimes it is not quite obvious in which direction a link should point. In our example, for instance, we say that there is a causal link from Sick to Loses because when a tree is sick this might cause the tree to lose its leaves. But couldn’t one say that when the tree loses its leaves it might be sick and then turn the link in the other direction? No, we cannot! It is the sickness that causes the lost leaves and not the lost leaves that cause the sickness.

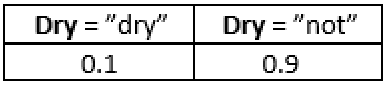
In Figure 1, we have the graphical representation of the BN. However, this is only what we call the *qualitative representation* of the BN. Before we can call it a BN, we need to specify the *quantitative representation*.

The quantitative representation of a BN is the set of CPTs of the nodes. Tables 1, 2, and 3 show the CPTs of the three nodes in the BN of Figure 1.



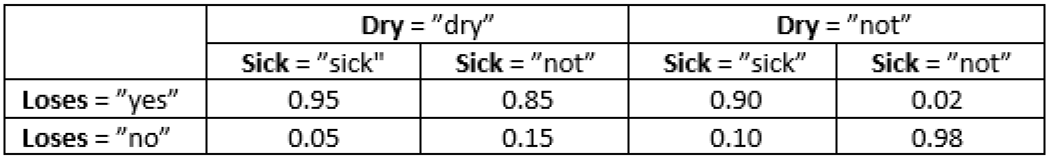
Sick = ”sick”	**Sick** = ”not”
0.1	0.9

Tabel 1: P(Sick)



Dry = ”dry”	**Dry** = ”not”
0.1	0.9

Tabel 2: P(Dry)



	Dry = ”dry”		**Dry** = ”not”	
	Sick = "sick"	**Sick** = ”not”	**Sick** = ”sick”	**Sick** = ”not”
Loses = ”yes”	0.95	0.85	0.90	0.02
Loses = ”no”	0.05	0.15	0.10	0.98

Tabel 3: P(Loses | Sick, Dry)

Note that all three tables show the probability of a node being in a specific state depending on the states of its parent nodes but since Sick and Dry do not have any parent nodes, the distributions in Tables 1 and 2 are not conditioned on anything.

BNs that are concerned with the causal relations between variables at a given instance, such as the one described above, are sometimes known as *Static Bayesian Networks* (SBNs). SBNs are concerned only with the current situation and do not explicitly model temporal sequences, i.e. the past is ignored and the future is not predicted. For example, in Figure 2, there are two diseases (D1 and D2) which can cause different symptoms (S1 and S2). Using the information at hand on the symptoms makes it possible to predict the probabilities of each disease.

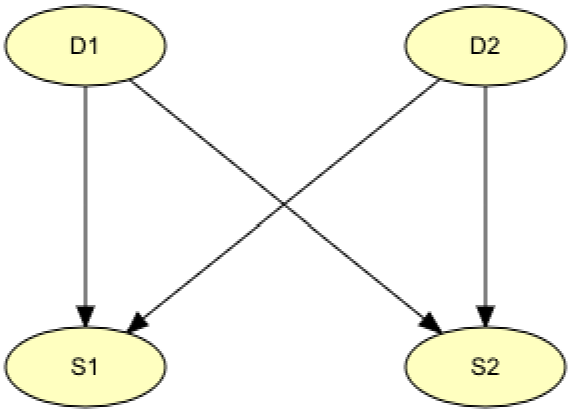


Figure 2: An example of Static Bayesian Network (SBN).

In many problem domains, such as the medical situation considered above, it is almost inconceivable to represent data and reason about them without using a temporal dimension, since things evolve through time. SBNs, like the one represented in Figure 2, can't be used for such systems and thus the network has to be expanded to include temporal information. Such networks are known as *Dynamic Bayesian Networks* (DBN). The simplest way to extend an SBN into a DBN is by including multiple instances (time slices) of the SBN and linking these together. For example, the network in Figure 3 is obtained by linking multiple instances of the network in Figure 2.

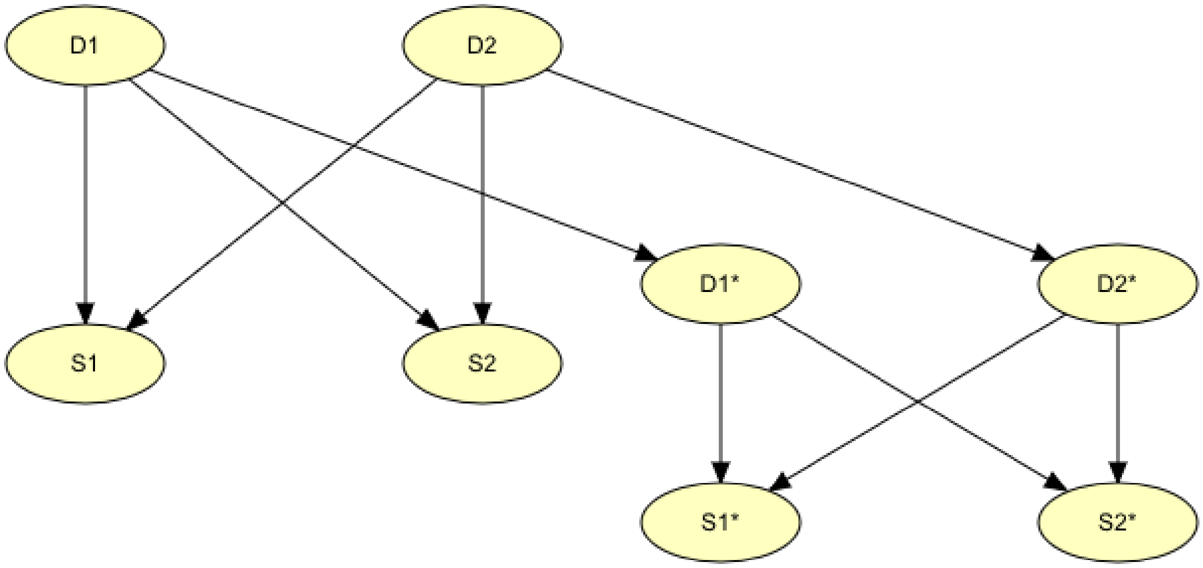


Figure 3: An example of Dynamic Bayesian Network (DBN).

The existence of a disease today will have an effect on whether or not the disease will be there tomorrow. Therefore there should be a link between the nodes representing “disease today” (nodes D1 and D2) and “disease tomorrow” (nodes D1* and D2*). Using this new network, it is possible to predict the progress of the diseases.

What was shown in the above examples is a description of how to construct very simple BNs. When we have constructed a network, we can use it for entering evidence in some of the nodes where the state is known and then retrieve the new probabilities computed in other nodes given this evidence. In the Apple Tree example, suppose we know that the tree is losing its leaves. We then enter this evidence by selecting the state “yes” in the Loses node. Then we can read the probability of the tree being sick as the probability of the node Sick being in state “sick” and the probability of the tree being dry as the probability of the node Dry being in state “dry”.

Computing the probabilities of other variables given some evidence, like the situations described above is known as *Belief Updating*. Another piece of information that might be interesting to find is the most likely global assignment of the states of all random variables given some evidence. This is know as *Belief Revision*.

HUGIN provides you with a tool to construct such networks. After constructing the BNs, you can do belief revision, belief updating, and much more. If you are in the process of learning more about the HUGIN Development Environment, now would be a good time to go through the *How to Build BNs* (page 26) tutorial. Here, the Apple Tree BN is constructed using the HUGIN Graphical User Interface. You can also go on reading the *Introduction to Object-Oriented Networks* (page 51); for example, object-oriented networks are very helpful when constructing networks with repetitive structures like in the diseases network above. Or you might wish to go on reading the *Introduction to Influence Diagrams (IDs)* (page 45); IDs are BNs extended with utility nodes and decision nodes.

Definition of Bayesian Networks

Formally, a Bayesian network can be defined as follows:

> A *Bayesian network* is a pair (G,P), where G=(V,E) is a *directed acyclic graph* (DAG) over a finite set of nodes (or vertices), V, interconnected by directed links (or edges), E, and P is a set of (conditional) probability distributions. The network has the following property:
>
> - Each node representing a variable A with parent nodes representing variables B_1, B_2,…, B_n (i.e., B_i, → A for each i=1,…,n) is assigned a conditional probability table (CPT) representing P(A | B_1, B_2, …, B_n).

The nodes represent random variables, and the links represent probabilistic dependences between variables. These dependences are quantified through a set of conditional probability tables (CPTs): Each variable is assigned a CPT of the variable given its parents. For variables without parents, this is an unconditional (also called a marginal) distribution.

Conditional Independence

An important concept for Bayesian networks is *conditional independence*. Two sets of variables, *A* and *B*, are said to be (conditionally) independent given a third set *C* of variables if when the values of the variables *C* are known, knowledge about the values of the variables B provides no further information about the values of the variables *A*:

$$P(A|B,C) = P(A|C)$$

Conditional independence can be directly read from the graph as follows: Let *A*, *B*, and *C* be disjoint sets of variables, then

- identify the smallest sub-graph that contains $A \cup B \cup C$ and their ancestors;
- add undirected edges between nodes having a common child;
- drop directions on all directed edges.

Now, if every path from a variable in A to a variable in *B* contains a variable in *C*, then A is conditionally independent of *B* given *C* *(Lauritzen et al. 1990)* (page 541).

To illustrate this concept, let us consider the following fictitious piece of fictitious medical knowledge

> "Shortness-of-breath (dyspnoea) [d] may be due to tuberculosis [t], lung cancer [l] or bronchitis [b], or none of them, or more than one of them. A recent visit to Asia [a] increases the risk of tuberculosis, while smoking [s] is known to be a risk factor for both lung cancer and bronchitis. The result of a single chest X-ray [x] does not discriminate between lung cancer and tuberculosis, neither does the presence or absence of dyspnoea" *(Lauritzen & Spiegelhalter 1988)* (page 541).

The last fact is represented in the graph by the intermediate variable *e*. This variable is a logical-or of its two parents (*t* and *l*); it summarizes the presence of one or both diseases or the absence of both.

Figure 4 shows a model for the knowledge.

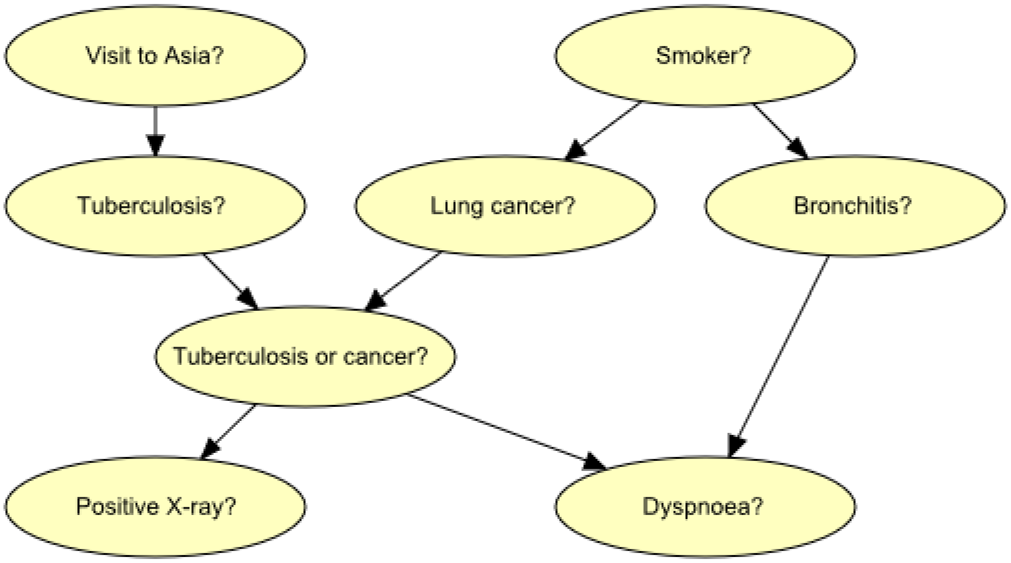


Figure 4: Graph representing structural aspects of medical knowledge concerning lung diseases.

If we learn the that a patient is a smoker, we will adjust our beliefs (increased risks) regarding lung cancer and bronchitis. However, our beliefs regarding tuberculosis are unchanged (i.e., t is conditionally independent of s given the empty set of variables). Now, suppose we get a positive X-ray result for the patient. This will affect our beliefs regarding tuberculosis and lung cancer, but not our beliefs regarding bronchitis (i.e., b is conditionally independent of x given s). However, had we also known that the patient suffers from shortness-of-breath, the X-ray result would also have affected our beliefs regarding bronchitis (i.e., b is not conditionally independent of x given s and d).

These (in)dependences can all be read from the graph of Figure 1 using the method described above.

Another, equivalent method to determine conditional independence is *d-separation* (page 289)., due to *Pearl (1988)* (page 541)..

Inference

Inference in a Bayesian network means computing the conditional probability for some variables given information (evidence) on other variables.

This is easy when all available evidence is on variables that are ancestors of the variable(s) of interest. But when evidence is available on a descendant of the variable(s) of interest, we have to perform inference opposite the direction of the edges. To this end, we employ *Bayes' Theorem*:

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

HUGIN inference is essentially a clever application of Bayes' Theorem; details can be found in the paper by *Jensen et al. (1990(1))* (page 541)..

Networks with Conditional Gaussian Variables

The HUGIN Decision Engine is able to handle networks with both discrete and continuous random variables. The continuous random variables must have a *Gaussian* (also known as a normal) distribution conditional on the values of the parents.

The distribution for a continuous variable Y with discrete parents I and continuous parents Z is a (one-dimensional) Gaussian distribution conditional on the values of the parents:

$$P(Y|I=i, Z=z) = \mathcal{N}(\alpha(\rangle) + \beta(\rangle)^{\tau}\ddagger, \gamma(\rangle))$$

Note that the mean depends linearly on the continuous parent variables and that the variance does not depend on the continuous parent variables. However, both the linear function and the variance are allowed to depend on the discrete parent variables. These restrictions ensure that exact inference is possible.

Note that discrete variables cannot have continuous parents.

Figure 5 shows a network model for a waste incinerator (*Lauritzen 1992* (page 541)):

> "The emissions (of dust and heavy metals) from a waste incinerator differ because of compositional differences in incoming waste [*W*]. Another important factor is the waste burning regimen [*B*], which can be monitored by measuring the concentration of CO2 in the emissions [*C*]. The filter efficiency [*E*] depends on the technical state [*F*] of the electro filter and on the amount and composition of waste [*W*]. The emission of heavy metals [*Mo*] depends on both the concentration of metals [*Mi*] in the incoming waste and the emission of dust particulates [*D*] in general. The emission of dust [*D*] is monitored through measuring the penetrability of light [*L*]."

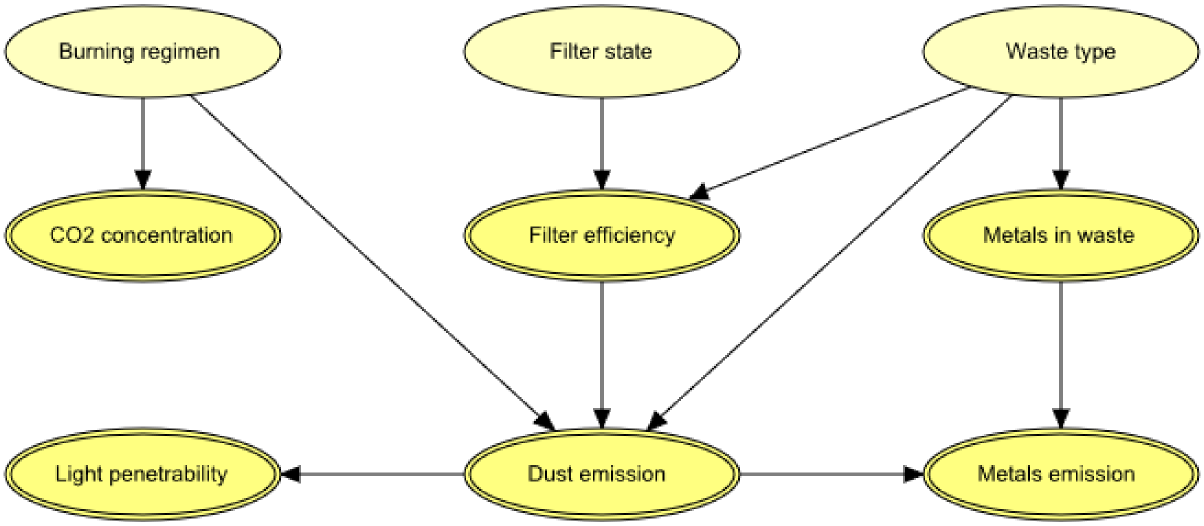


Figure 5: The structural aspects of the waste incinerator model: *B*, *F*, and *W* are discrete variables, while the remaining variables are continuous.

The result of inference within a network model containing conditional Gaussian variables is - as always - the beliefs (i.e., marginal distributions) of the individual variables given evidence. For a discrete variable this amounts to a probability distribution over the states of the variable. For a conditional Gaussian variable two measures are provided:

1. the mean and variance of the distribution;
2. since the distribution is in general not a simple Gaussian distribution, but a mixture (i.e., a weighted sum) of Gaussians, a list of the parameters (weight, mean, and variance) for each of the Gaussians is available.

From the network shown in Figure 5 (and given that the discrete variables *B*, *F*, and *W* are all binary), we see that

- the distribution for *C* can be comprised of up to two Gaussians (one if *B* is instantiated);
- initially (i.e., with no evidence incorporated), the distribution for *E* is comprised of up to four Gaussians;
- if L is instantiated (and none of *B*, *F*, or *W* is instantiated), then the distribution for *E* is comprised of up to eight Gaussians.

See also the section on Gaussian distribution functions for details on how to specify a conditional Gaussian distribution function using the HUGIN Graphical User Interface.

To learn how to build a Bayesian network using the HUGIN Graphical User Interface, please consult the tutorial *How to Build BNs* (page 26).

3.2.3 How to Build a Bayesian Network

This tutorial shows you how to implement a small Bayesian network in the Hugin Graphical User Interface. The network we are about to implement is the one modeled in the Apple Tree example in the *Bayesian Networks Tutorial* (page 19).

The qualitative representation of our network is shown in Figure 1.

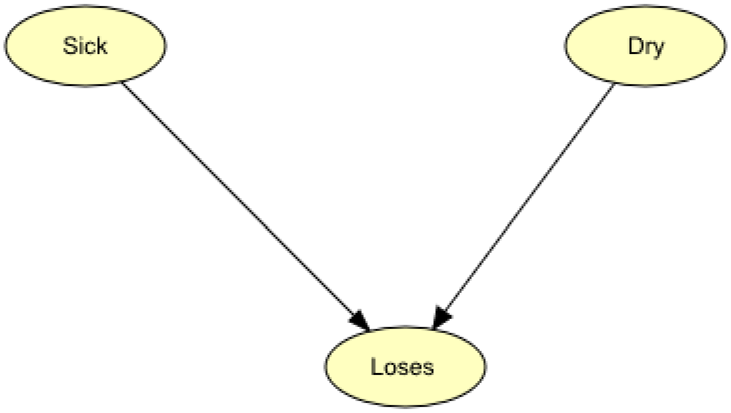


Figure 1: Bayesian network representing the Apple Tree problem.

If you want to understand the design of this network, you should read about it in the *Bayesian Networks Tutorial* (page 19).

Constructing a New Network

When you choose to start up the HUGIN Graphical User Interface, the *Main HUGIN Window* (or simply the *Main Window*) opens. This window contains a menu bar (called the Main Window Menu Bar), a tool bar (called the Main Window Tool Bar), and a document pane (called the Main Window Document Pane or simply the Document Pane). In the Document Pane, a new empty network called “unnamed1” is automatically opened in a *network window* (see Figure 2). It starts up in *Edit Mode* which allows you to start constructing the network immediately (the other main mode is *Run Mode* which allow you to use the network).

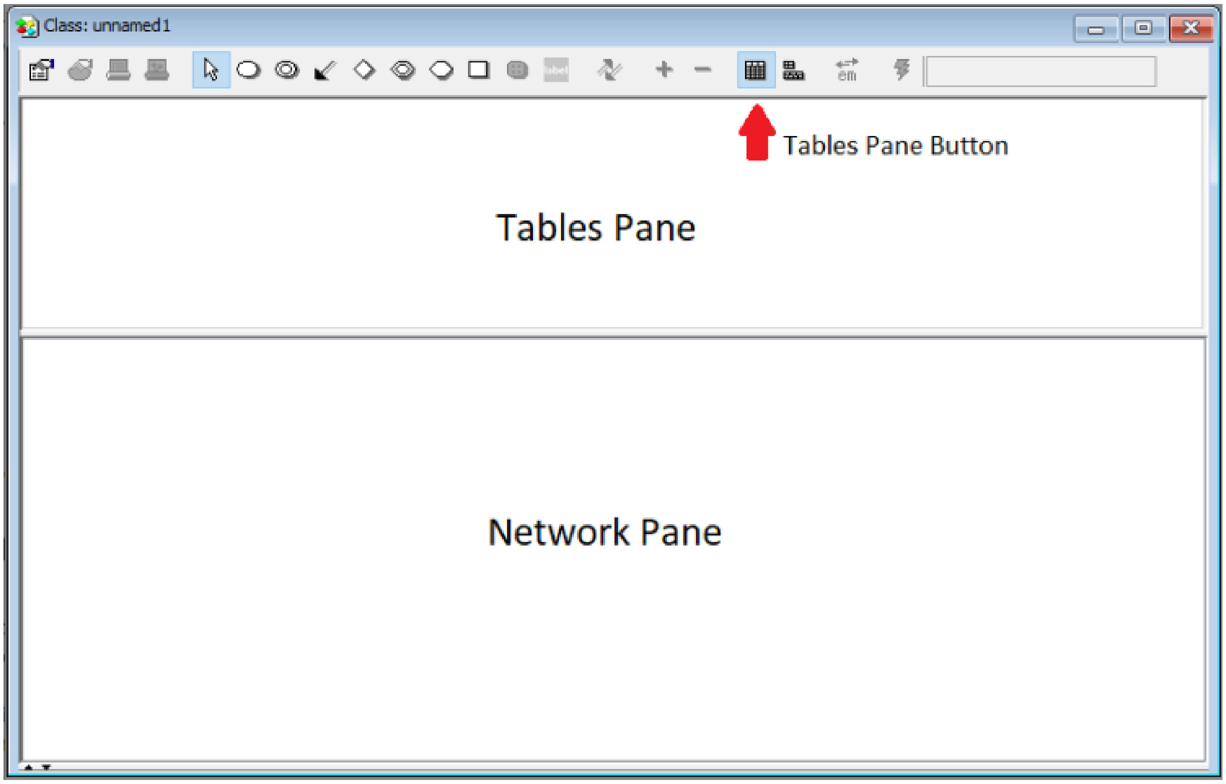


Figure 2: The network window containing a *Tool Bar*, a *Tables Pane*, and a *Network Pane*.

Adding Nodes

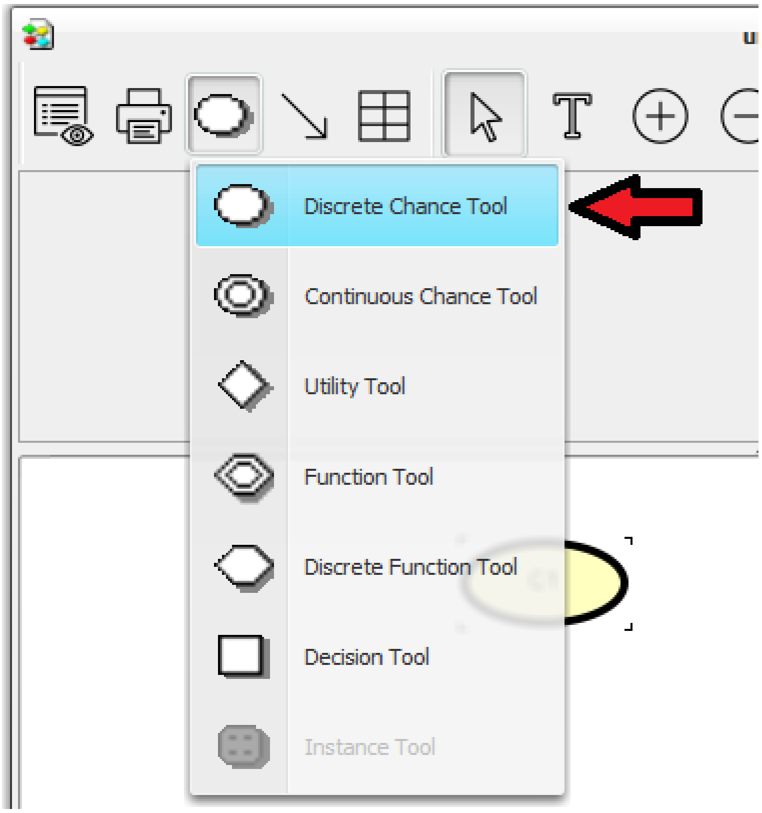
The first thing we will do is add the Sick node. This can be done as follows:

- Select the *Discrete Chance Tool* in the Tool Bar of the "unnamed1" network window (see Figure 3).
- Click somewhere in the *Network Pane* (see Figure 2).

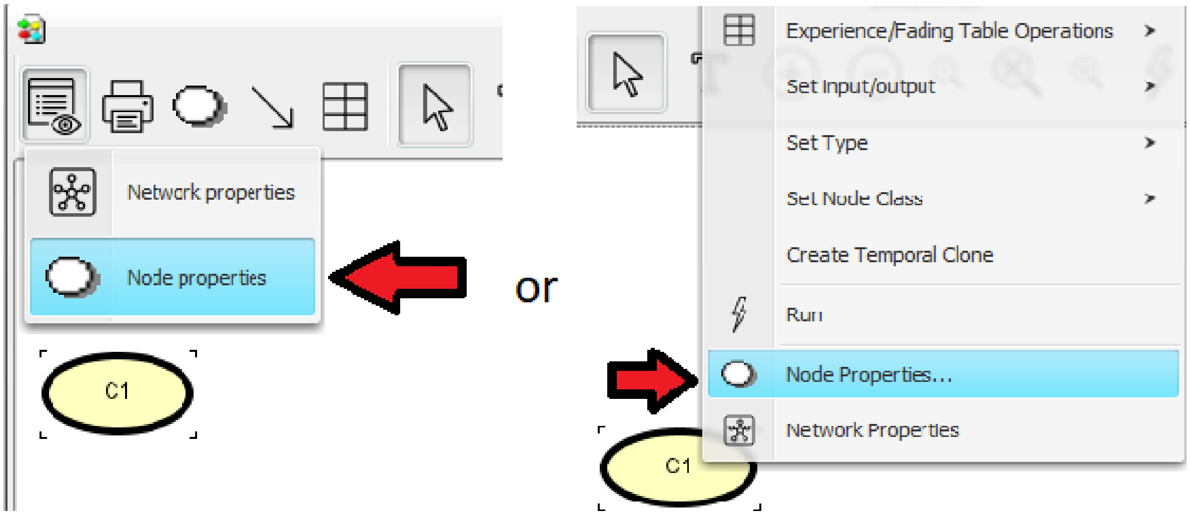
When we have clicked in the Network Pane, a node labeled "C1" appears. We want to change this label to "Sick":

- Select the node with the mouse cursor.
- Enter "Node Properties" by pressing the *node properties tool* (see Figure 3).
- Change both the "Name" and the "Label" fields to "Sick".
- Press the "OK" button.

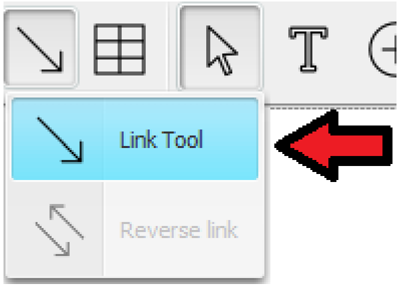
The "Name" is the internal name of the node while "Label" is the label of the node. If no label is specified (as was the case before we changed the label) the label used is the internal name. The internal name can consist of only the letters 'a'-'z' and 'A'-'Z', the digits '0'-'9', and the underscore character '_' while the label can be almost anything. Please note that the first character of the name must be a letter.

- 

: The *Discrete Chance tool*

- 

: The *Node Properties tool*

- 

: The *Link tool*

The Dry and Loses nodes are added the same way. We can add more nodes without having to press the Discrete Chance

Tool all the time by holding down the SHIFT key while clicking in the Network Pane. When we have chosen a node in the Network Pane, we can access the node properties tool by holding down the right mouse button.

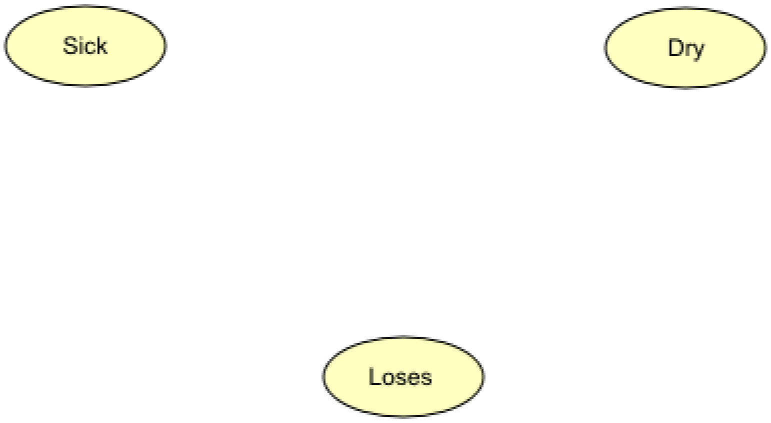


Figure 4: The Network pane contains the three nodes Sick, Dry, and Loses that have been added to the network.

Adding Links

Now, we have a network similar to the one shown in the Network Pane in Figure 4. To add the links from Sick to Loses and from Dry to Loses, do as follows:

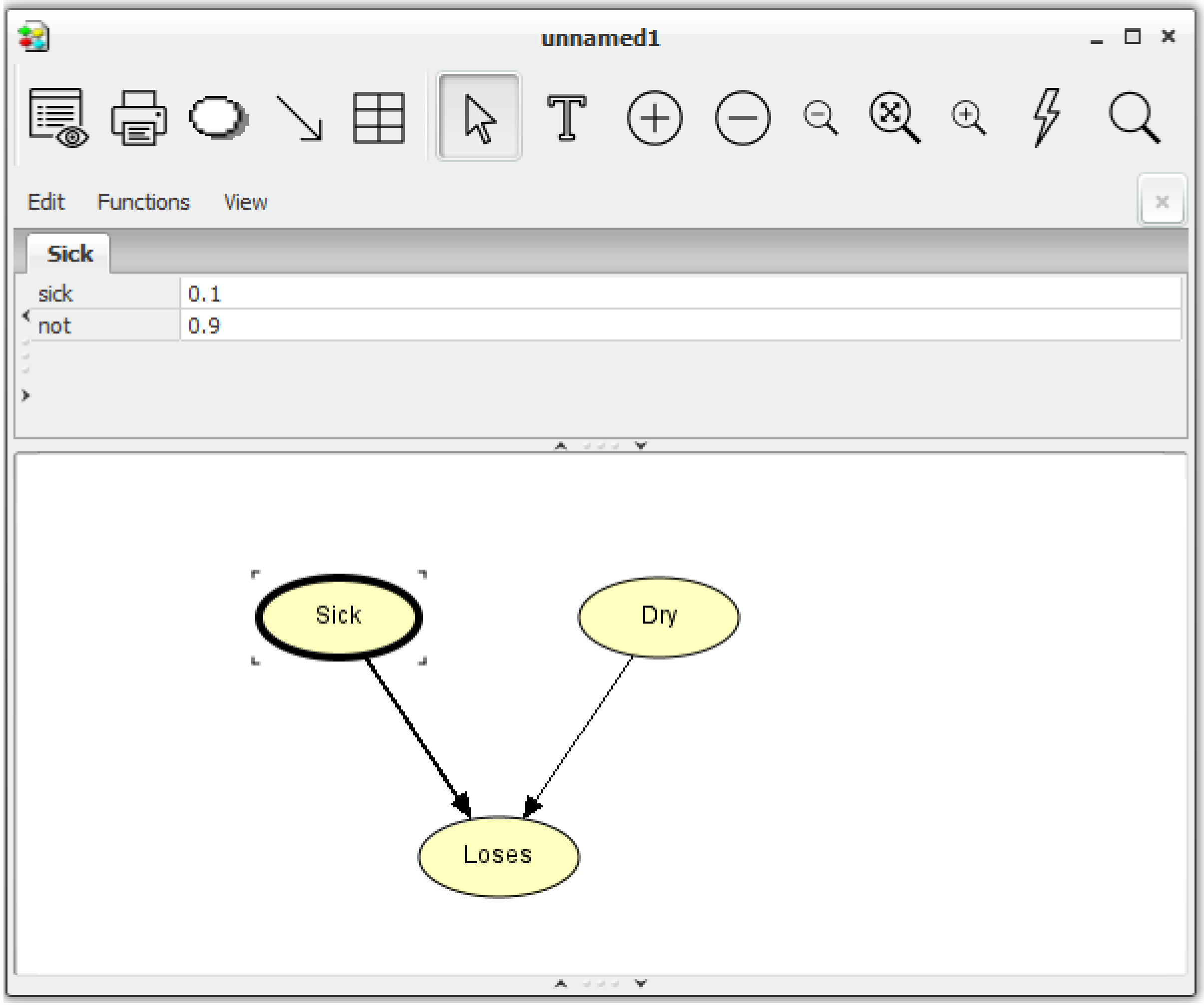
- Press the *Link Tool* (see Figure 3).
- Drag a link from Sick to Loses with the left mouse button while holding down the SHIFT key. The SHIFT key ensures that we can add more links without having to press the Link Tool again.
- Drag a link from Dry to Loses with the left mouse button.

What we have now is the complete qualitative representation which is similar to the one in Figure 1. The next step will be to specify the states and the conditional probability table (CPT) of each node.

The States

In the *introduction to BNs* (page 19) the states of the nodes were specified as follows: Sick has two states: “sick” and “not”, Dry has two states: “dry” and “not”, and **Loses** has two states “yes” and “no”.

First, we open the Tables Pane by clicking the tables-pane button.



Sick	
sick	0.1
not	0.9

Figure 5: The CPT is opened by pressing the left mouse button over a node, while holding down the “ctrl” button.

Next, we specify the states of Sick:

- Hold down the “ctrl” button while clicking the left mouse button over the node “Sick” to display its CPT in the Tables Pane, as shown in Figure 5.
- Click the field containing the text “State 1” in the CPT in the Tables Pane.
- Type the text “sick” in the field to give the state this name.
- Click the field containing the text “State 2” in the CPT.
- Type the text “not” in the field.

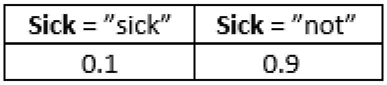
Now, do the same with Dry.

We can do exactly the same with Loses. Beware that the CPT of Loses is a little bigger than those of Sick and Dry. This is just because Loses has parent nodes (Sick and Dry don’t).

- Name the two states of Loses “yes” and “no”.

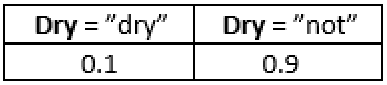
Entering CPT Values

The next step is to enter the CPT values correctly (as default, the Hugin Graphical User Interface has given all nodes a uniform distribution). The values were specified in the *introduction to BNs* (page 19) and they are shown in Tables 1, 2, and 3.



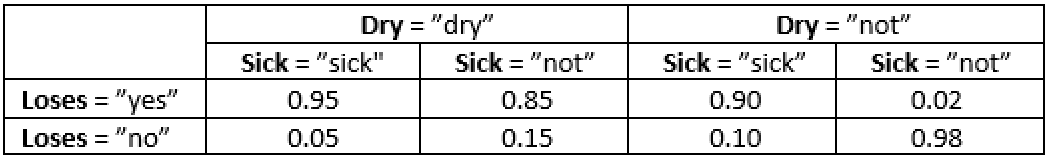
Sick = ″sick″	**Sick** = ″not″
0.1	0.9

Tabel 1: P(Sick)



Dry = ″dry″	**Dry** = ″not″
0.1	0.9

Tabel 2: P(Dry)



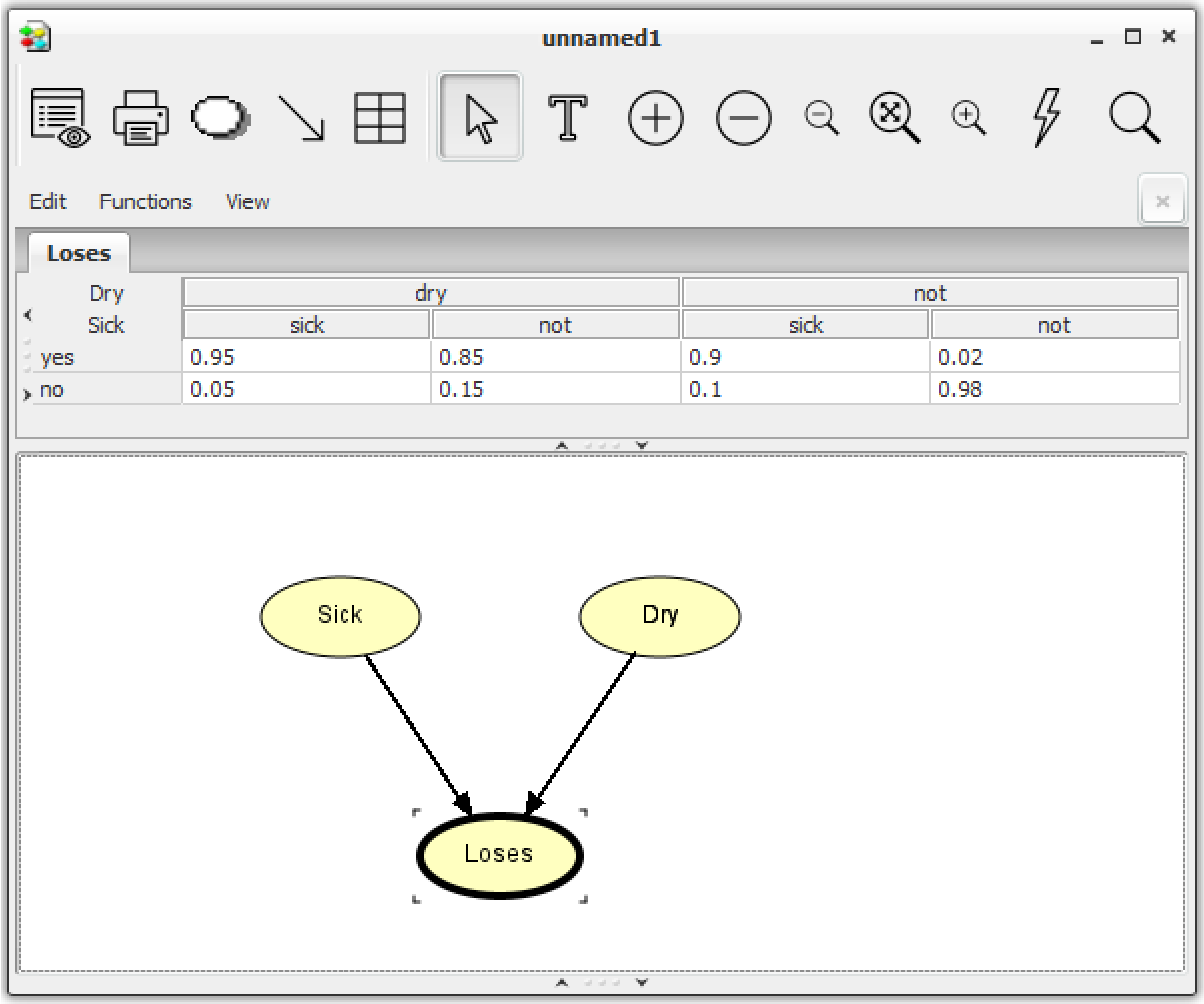
	Dry = ″dry″		**Dry** = ″not″	
	Sick = ″sick"	**Sick** = ″not″	**Sick** = ″sick″	**Sick** = ″not″
Loses = ″yes″	0.95	0.85	0.90	0.02
Loses = ″no″	0.05	0.15	0.10	0.98

Tabel 3: P(Loses | Sick, Dry)

First, we select all three nodes (shortcut: Ctrl+A) to get the CPTs displayed in the Tables Pane. Next, we enter the values into the Sick node:

- Click the field representing Sick=”sick”.
- Enter the value 0.1 (from Table 1).
- Click the field representing Sick=”not”.
- Enter the value 0.9 (from Table 1).

Enter the values for Dry and Loses the same way. When you have entered the CPT for Loses, the network window should look like Figure 6.



Dry	dry		not	
Sick	sick	not	sick	not
yes	0.95	0.85	0.9	0.02
no	0.05	0.15	0.1	0.98

Figure 6: The network window with node Loses selected. The CPT of Loses appears in the Tables Pane.

This completes the construction of the network. At this point it would be a good idea to save the network. Here is how to do it:

- Select “Save” (or “Save As”) from the “File” menu.
- Enter a name (e.g. “apple”).
- Press “Save”.

Compiling the Network

Now, let’s compile the network and see how it works:

- Press the Run Mode tool button in the Tool Bar (see Figure 7).

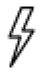


Figure 7: The Run Mode tool button.

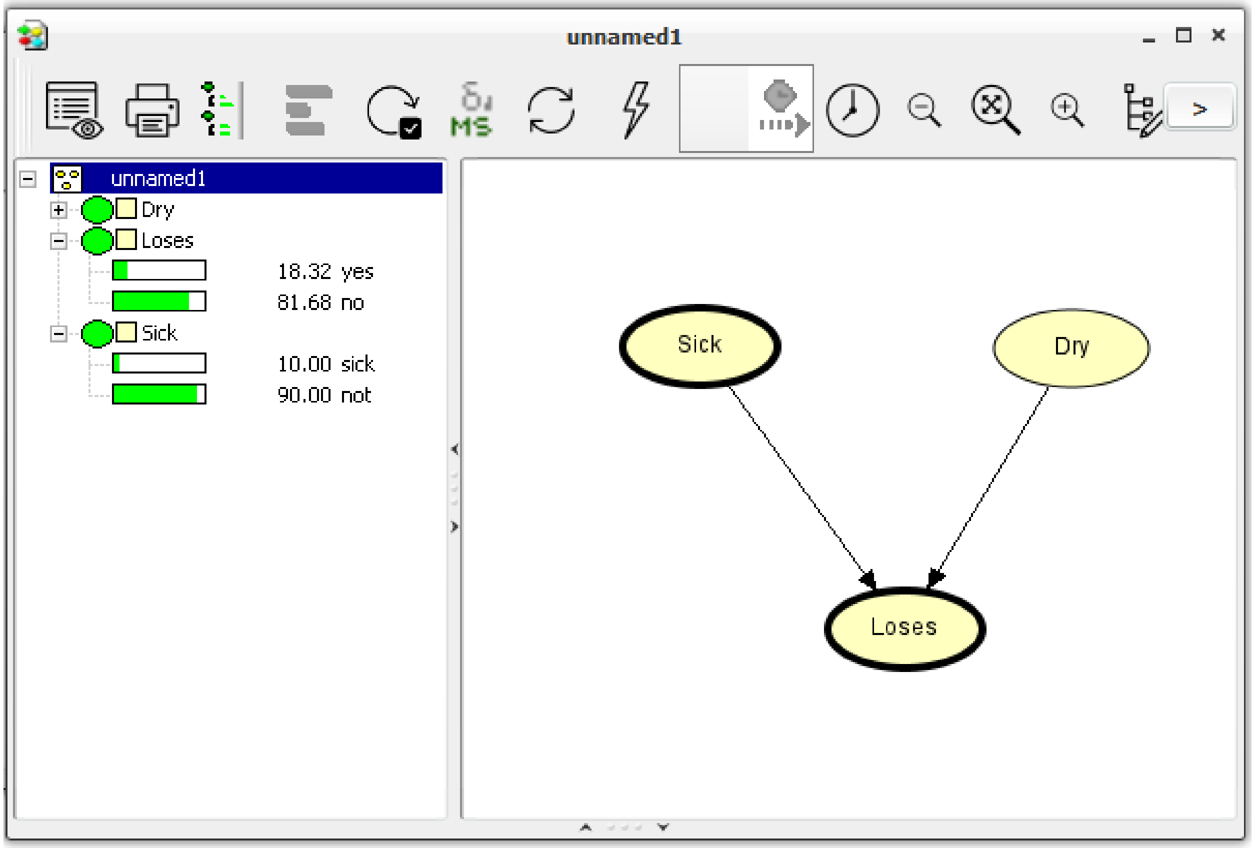
The compiler checks for the following errors:

- Cycles. There must be no directed cycles in a network.
- For each parent configuration of a node the probabilities of the different states must have the sum of 1. In other words, each column of the table must sum to 1. If there is a column that does not sum to 1, the compiler will normalize the values. This fact can be utilized when filling in the probabilities. Say, for example, that the probability of a tree being sick is based on the observation of 13527 trees over one season, where 1678 got sick and the rest didn't. Instead of first computing the fractions, we just put 1678 in the sick state of the Sick node, and 11849 in the no state. Then the compiler will compute the proper values.

If you have done exactly as this tutorial told you, there should not be any errors in the compilation process. The compilation should be finished very fast with a small network like ours. After the compilation, the Run Mode is entered (we have so far only been working in Edit Mode).

Running the Network

Running in Run Mode, the network window is split into two by a vertical bar (see Figure 8). To the left is the Node List Pane and to the right is the Network Pane.



unnamed1
Dry
Loses
18.32 yes
81.68 no
Sick
10.00 sick
90.00 not

Figure 8: The network window in Run Mode. To the left is the Node List Pane (having **Loses** and **Sick** expanded) and to the right is the Network Pane.

We can view the probabilities of a node being in a certain state by expanding the node in the Node List Pane. We expand (collapse) a node by clicking its expand (collapse) icon in the Node List Pane, by double-clicking its node symbol in the Node List Pane, or by selecting (deselecting) it in the Network Pane. We can also expand (collapse) all nodes at once by pressing the *expand (collapse) node list tool* in the Tool Bar just to the right of the node properties tool.

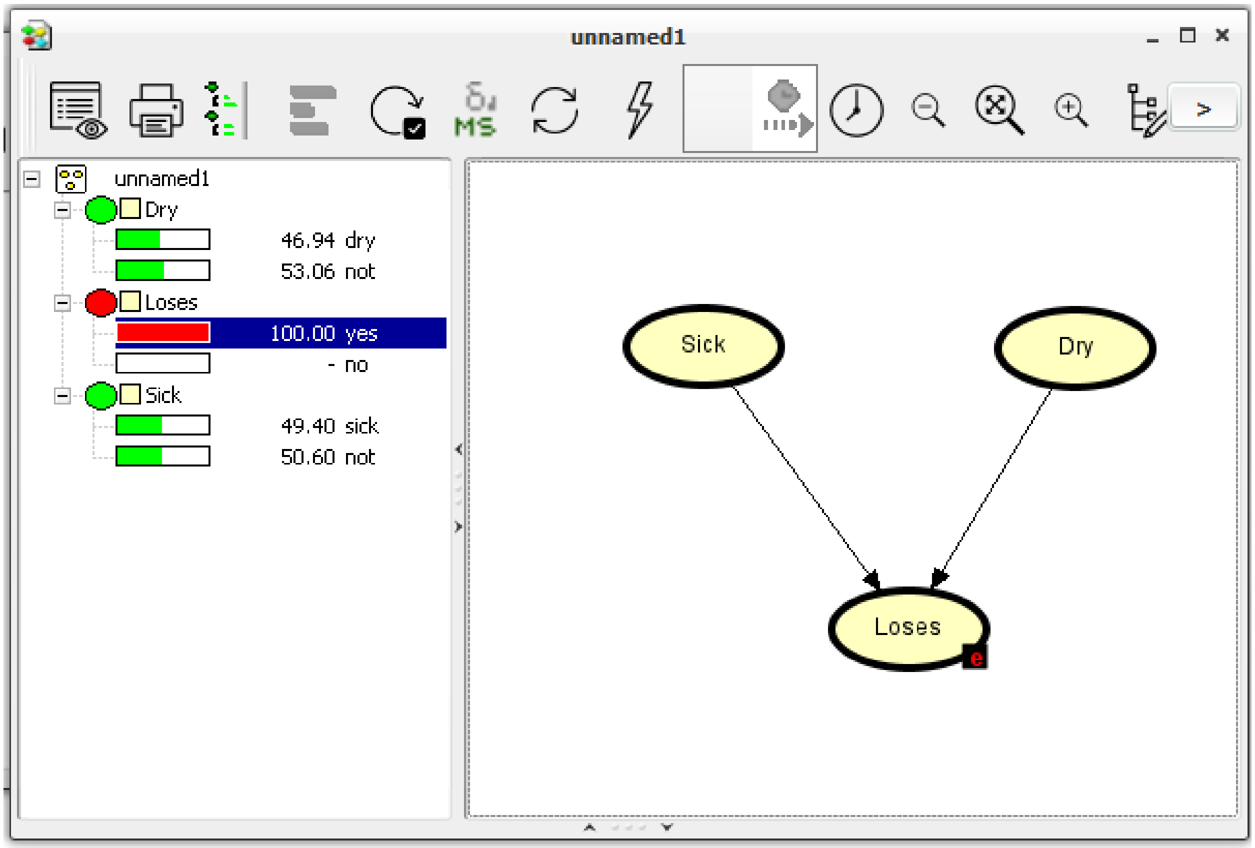
Is the Tree Sick?

Now, imagine that we want to use our network to find the probability of an apple tree being sick given the information that the tree is losing its leaves. This is done as follows:

- Expand all nodes (by pressing the expand node list tool).
- Enter the fact that the tree is losing its leaves by double clicking the state "yes" of the **Loses** node.
- Propagate this piece of evidence by pressing the *Sum Propagation Tool* in the Tool Bar (see Figure 9).
- Read the probability of **Sick** being in state "sick"

Figure 9: The Sum Propagation Tool.

This should give the output shown in Figure 10.



```
unnamed1
  Dry
    46.94 dry
    53.06 not
  Loses
    100.00 yes
         - no
  Sick
    49.40 sick
    50.60 not
```

Figure 10: Our network after the evidence that the tree is losing its leaves has been entered and propagated.

The probability of the tree being sick is now 0.49.

If you do not read the value specified above, you have probably mistyped something when filling in the CPTs. Then please check the CPTs of all the nodes.

The Monitor Windows

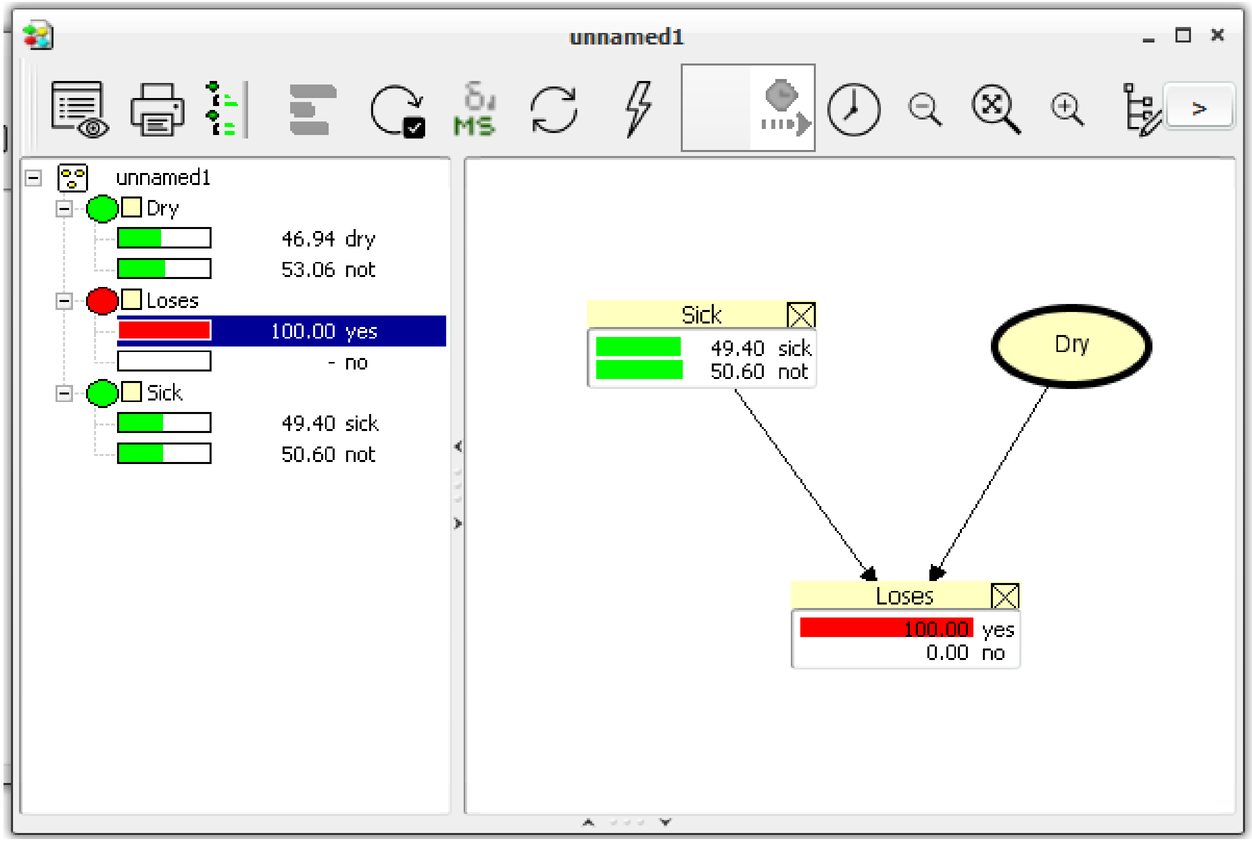
In the last section, we used the Node List Pane to enter evidence and retrieve beliefs. We can also do this by using the *monitor windows*. The monitor windows show the same information as the Node List Pane but we have the opportunity to place the monitor windows near the corresponding nodes of the network in the Network Pane. We can open a monitor window for each node in the Network Pane, but the best way to use them is probably only to open a monitor window for the nodes in the network which have special interest. Otherwise, they might take up too much space.

Now, we shall open monitor windows for Sick and Loses and repeat the computations from before. First, initialize the network:

- Press the Initialize Tool button (to the left of the Sum Propagation Tool).

Then, we are ready to open the monitor windows of Sick and loses.

- Select Sick and Loses (hold down the SHIFT key to select more nodes at the same time).
- Choose "Show Monitor Windows" from the "View" menu.



unnamed1

Dry
46.94 dry
53.06 not

Loses
100.00 yes
- no

Sick
49.40 sick
50.60 not

Sick
49.40 sick
50.60 not

Dry

Loses
100.00 yes
0.00 no

Figure 11: Monitor windows of Sick and loses shown in the Network Pane.

The rest of this tutorial introduces some very useful aspects of the HUGIN Graphical User Interface, but it can be skipped.

The Most Likely Combination

From the propagation in the previous section we could see that the probability of the apple tree suffering from drought is 0.47. In both the case of Sick and Dry it is more likely that the state is “not”. This could make one believe that the most likely combination of states is when both Sick and Dry are in state “not”. However, this is a wrong conclusion. If we want to find the most likely combination of states in all nodes, we should use max propagation (in stead of sum propagation). The *Max Propagation Tool* is found in the Tool Bar just to the right of the Sum Propagation Tool.

Now, try to press the Max Propagation Tool. In each node, a state having the value 100.00 belongs to a most likely combination of states. In this case, this gives one unique combination being the most likely: Sick is “sick” and Dry is “not”

We see that even if Sick=”sick” is less likely than Sick=”not”, Sick=”sick” is contained in the most likely combination of the states of the nodes while Sick=”not” is not. This shows that we need to be careful in making conclusions from the result of a propagation.

Now, one might want to know the probability of this most likely combination of states (or of any other combination of states) under the assumption that the entered evidence holds.

Computing the Probability of a Combination of States

Here, we shall describe a technique to compute the probability of the most likely combination of states given the evidence that the apple tree is losing its leaves. This probability is written:

$$P(Sick = ”yes”, Dry = ”not”|Loses = ”yes”)$$

Each time we perform sum propagation in a network, the probability of the entered evidence is shown in the lower left corner of the HUGIN Graphical User Interface window (the P(All) value). If we have chosen the “yes” state of the Loses node and performed sum propagation, we can read the probability of Loses=”yes” (written P(Loses=”yes”)). This value should be 0.1832.

The technique uses the following rule from probability theory (known as the fundamental rule):

$$P(A, B) = P(A|B)P(B)$$

The only kind of probability we can get from HUGIN is the probability of a series of pieces of evidence which can be written in the form:

$$P(A1, A2, ..., An)$$

We use the fundamental rule to rewrite our requested probability to some expression composed by such components:

$$\begin{aligned} &P(Sick = ”yes”, Dry = ”not”|Loses = ”yes”) \\ = &P(Sick = ”sick”, Dry = ”not”, Loses = ”yes”)/P(Loses = ”yes”) \end{aligned}$$

In the fundamental rule, we have divided both sides with P(B). Then we have substituted A with Sick=”yes”, Dry=”not” and B with Loses=”yes”.

We already know P(Loses=”yes”) so we only need to compute P(Sick=”sick”, Dry=”not”, Loses=”yes”). This is done as follows:

- Enter Sick=”sick”, Dry=”not”, and Loses=”yes” in the network.
- Press the Sum Propagation Tool.
- Read P(Sick=”sick”, Dry=”not”, Loses=”yes”) as the P(All) value in the lower left corner.

This value should be 0.081. Now, we are ready to compute the requested probability:

$$\begin{aligned} P(Sick = "yes", Dry = "not" | Loses = "yes") \\ = 0.081/0.1832 \\ = 0.442 \end{aligned}$$

So, the probability of the most likely combination of states of Sick and Dry, given that Loses="yes", is 0.442.

3.2.4 Introduction to (Limited Memory) Influence Diagrams

Introduction to (Limited Memory) Influence Diagrams A (limited memory) influence diagram (ID) may be used instead of a Bayesian network (BN) if you wish to explicitly model various decision alternatives and the utilities associated with these. To some extent it is possible to construct a model for decision making with a pure BN, but the concepts of utility and decisions are not explicitly covered. An influence diagram is simply a BN, extended with utility nodes and decision nodes. Having these two new types of nodes, we also need to have a name for the old node type. We shall call these nodes chance nodes. We shall present the concept of influence diagrams by extending the BN constructed in the apple tree example from the Introduction to BNs.

The Apple Tree Example

Again, we consider the Apple Tree Example (see Figure 1).

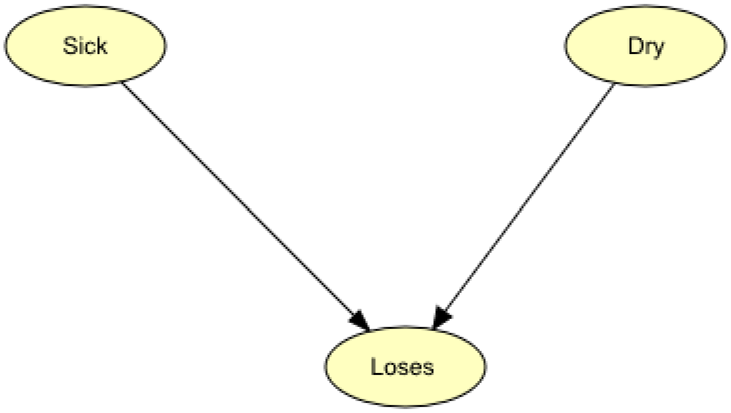


Figure 1: The BN constructed in the Introduction to BNs

Apple Jack now wants to decide whether or not to invest resources in giving the tree some treatment. Since this involves a decision through time, we have to modify the BN into a dynamic one as described in the Introduction to BNs *Introduction to BNs* (page 19) section. We first add three nodes very similar to those already in the network. The new nodes Sick', Dry', and Loses' represent the same as the old nodes - except that they represent the situation at the time of harvest. These nodes have been added in Figure 2.

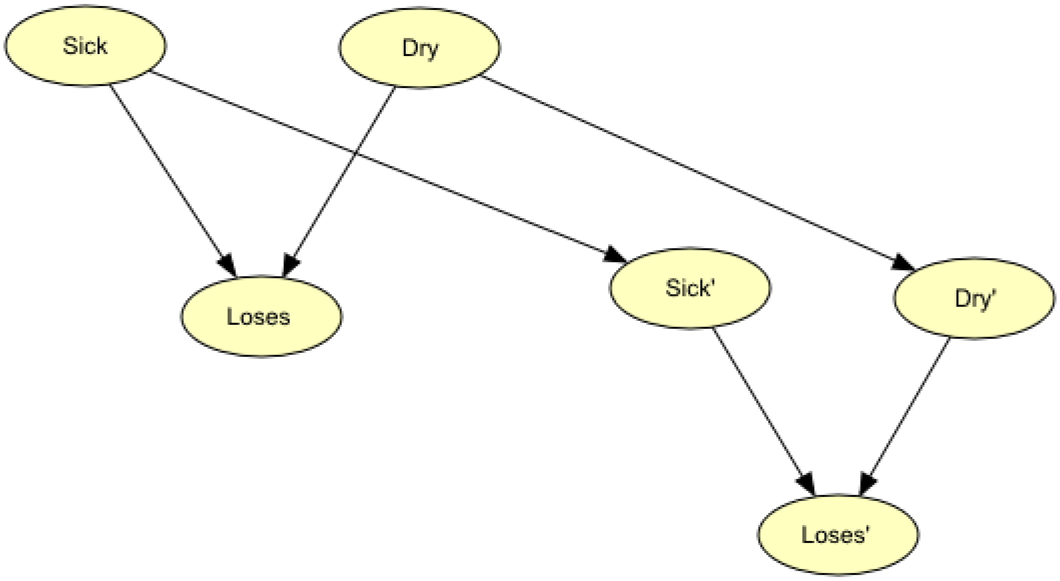


Figure 2: Addition of similar nodes representing the harvest time expectations of the state of the tree.

The new nodes can be in the same states as the old nodes: Sick’ can be either “sick” or “not” - Dry’ can be either “dry” or “not” - and Loses’ can be either “yes” or “no”. In the new model, we expect a causal dependence from both the old Sick node to the new Sick’ node and the old Dry node to the new Dry’ node. This is because if, for example, we expect the tree to be sick now, then this is also very likely to be the case in the future. Of course the strength of the dependence depends on how far out in the future we look. Perhaps one could also have a dependence from Loses to Loses’ but we have not done so in this model.

Apple Jack has the opportunity to do something about his problem. He can try to heal the tree with a treatment to get rid of the possible sickness. If he expects that the losing of leaves is caused by drought, he might save his money and just wait for rain. The action of giving the tree a treatment is now added as a *decision node* to the BN which will then no longer be a BN. Instead it will be the influence diagram shown in Figure 3. Action nodes are represented by rectangles.

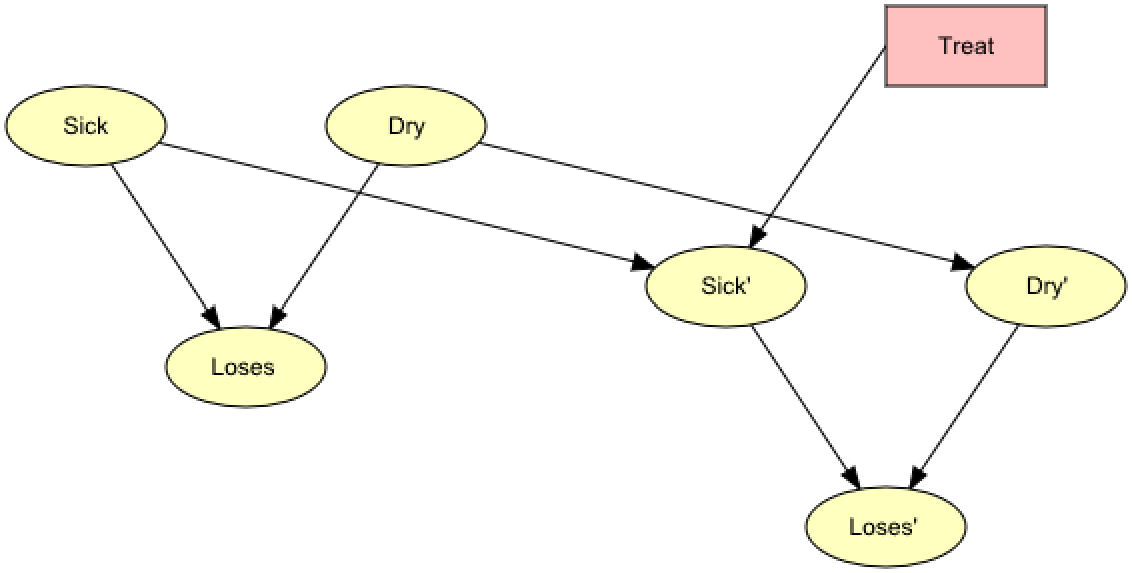


Figure 3: Addition of a decision node for treatment.

The Treat decision node has the states “treat” and “not”. As it appears from Figure 3, we have added a link from Treat to Sick’. This is because we expect the treatment to have an impact on the future health of the tree. Before the influence diagram is completed, we need to specify the utility function enabling us to compute the expected utility of a decision. This is done by adding utility nodes to the diagram, each contributing with one part of the total utility. The utility nodes are added in Figure 4. Utility nodes are represented by diamonds.

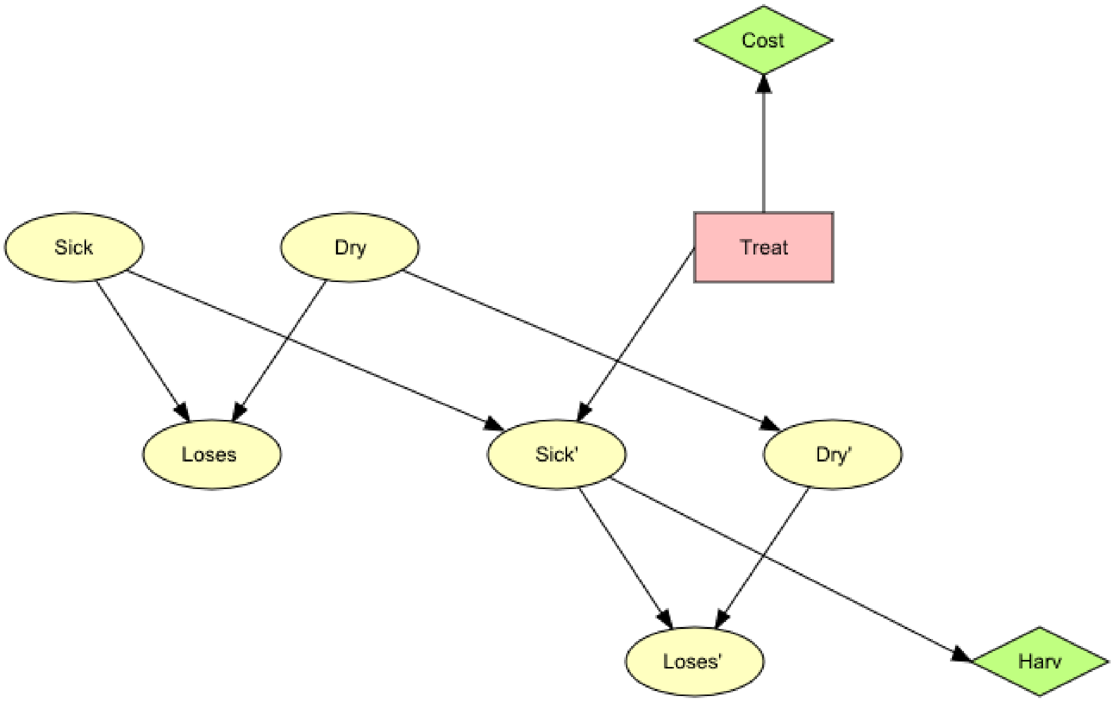
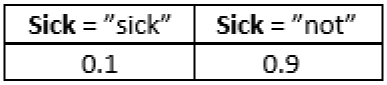


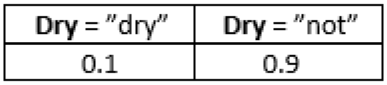
Figure 4: The complete qualitative representation of the (limited memory) influence diagram used for decision making in Apple Jack's orchard.

The utility node Cost gathers information about the cost of the treatment while Harv represents the utility at the time of the harvest. It depends on the state of Sick' indicating that the production of apples depends on the health of the tree. Figure 4 shows the complete qualitative representation of the influence diagram. To get the quantitative representation as well, we need to construct a conditional probability table (CPT) for each chance node and a utility table for each utility node. A decision node does not have any table. The following tables show one way of how the CPTs of the chance nodes could be specified.



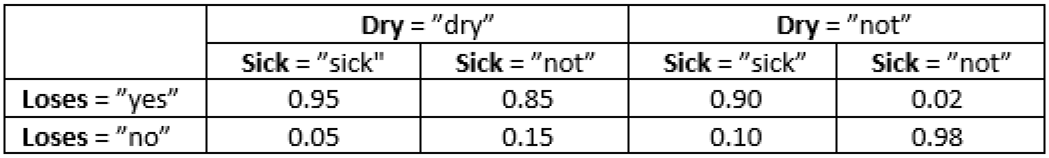
Sick = "sick"	**Sick** = "not"
0.1	0.9

Table 1: P(Sick).



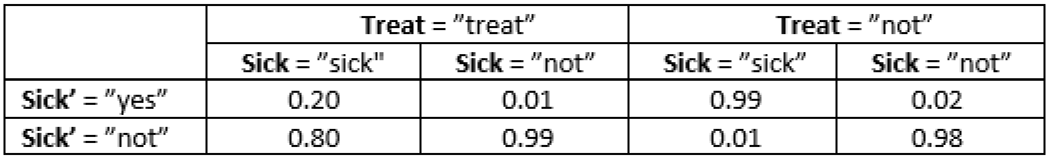
Dry = "dry"	**Dry** = "not"
0.1	0.9

Table 2: P(Dry).



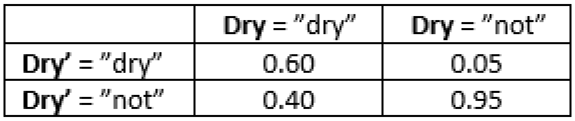
	Dry = ”dry”		**Dry** = ”not”	
	Sick = ”sick"	**Sick** = ”not”	**Sick** = ”sick”	**Sick** = ”not”
Loses = ”yes”	0.95	0.85	0.90	0.02
Loses = ”no”	0.05	0.15	0.10	0.98

Table 3: P(Loses | Sick, Dry).



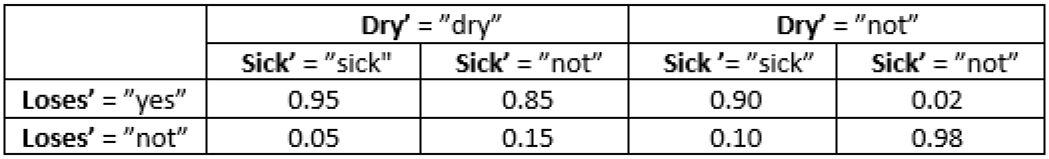
	Treat = ”treat”		**Treat** = ”not”	
	Sick = ”sick"	**Sick** = ”not”	**Sick** = ”sick”	**Sick** = ”not”
Sick’ = ”yes”	0.20	0.01	0.99	0.02
Sick’ = ”not”	0.80	0.99	0.01	0.98

Table 4: P(Sick’ | Sick, Treat).



	Dry = ”dry”	**Dry** = ”not”
Dry’ = ”dry”	0.60	0.05
Dry’ = ”not”	0.40	0.95

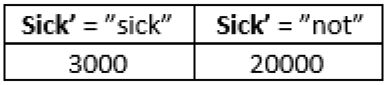
Table 5: P(Dry’ | Dry).



	Dry’ = ”dry”		**Dry’** = ”not”	
	Sick’ = ”sick"	**Sick’** = ”not”	**Sick** ’= ”sick”	**Sick’** = ”not”
Loses’ = ”yes”	0.95	0.85	0.90	0.02
Loses’ = ”not”	0.05	0.15	0.10	0.98

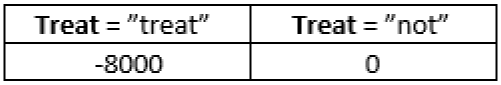
Table 6: P(Loses’ | Sick’, Dry).

The following tables show how the tables of the utility nodes could be specified ín USD.



Sick’ = ”sick”	**Sick’** = ”not”
3000	20000

Table 7: U(Harv).



Treat = ”treat”	**Treat** = ”not”
-8000	0

Table 8: U(Cost).

The utility tables are simply cost functions. Table 7 can be interpreted as if we have healthy tree (Sick’ is in state “not”), then Apple Jack will get a $20000 income, while if the tree is sick (Sick’ is in state “yes”) Apple Jack’s income will be only $3000. Table 8 shows that to treat the tree, Apple Jack has to spend $8000. The purpose of our influence diagram is to be able to compute the action of the Treat node giving the highest expected utility. This is a very tricky job if you are to do it without the help from a computer and we shall not do it here. In stead, we suggest that you now try to go through the *two tutorials* (page 15) describing how to implement this influence diagram in the HUGIN Graphical User Interface and let it do the computations. Also, you might be interested in learning more about the semantics of *(limited memory) influence diagrams* (page 345)

Definition of (Limited Memory) Influence Diagrams

An (*limited memory) influence diagram* is a Bayesian network augmented with decision and utility nodes (the random variables of a limited memory influence diagram are often called chance variables). The limited memory influence diagram relaxes two fundamental assumptions of the traditional influence diagram: the non-forgetting assumption and the total order on decisions.

We are interested in making the best possible decisions for our application. We therefore associate *utilities* with the state configurations of the network. These utilities are represented by *utility nodes*. Each utility node has a utility function that to each configuration of states of its parents associates a utility. (Utility nodes do not have children). By making decisions, we influence the probabilities of the configurations of the network. We can therefore compute the *expected utility* of each decision alternative (the global utility function is the sum of all the local utility functions). We shall choose the alternative with the highest expected utility; this is known as the *maximum expected utility principle*.

Relaxing the non-forgetting assumption and the total order on decisions implies a significant change in the semantics of a LIMID compared to a traditional influence diagram. In a LIMID it is necessary to specify for each decision the information available to the decision maker at that decision. There are no implicit informational links in a LIMID.A link into a decision node specify that the value of the parent node is known at the decision.

The computational method underlying the implementation of limited memory influence diagrams in the HUGIN Decision Engine is described by *Lauritzen & Nilsson (2001)* (page 541). The algorithm used is known as *Single Policy Updating.* (page 258)

The Oil Wildcatter Example

As another example, consider the following decision scenario:

> An oil wildcatter must decide whether or not to drill. He is uncertain whether the hole is *dry*, *wet*, or *soaking*. At a cost of $10,000, the oil wildcatter could take seismic soundings which will help determine the underlying geological structure at the site. The soundings will disclose whether the terrain below has *closed* structure (good), *open* structure (so-so), or *no* structure (bad).

This decision scenario can be represented by the influence diagram of Figure 5, where T represents the decision on whether or not to test; D represents the decision on whether to drill or not to drill; S represents the outcome of the seismic soundings test (if the oil wildcatter decides to test); H represents the state of the hole; C represents the cost associated with the seismic soundings test; and P represents the expected payoff associated with drilling.

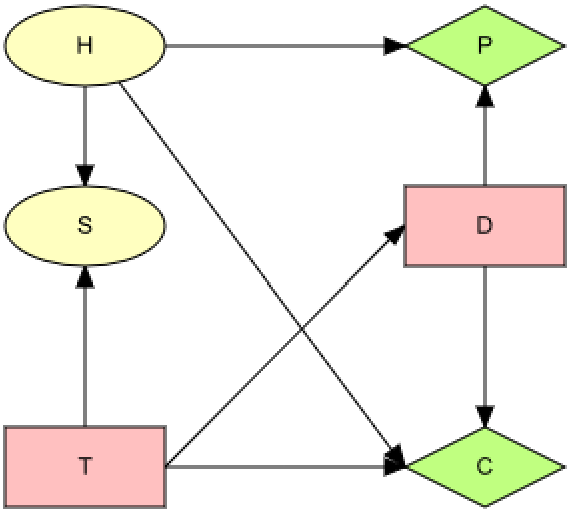
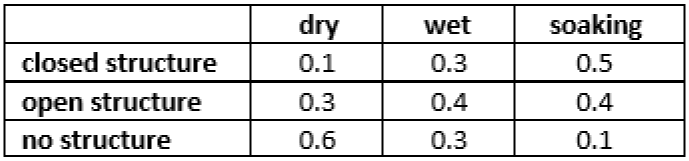


Figure 5: An influence diagram for the oil wildcatters decision problem.

The cost of drilling is $70,000. If the oil wildcatter decides to drill, the expected payoff (i.e., the value of the oil found minus the cost of drilling) is $-70,000 if the hole is *dry*, $50,000 if the hole is *wet*, and $200,000 if the hole is *soaking*; if the oil wildcatter decides not to drill, the payoff is (of course) $0.

The experts have estimated the following probability distribution for the state of the hole: P(*dry*)=0.5, P(*wet*)=0.3, and P(*soaking*)=0.2. Moreover, the seismic soundings test is not perfect; the conditional probabilities for the outcomes of the test given the state of the hole are:



	dry	wet	soaking
closed structure	0.1	0.3	0.5
open structure	0.3	0.4	0.4
no structure	0.6	0.3	0.1

On the basis of this (limited memory) influence diagram, the HUGIN Decision Engine computes the utility associated with testing to be $22,500 and the utility associated with not testing to be $20,000. The optimal strategy is to perform the seismic soundings test, and then decide whether to drill or not to drill based on the outcome of the test.

Notice that the informational link from node *T* to node *D* is of outmost importanct. The LIMID relaxes the non-forgetting assumption of the traditional influence diagram. This implies that for each decision node it is necessary (and very important) to specify exactly the information available to the decision maker. No informational links can be assumed implicitly present in the network. [This example is due to *Raiffa (1968)* (page 541). A more thorough description of this example is found in the *Examples* (page 517) section.]

The Apple Tree Example Continued

The LIMID in Figure 4 representing the decision problem of Apple Jack assumes a single decision with no observations made prior to the decision. In a more realistic setting Apple Jack is likely to monitor the tree over a period of time such as each day. Assume he prior to making a decision on treating the tree observes whether or not the tree is loosing its leaves. Each day he observes if the tree is loosing its leaves and makes a decision on the treatment irrespectively of what he did the previous day. We may use a limited memory influence diagram to model this situation.

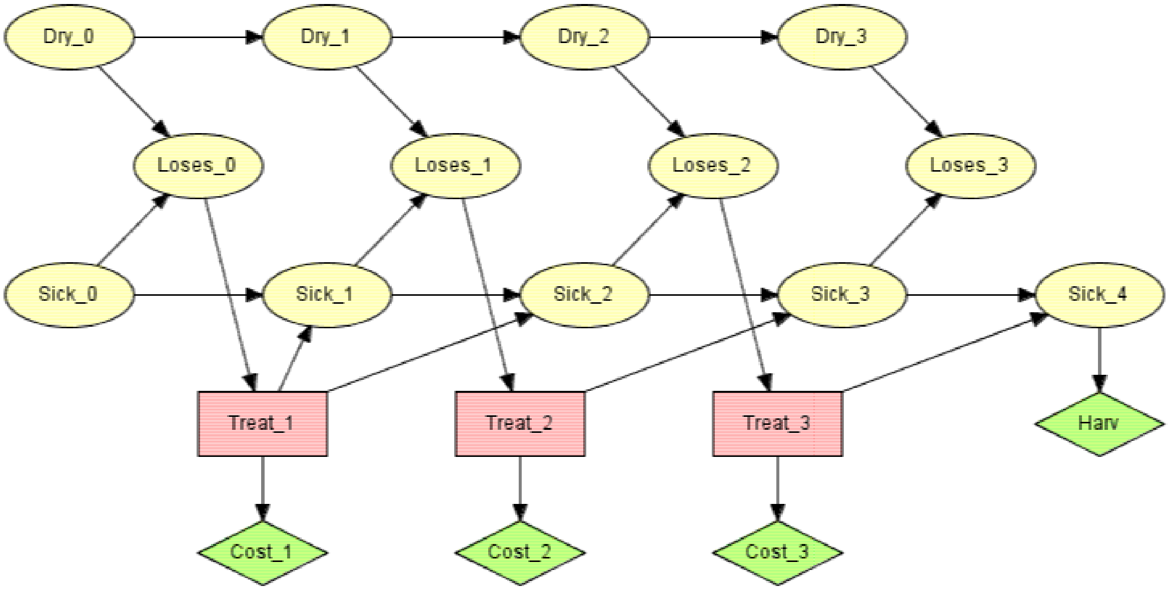


Figure 6: A limited memory influence diagram for Apple Jack with multiple decisions.

Figure 6 shows a limited memory influence diagram for the situation described above where Apple Jack monitors the tree for three days before harvest. The informational links of the diagram specifies that Apple Jack each day observes whether the tree is (still) loosing its leaves, but otherwise neither recalls the observations nor the decisions made at previous time steps.

The solution to the LIMID is a strategy consisting of one policy for each decision. The policy is a function from the known variables to the states of the decision. It is not a function of all past observations as the decision maker is assumed only to know the most recent observation on loses leaves. This is different from the traditional influence diagram where the policy would be a function from all past observations and decisions as the decision maker is assumed to be non-forgetting.

In the example there is a total order on the decisions. This need not be the case in general.

To learn how to build a LIMID using the HUGIN Graphical User Interface, please consult the *How to Build LIMIDs tutorial* (page 45). Also, you might be interested in learning more about the *semantics of (limited memory) influence diagrams* (page 345) and the associated constraints imposed on their usage.

3.2.5 How to Build a (Limited Memory) Influence Diagram

This tutorial shows you how to implement a small influence diagram in the HUGIN Graphical User Interface. It requires that you have already constructed the Bayesian network from the *How to Build a Bayesian Network Tutorial* (page 26). The influence diagram you are about to implement is the one modeled in the *Influence Diagrams Tutorial* (page 37). It helps plantation owner Apple Jack to decide whether or not to give his apple tree, which is losing its leaves, some treatment. The qualitative representation of the influence diagram is shown in Figure 1.

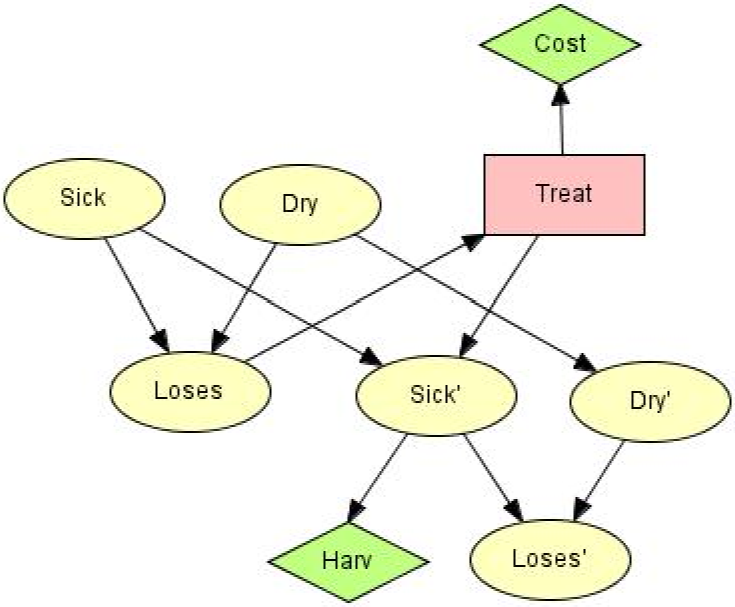


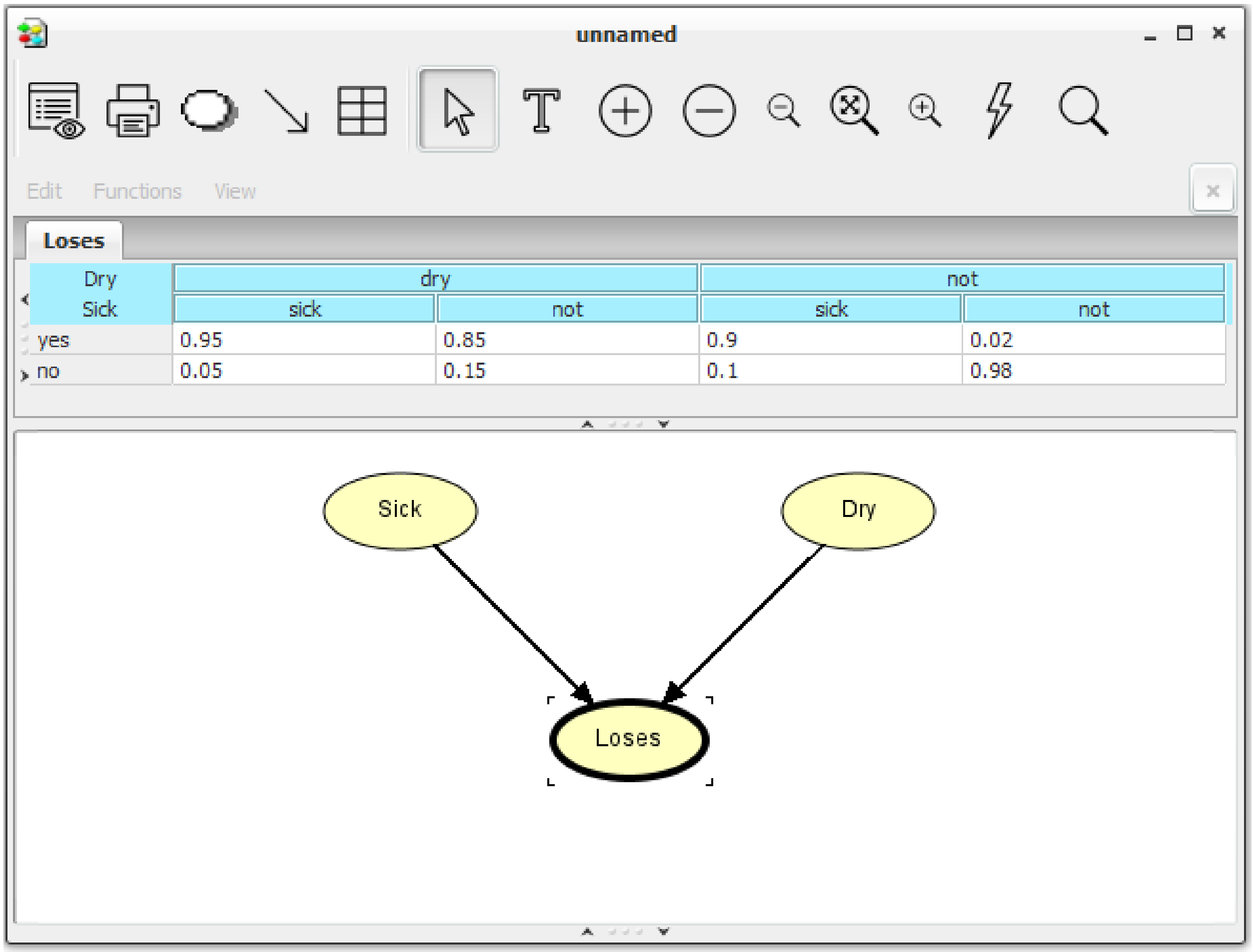
Figure 1: The qualitative representation of the influence diagram used for decision making in Apple Jacks plantation.

Open the Network for Editing

First, you must open the network constructed in the *How to Build BNs* (page 26) tutorial if it is not already open. Here is how to do it:

- Select “Open” from the “File” menu.
- Enter the name of the network file (“apple.net”). You can do this by selecting it from the list of network files (which have the “net” extension).

In Figure 2, the network has been opened and the HUGIN Graphical User Interface is currently working in Edit Mode. We need to be in Edit Mode to edit the network, so if your network window shows the network in Run Mode, press the Edit Mode tool button. If you opened it in Edit Mode, you do not need to do anything.



Dry	dry		not	
Sick	sick	not	sick	not
yes	0.95	0.85	0.9	0.02
no	0.05	0.15	0.1	0.98

Figure 2: The Network Window in Edit Mode with the network from the How to Build BNs tutorial.

Copying Nodes

In the influence diagram in Figure 1, there are three nodes very similar to those that we already have. In this case, the HUGIN Graphical User Interface` allows you to copy a group of nodes and paste them in another area of the Network Pane. Here is how to do it:

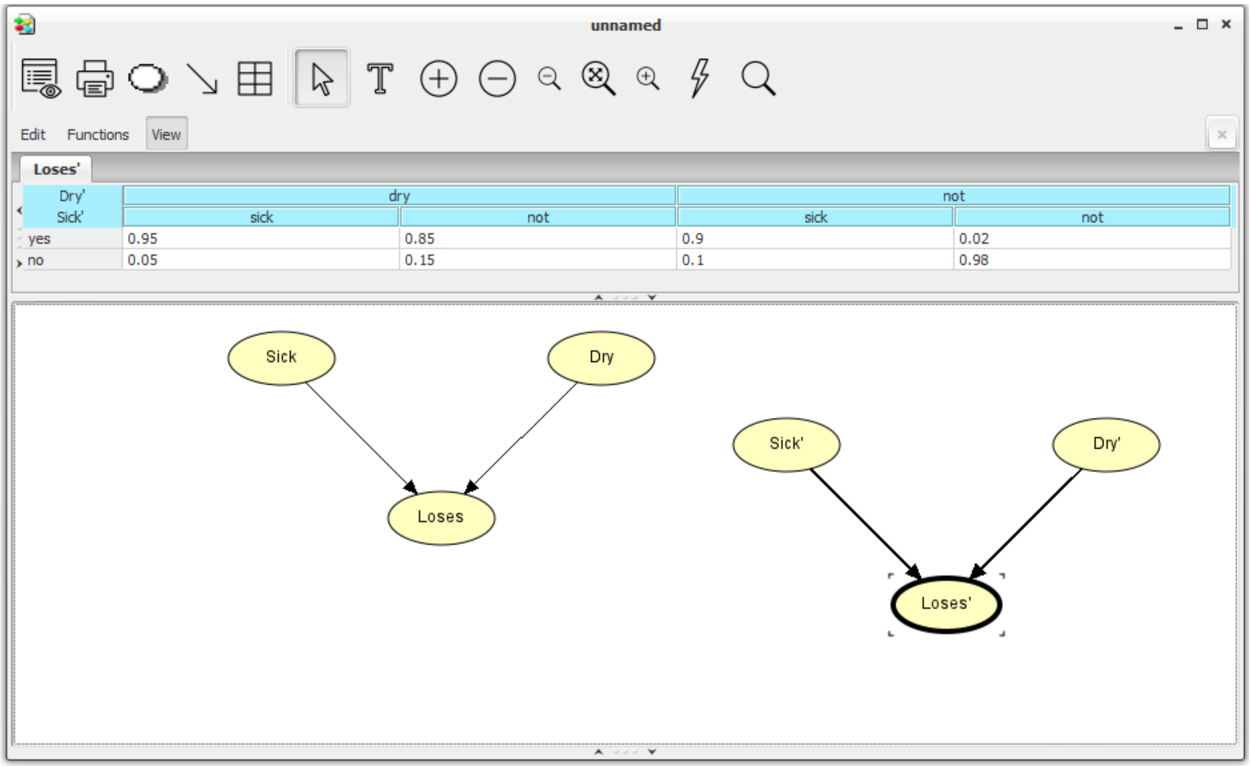
- Create a rectangle selection with the mouse cursor around all three nodes (drag a rectangle by holding down the left mouse button).
- Select “Copy” from the “Edit” menu in the Main Window Menu Bar.
- Select “Paste” from the “Edit” menu in the Main Window Menu Bar.
- Move the new group of nodes to a spot where there is room for them.

The HUGIN Graphical User Interface generates new names and labels for the new nodes. You can keep the names and change the labels to Sick’, Dry’, and Loses’ (you cannot use “Sick’” as the name because it contains the prime character which is illegal in names):

- Select the node with the mouse cursor.
- Enter “Node Properties” by pressing the node properties tool (the 2nd left-most tool button in the tool bar of the network window).

- Change the "Label" field.
- Press the "OK" button.

Perform the steps above for all three new nodes. Your network should then look as the one in Figure 3.



Dry'	dry		not	
Sick'	sick	not	sick	not
yes	0.95	0.85	0.9	0.02
no	0.05	0.15	0.1	0.98

Figure 3: The network extended with Sick', Dry', and Loses'.

The next step is to add causal links from Sick to Sick' and from Dry to Dry': * Press the Link Tool. * Drag a link from Sick to Sick' with the left mouse button (while holding down the SHIFT key). * Repeat for Dry to Dry'.

Holding down the SHIFT key enables you to create more causal links sequentially without having to reactivate the Link Tool.

Adding a Utility Node

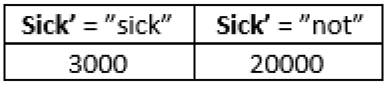
So far, the network we have constructed is still a Bayesian network. Now, we shall make the first change that makes it an influence diagram. This change is the addition of a utility node. The utility node we shall add is the Harv node (see Figure 1) representing the utility gained from the harvest. Here is how to add it:

- Press the *Utility Tool* (to the right of the Link Tool).
- Click somewhere in the Network Pane (a good place would be in the lower right corner besides the Loses' node).
- Change the name and label of the new utility node to "Harv".

The harvest depends on the state of Sick' and thus there is an link from Sick' to Harv. Add this link:

- Press the *Link Tool*.
- Drag a link from Sick' to Harv.

The utility of the harvest was specified to that found in Table 1.



Sick' = "sick"	Sick' = "not"
3000	20000

Table 1: U(Harv).

You enter the values of Table 1 into the utility table of Harv as follows:

- Select the Harv node by clicking it with the left mouse button.
- Enter the values from Table 1 in the utility table in the Tables Pane.

A Desicion Node and One More Utility Node

Now, you are about to add the decision node Treat (see Figure 1). This is done similar to the way you add chance nodes and utility nodes:

- Press the *Decision Tool* (to the right of the Utility Tool).
- Click somewhere in the Network Pane (a good place would be to the right of the Dry node)
- Change the name and label of the new decision node to “Treat”.

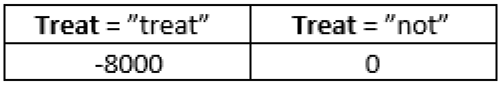
You add an action to a decision node in the same way as you add a state to a chance node:

- Select the Treat node with the left mouse button.
- Press the add state tool.
- Change the action names to “treat” and “not”.

The Treat decision node has an impact on the Sick’ node so:

- Add a link from Treat to Sick’.

The new decision node represents the decision to give the tree some treatment or not. If the plantation owner (Apple Jack) chooses to give treatment this will cost him something which shall be modeled by the Cost utility node. The Cost node has the utility table shown in Table 2.



Treat = "treat"	Treat = "not"
-8000	0

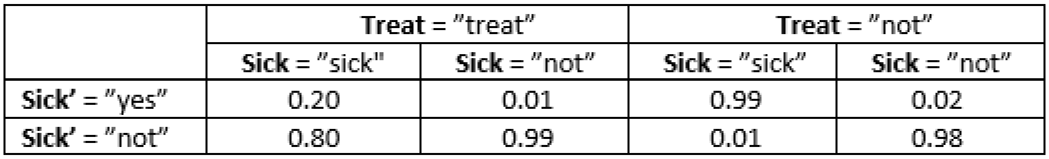
Table 2: U(Cost).

Now, add the Cost utility node to the influence diagram:

- Add a new utility node (a good place would be to the right of the Treat node).
- Change the name and label of this node to “Cost”
- Add a link from Treat to Cost.
- Fill in Table 2 in the utility table of Cost.

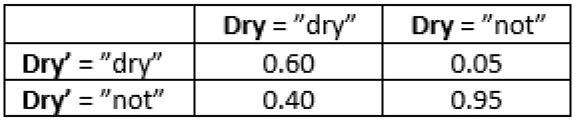
Filling in CPTs

When we copied the nodes Sick’ and Dry’, they inherited the CPTs of Sick and Dry. However, as both these nodes have become children of other nodes, their CPTs are no longer correct. Their new CPTs were specified to those found in Table 3 and Table 4. * Fill in Table 3 as the CPT of Sick’. * Fill in Table 4 as the CPT of Dry’.



	Treat = ”treat”		**Treat** = ”not”	
	Sick = ”sick"	**Sick** = ”not”	**Sick** = ”sick”	**Sick** = ”not”
Sick’ = ”yes”	0.20	0.01	0.99	0.02
Sick’ = ”not”	0.80	0.99	0.01	0.98

Table 3: P(Sick’ | Sick, Treat).



	Dry = ”dry”	**Dry** = ”not”
Dry’ = ”dry”	0.60	0.05
Dry’ = ”not”	0.40	0.95

Table 4: P(Dry’ | Dry).

Now, your (limited memory) influence diagram (LIMID) is finished and it should look like the one in Figure 4. At this point it would be a good idea to save your LIMID.

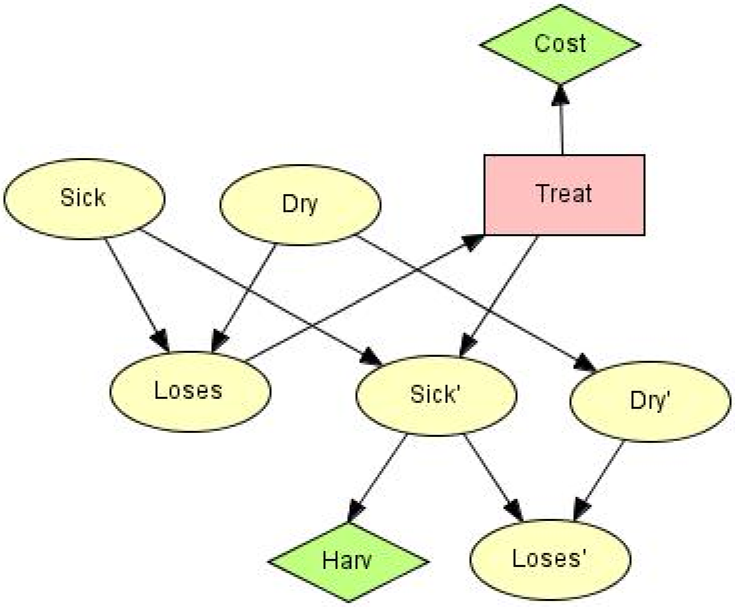


Figure 4: The complete influence diagram.

Compiling the Limited Memory Influence Diagram

You can now try out the LIMID. First, compile the LIMID:

- Press the compile tool (the right most tool button in the network window Tool Bar).

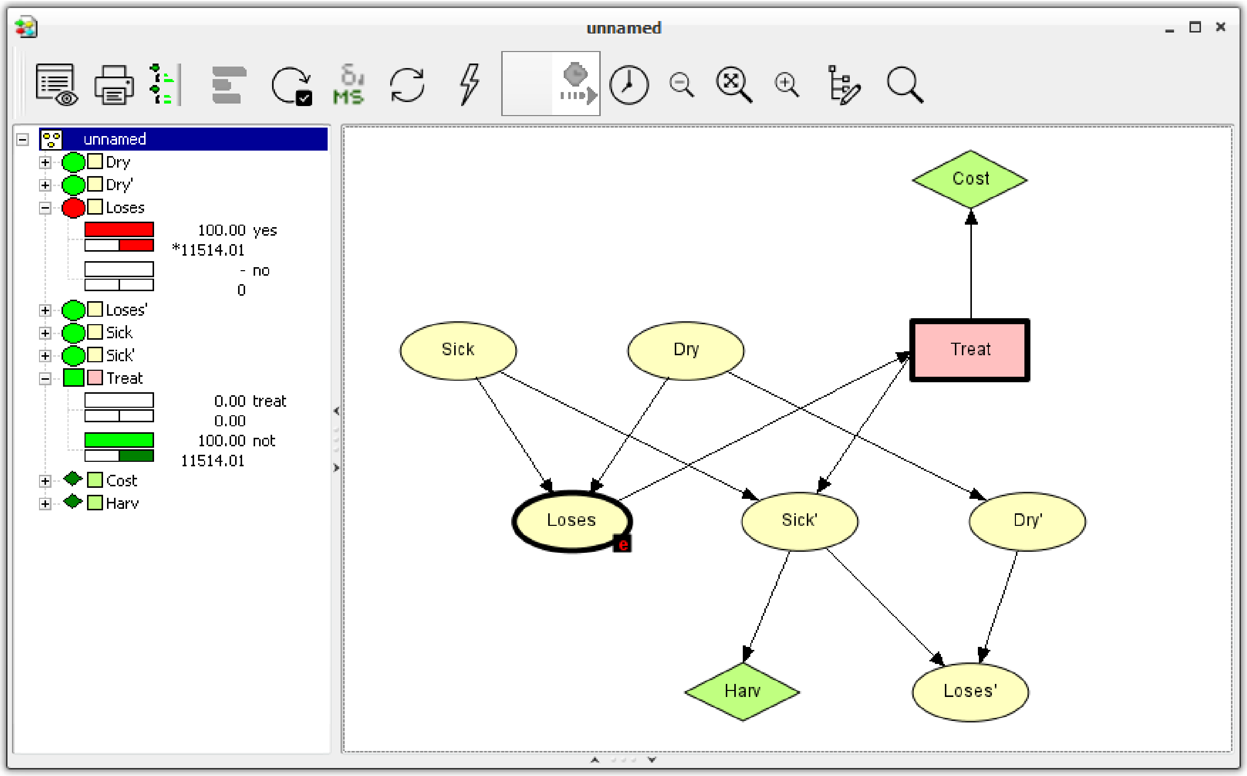
The compilation of an influence diagram may produce some of the same errors as described in the *How to Build BNs* (page 26) tutorial. If the LIMID does not compile, you have probably made some minor error. Once the influence diagram has been compiled, probabilities and expected utilities are computed under the initial policy. To solve the influence diagram it is necessary to invoke *Single Policy Updating* (page 258).

What Should Apple Jack Do?

When the LIMID has been compiled, you should do a *Single Policy Updating* (page 258) . Now, imagine that the only thing Jack knows about his tree is that it is losing leaves. Then, what will be the best thing for him to do? To find out this, follow these steps:

- Expand the Loses chance node and the Treat decision node in the node list pane on the left (simply select them).
- Enter the evidence that Loses is "yes" (by double clicking the "yes" state).
- Propagate the influence diagram (press the Sum Propagation Tool, unless auto-propagate is set).
- Read the expected utility of "treat" and "not" in the Treat decision node.

You should be reading something looking like that in Figure 5.



```
unnamed
Dry
Dry'
Loses
        100.00 yes
     *11514.01
             - no
             0
Loses'
Sick
Sick'
Treat
          0.00 treat
          0.00
        100.00 not
      11514.01
Cost
Harv
```

Figure 5: The influence diagram propagated with the evidence that Loses="yes".

You read 11514 as the expected utility of doing nothing. This suggests that it will be best for Apple Jack not to treat the tree.

This finishes the tutorial. You should now be able use the HUGIN Graphical User Interface to construct your own (limited-memory) influence diagrams. However, if you want to create large and complex models, you should study the area more than just reading this tutorial.

Please read the document *semantics of LIMID* (page 345) to learn more about LIMIDs.

3.2.6 Introduction to Object-Oriented Networks

An *Object-Oriented Network* is a network (i.e., Bayesian network or LIMID model) that, in addition to the usual nodes, contains *instance nodes*. An instance node is a node representing an instance of another network. In other words, an instance node represents a subnet. Therefore, following standard object-oriented terminology, an object-oriented network is often referred to as a *class*. Of course, the network of which instances exist in other networks can itself contain instance nodes, whereby an object-oriented network can be viewed as a hierarchical description (or model) of a problem domain. There are three main advantages of constructing a network using instance nodes, as described in the following.

The model construction activity most often involves repeated changes of level of abstraction. That is, it is performed in a top-down fashion, a bottom-up fashion, or a mix of the two. Such repeated changes of focus are due partly to the fact that humans naturally think about systems in terms of hierarchies of abstractions and partly due to lack of ability to mentally capture all details of a complex system simultaneously. The use of instance nodes provides support for working with different levels of abstraction in constructing network models.

As systems often are composed of collections of identical or almost identical components, models of systems often contain repetitive patterns (i.e., commonly occurring solutions or problem types). In Bayesian networks and LIMIDs, such patterns are network fragments. The notion of instance nodes makes it very easy to construct multiple identical instances of a network fragment.

Describing a network in a hierarchical fashion often makes the network much less cluttered, and thus provides a much better means of communicating ideas among knowledge engineers and users.

The HUGIN Graphical User Interface 6.* supports construction of object-oriented networks in the basic interpretation of the concept mentioned above. A fully object-oriented paradigm for constructing Bayesian networks and LIMIDs should also support the notions of subclasses and inheritance, as known from object-oriented programming languages. Thus, the basic mechanisms provided by the HUGIN Graphical User Interface 6.* supports the construction of what might be called *hierarchical networks*.

Interface nodes

An instance node connects to other nodes via some of the (basic) nodes in the copy of the network (the *master*) of which it is an instance. (Note that an instance node should be thought of as a copy of the network of which it is an instance.) These nodes are known as *interface nodes*. As we wish to support information hiding, the interface nodes only comprise a subset of the nodes in the master network. Interface nodes are subdivided into a set of *input nodes* and output nodes.

Input nodes of an instance of a master network are not real nodes but only to be considered as placeholders for (basic) nodes of the network(s) containing instances of the master network. These basic nodes are said to be *bound* to the input nodes (and vice versa). Note that if an input node of an instance hasn't been bound, a default potential (probability table) will be associated with it. The probabilities in this table is specified in the master network. Output nodes of an instance of a master network are real nodes that can be specified as parents of nodes in the network containing the instance node or can be bound to an input node of another instance node of the network. The following example demonstrates what all this means.

The Disease Example Revisited

In the *Bayesian Networks* (page 19) section, we presented the dynamic Bayesian network example shown in Figure 1. This example illustrates the progression of diseases D1 and D2 over time such that D1 at the next time slice depends on D1 at the current time slice, and similarly for D2. S1 and S2 represent symptoms for the diseases. In Figure 1 we have only indicated the model for two consecutive time slices.

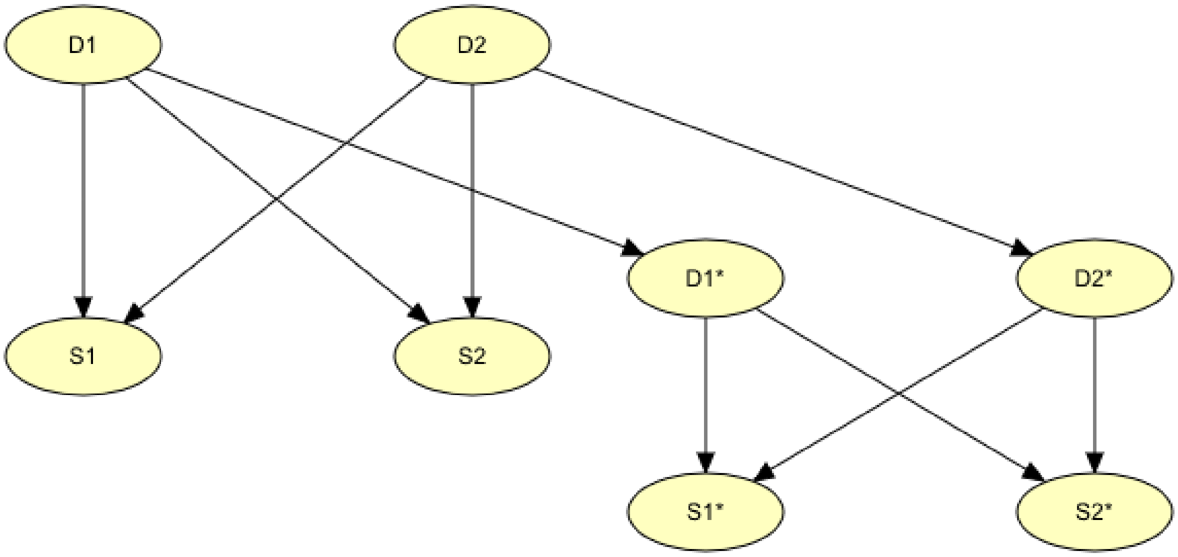


Figure 1: An example of Dynamic Bayesian Network (DBN).

Creating the model in this fashion covering several time slices is a tedious job. Assuming that the structures and the tables of the time slices are identical, and that the transition probabilities P(D1|D1 prev) and P(D2|D2 prev) identical for all time slices (which is often the case in real-world applications), constructing such a time-sliced model can be done very elegantly using instance nodes. First, we create the model for a single time slice (see Figure 2). This model takes as input the disease nodes of the previous time slice, and the disease nodes D1 and D2 act as output nodes, since they should be bound to the input nodes of the next time slice. The input nodes are indicated in Figure 2 with dashed outlines.

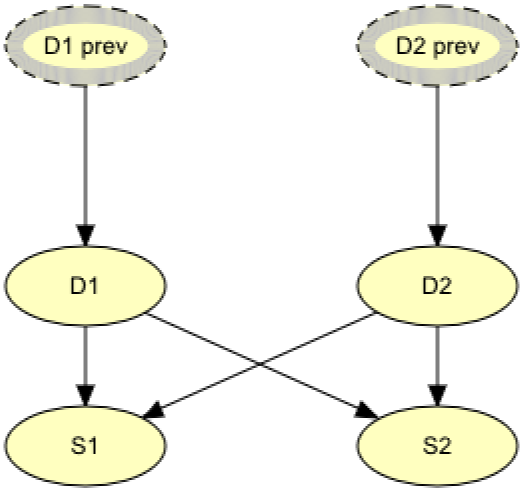


Figure 2: BN representing a single time slice in the disease problem.

The transition probabilities P(D1|D1 prev) and P(D2|D2 prev) describing the temporal behavior as well as the probabilities describing the atemporal behavior are contained in this model.

Second, we construct a model covering, say 10, time slices simply by creating k instances of the network in Figure 1 and bind the outputs of time slice 1 to the inputs of time slice 2, etc. In Figure 3, we have shown the resulting model for k=3.

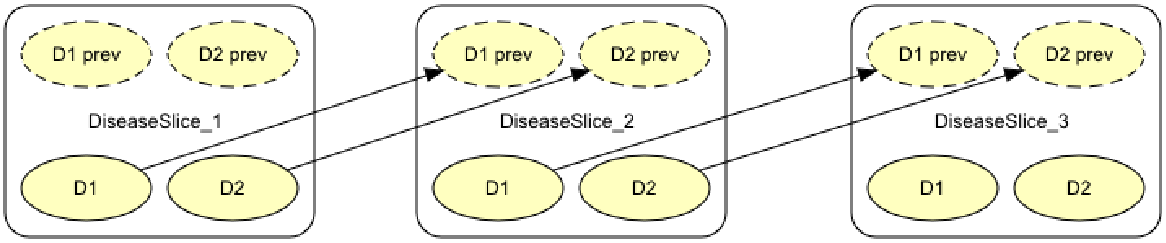


Figure 3: A 3-time slices dynamic Bayesian network (DBN) specified as an object-oriented network.

If there are many instance nodes, the object-oriented network might appear somewhat cluttered. Therefore, once the binding links have been created, we most often prefer to collapse the instance nodes, as shown in Figure 4.

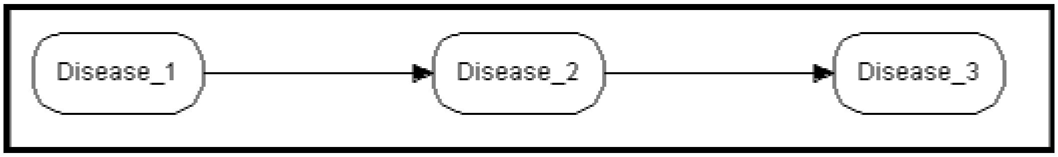


Figure 4: The DBN in Figure 3 with collapsed instance nodes.

Accident Example

The following example was first used in Koller & Pfeffer (1997) Object-Oriented Bayesian Networks, Proceedings of the 13th Conf. on UAI. The network (see Figure 5) models a car accident situation, and contains instance nodes for subnetworks representing characteristics of the driver, the car, and the road.

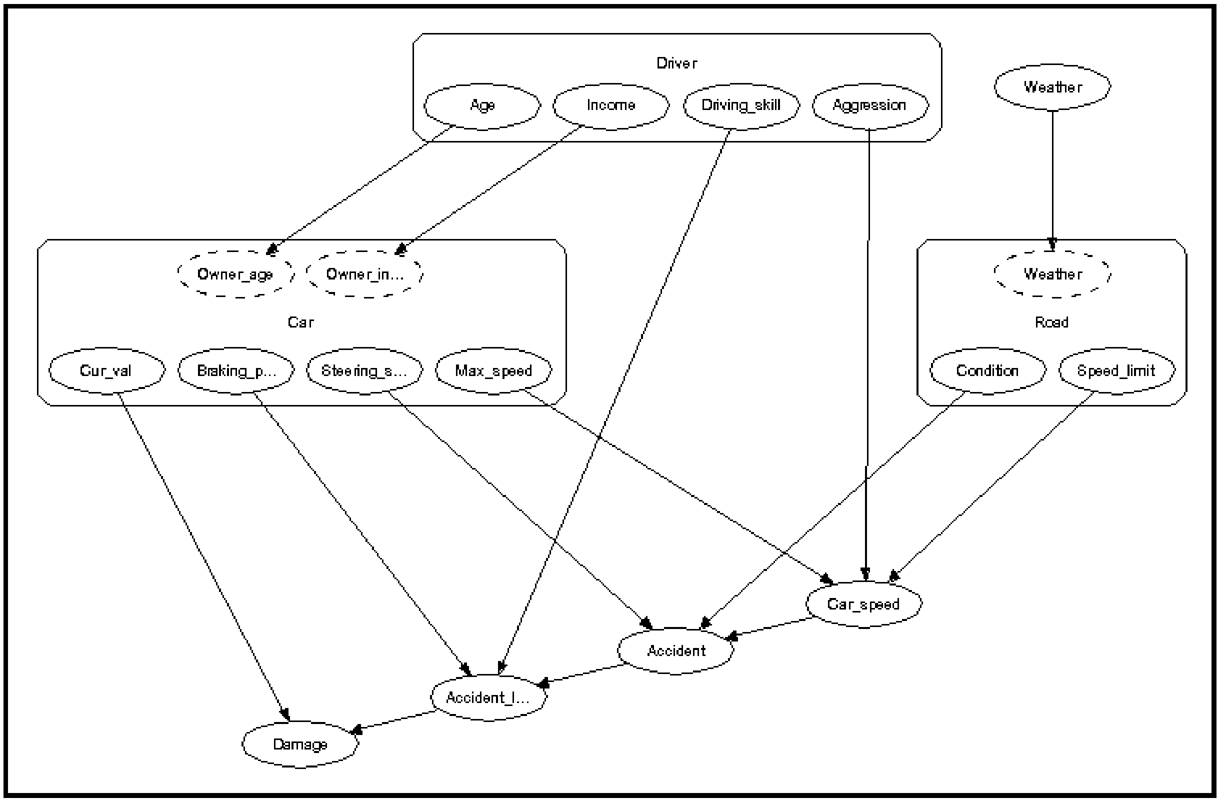


Figure 5: An object-oriented network describing a car accident situation.

Again, collapsing the instance nodes makes the network less cluttered, as illustrated in Figure 6.

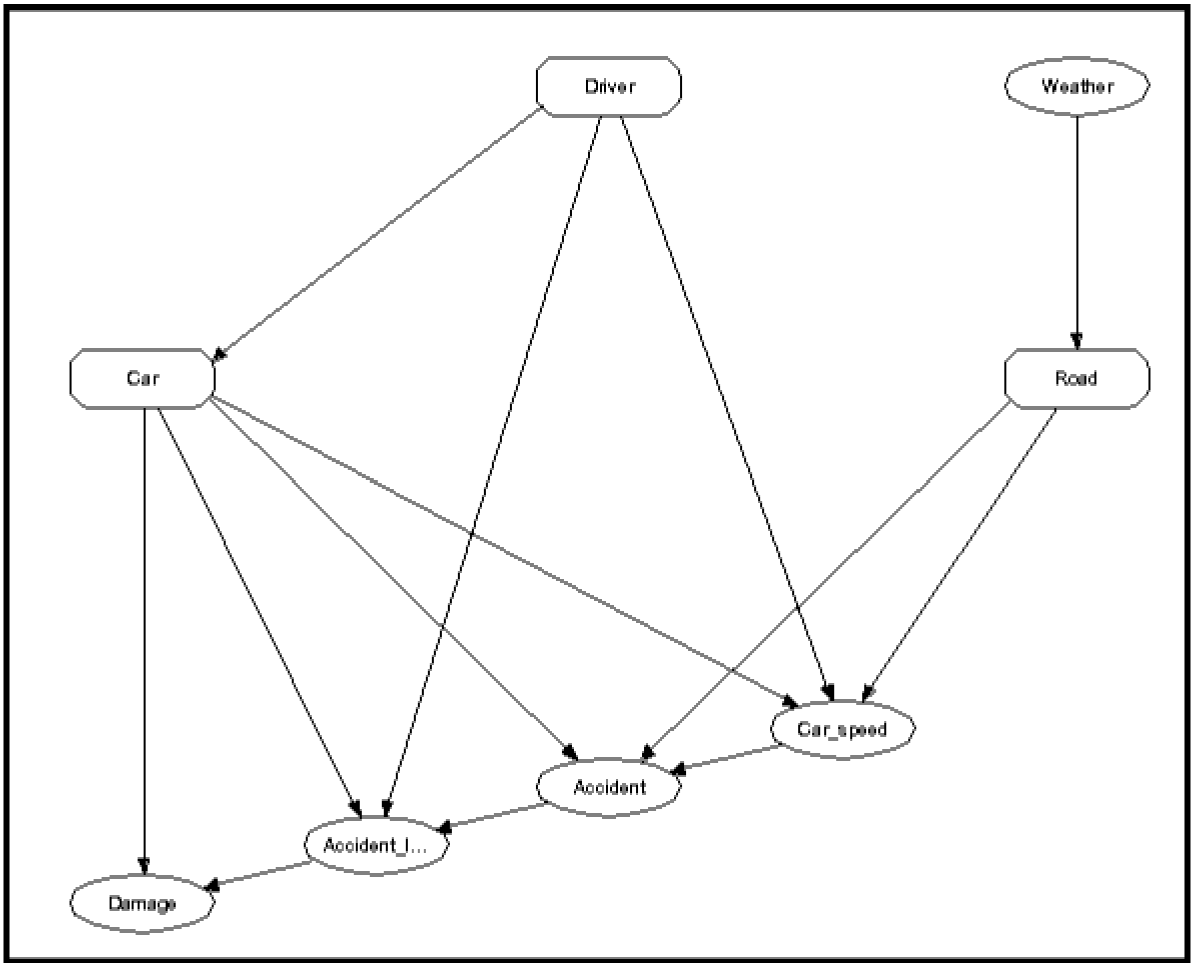


Figure 6: The network in Figure 5 with instance nodes collapsed.

Now, the network of which the car node is an instance contains itself several instance nodes, as shown in Figure 7.

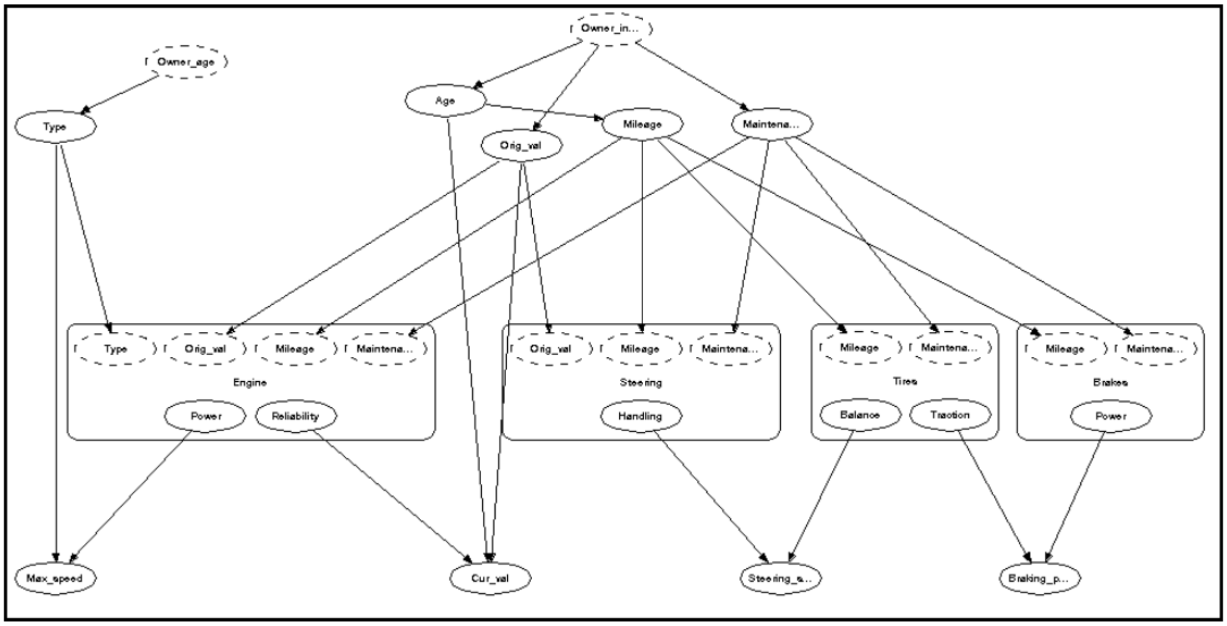


Figure 7: The network of which the car node is an instance.

Figure 7: The network of which the car node is an instance. The benefit of constructing this model using the object-oriented network technology is indeed clear. HUGIN provides you with a tool to construct object-oriented networks. After constructing an object-oriented network, you can do belief revision, belief updating, and much more, just as for ordinary BNs. To learn how to build object-oriented networks using the HUGIN Graphical User Interface, please consult the *How to Build OOBNs* (page 56) tutorial.

3.2.7 How to Build an Object-Oriented Bayesian Network

This tutorial shows how to implement a small object-oriented network in the HUGIN Graphical User Interface. The network we are about to construct is the one modeled in the Diseases example in the *Object Orientation* (page 51) tutorial. The qualitative (or structural) representation of our object-oriented network is shown in Figure 1. In this tutorial, we shall ignore the specification of the CPTs.

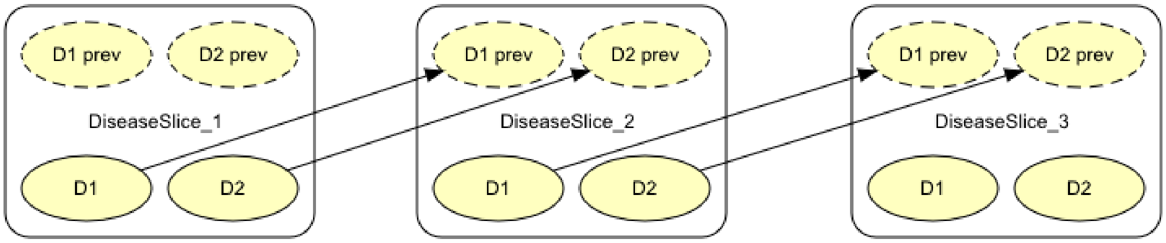


Figure 1: Object-oriented network representing the Diseases problem.

If you want to understand the design of this object-oriented network, you should read about it in the Object Orientation tutorial. Were we to construct this time-sliced network as a Bayesian network, we would get the network shown in Figure 2.

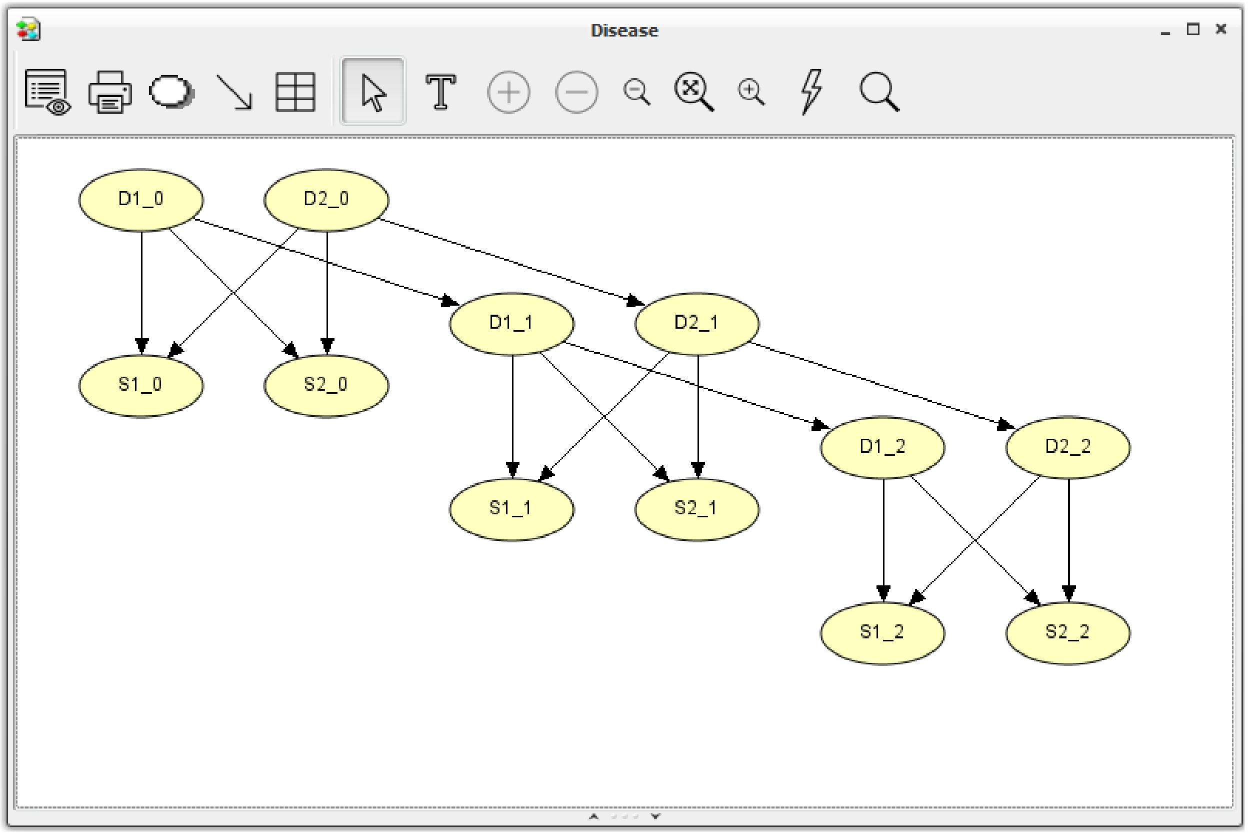


Figure 2: BN representation of the Diseases problem.

Creating the Time-Slice Model

A network (i.e., Bayesian network or LIMID) is a special case of an object-oriented network. What makes a network object oriented is the existence of instance nodes (i.e., nodes that represent instances of other networks). Thus, we start out by creating a new empty network by selecting the “New” menu item in the “File” menu. This gives us a new network window containing an empty network called “unnamed<x>”, where x is some integer. It starts up in *Edit Mode* which allows us to start constructing the object-oriented network immediately (the other main mode is *Run Mode* which allows you to use the network). ,

In Figure 2, we observe that each of the three time slices contains four nodes: D1, D2, S1, and S2, where D1 and D2 represent two different diseases with states “Present” and “Absent”, and S1 and S2 represent symptoms that both may be observed as consequences of each of the diseases. We shall assume that S1 and S2 represent symptoms with two possible outcomes, “Observed” and “Unobserved”. As each of the time slices are identical, both at the qualitative (or structural) level and quantitative level (i.e., the CPTs are identical, including those that describe the temporal aspect, namely P(D1_2|D1_1), P(D2_2|D2_1), etc.), we need only construct a model describing a generic time slice and then connect three instances of this network.

Creating the Nodes

First, we construct the generic time-slice model, containing the four nodes D1, D2, S1, and S2. The nodes all represent discrete chance variables. Therefore, we select the *Discrete Chance Tool* (page 223) and create the four nodes by clicking the left mouse button at four different locations in the *network pane* (page 177) while keeping the Shift key down (to avoid reselecting the tool for each node). We then change the default names of the nodes and their default state names using the *Node Properties* (page 212) pane. Second, we select the Link Tool and create the link from D1 to S1 by dragging the mouse cursor (i.e., pressing the left mouse button and moving the mouse cursor while keeping the button pressed) from a point inside D1 to a point inside S1 and then releasing the mouse button. Again, we keep the Shift key pressed, and create the other three links in the same manner. The result is illustrated in Figure 3.

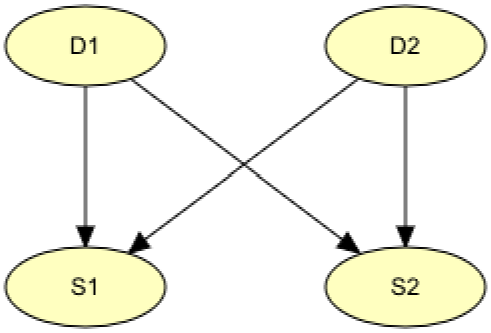


Figure 3: BN for a single time slice of the Diseases problem.

Output Nodes

Now, in order for a network for a single time-slice have parent nodes in the immediately preceding network, we need to be able to refer to nodes outside the network in Figure 3. In a conventional BN, this is not possible. Thus, as D1 and D2 are going to be parents of D1 and D2, respectively, in the next time slice, we must declare D1 and D2 as *output nodes*, making them visible outside the network (or rather through instances of the network).

Input Nodes

Also, in the network in Figure 3, we should be able to specify the temporal aspect, namely the CPTs P(D1|D1 prev) and P(D2|D2 prev), where the nodes “D1 prev” and “D2 prev” are placeholder nodes for D1 and D2, respectively, in the immediately preceding time slice. Such placeholder nodes are referred to as *input nodes*, and shouldn’t be confused with real nodes. A real node, which is type consistent with an input node, can be bound to that input node. That is, an input node becomes identical with the node that is bound to it. However, if an input node hasn’t got a binding associated with it, the network containing the input node can still be used (i.e., compiled in to a junction tree and used for inference). In that case the input node is treated as a real node. That is, each input node has a CPT associated with it just as any ordinary node, but this CPT is used only if no nodes have been bound to the input node in a network containing an instance of the network in which the input node is defined. Input nodes and output nodes are collectively referred to as *interface nodes*.

Creating the Interface Nodes

Now, let's try to put all this into practice. First, we declare D1 and D2 as output nodes. This is done by clicking the "Output" check box in the *Node Properties* (page 212) pane for each of them. To indicate their new status as output nodes, D1 and D2 now are drawn with thick borders of the color selected for interface nodes (as set in the *Network Properties* (page 178) pane).

To create the two input nodes, "D1 prev" and "D2 prev", we first create two ordinary nodes and set their names to D1_prev and D2_prev (and/or their labels to "D1 prev" and "D2 prev"), respectively, in the Node Properties pane. Also, in the Node Properties pane for each of these two new nodes, we click the "Input" check boxes to declare them as input nodes. Similar to D1 and D2, "D1 prev" and "D2 prev" are drawn with thick borders of the interface nodes color. In addition, the appearance of the (regular) borders of "D1 prev" and "D2 prev" changes from solid to dashed, which indicates that they are not real nodes.

Finally, we create links from "D1 prev" and "D2 prev" to D1 and D2, respectively. The result of these operations appear in Figure 4.

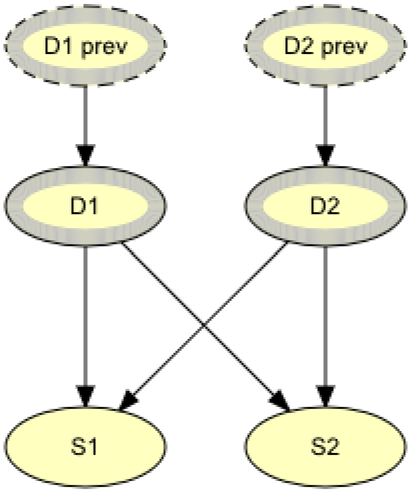


Figure 4: BN for a single time slice of the Diseases problem, including specifications of interface nodes.

Creating the Diseases Mode

To create the final Diseases model spanning three time slices, we first create a new empty network (via the "New" menu item in the "File" menu). Next, we select the *Instance Tool* (page 152) and create three instance nodes by clicking the left mouse button at three different locations in the *network pane* (page 177) while keeping the Shift key down. The result appears in Figure 5 (scaled to 81%).

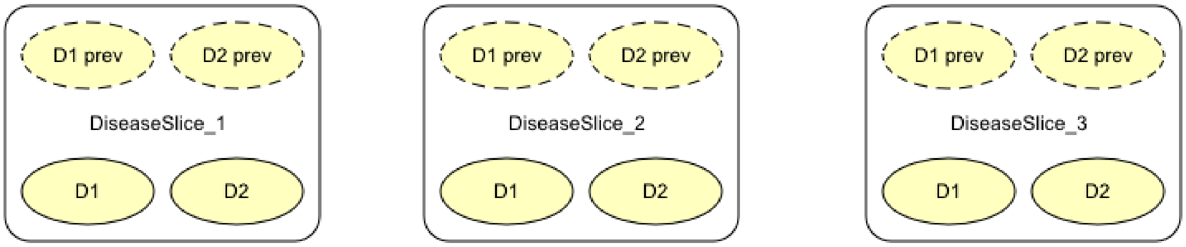


Figure 5: The Network Pane contains the three instance nodes that represent three instances of the time-slice model shown in Figure 4.

Each instance node appear as a rectangle with rounded corners. We note that the nodes declared as interface nodes in the generic time-slice model appear in each of the instance nodes. The input nodes appear in a row at the top of the instance node, and, similarly, the output nodes appear at the bottom of the instance node.

Next, we need to bind the output nodes of instances DiseasesSlice_1 and DiseasesSlice_2 to the input nodes of instances DiseasesSlice_2 and DiseasesSlice_3, respectively. This is done by creating links (via the :ref:` Link Tool<Link_tool>`) from the output nodes to the corresponding input nodes. The resulting model appears in Figure 6.

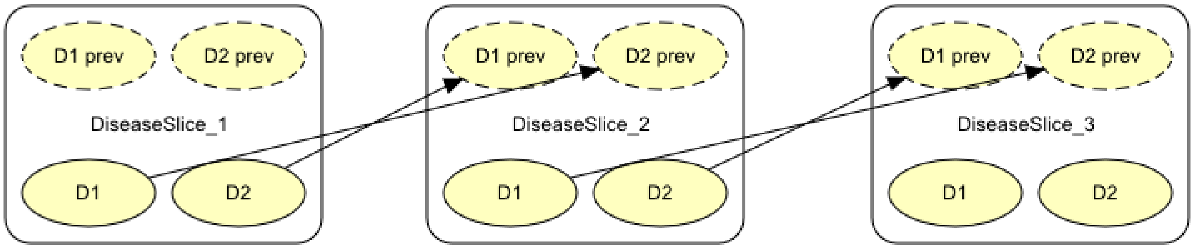


Figure 6: The output nodes of an instance are bound to input nodes of another instance using the Link Tool.

Obviously, the model would look nicer if the order of appearance of the input nodes were reversed. If we select the *Select Tool* (page 153), we can easily alter the order of appearance of the interface nodes. We move an interface node one position to the left (right if the Shift key is down) by placing the mouse cursor on top of the interface node and clicking the left mouse button. After reordering the network appears as in Figure 7.

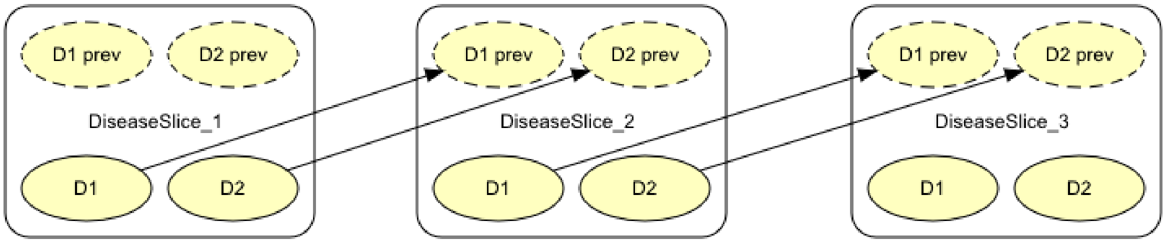


Figure 7: The order of appearance of interface nodes can be altered through simple mouse clicks..

Finally, it is often desireable to be able to collapse one or more of the interface nodes in order to hide away irrelevant details, thereby making the network much less cluttered. Again, if we activate the *Select Tool* (page 153), we can collapse (expand) an expanded (collapsed) instance node, simply by clicking the left mouse button right outside the node. Alternatively, we can choose the “Collapse Instance Nodes” (“Expand Instance Nodes”) menu item in the “View” menu, which collapses (expands) all instance nodes.

The object-oriented network in Figure 7 with the instance nodes collapsed appear in Figure 8.

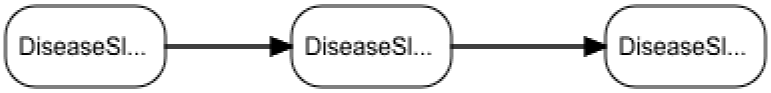


Figure 8: The object-oriented network in Figure 7 with the instance nodes collapsed.

Compiling the Object-Oriented Network

Now, assuming the CPTs of the time-slice model in Figure 4 has been filled in (see the *tutorial on BNs* (page 26) for details), it is the time to compile the network and see how it works:

- Press the Run Mode tool button in the Tool Bar (see Figure 9).

Figure 9: The Run Mode tool button.

- For each configuration of parent states in the CPT of a node the probabilities of the different states of the node must sum to 1. In other words, each column of the table must sum to 1. If there is a column that does not sum to 1, the compiler will normalize the values. This fact can be utilized when filling in the probabilities. Say, for example, that the probability of D1=Present in the first time slice is based on the observation of 13527 patients, 168 of whom were observed to have the disease. Instead of first computing the fractions, you just put 168 in the Present state of D1, and 13359 in the Absent state. Then the compiler will compute the proper values.

The compilation of small networks like the Diseases network is completed in very short time. After the compilation, the Run Mode is entered (we have so far only been working in Edit Mode).

Running the Object-Oriented Network

In Run Mode, the network window is split into two by a vertical bar (see Figure 10). To the left is the Node List Pane and to the right is the Network Pane.

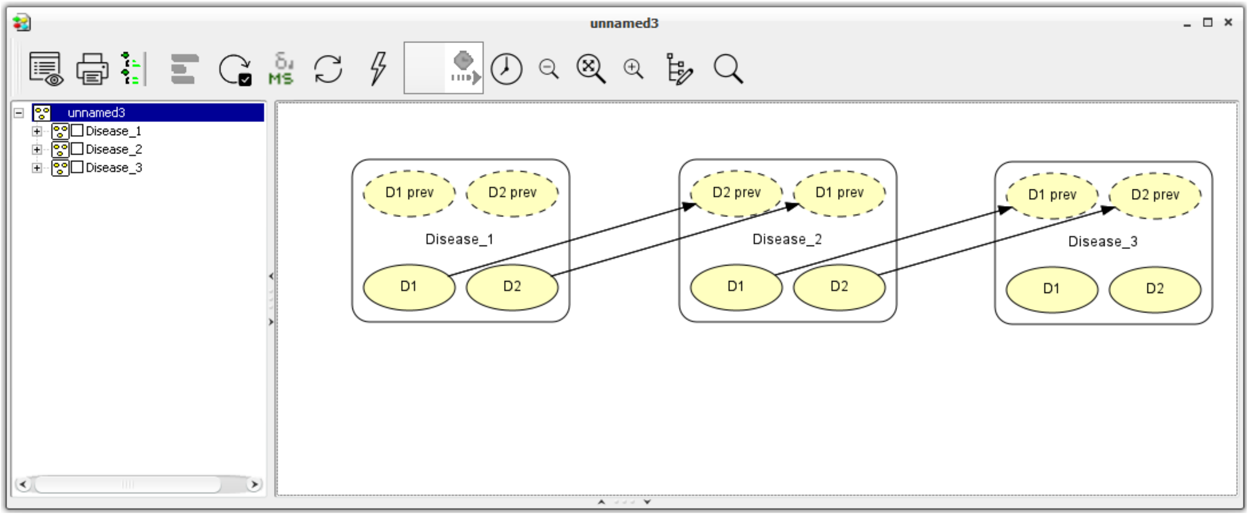


Figure 10: The network window in Run Mode. To the left is the Node List Pane (with all nodes collapsed) and to the right is the Network Pane.

You can view the probabilities of a node being in a certain state by expanding the node in the Node List Pane. You expand (collapse) a node by clicking its expand (collapse) icon in the Node List Pane, by double-clicking its node symbol in the Node List Pane, or by selecting (deselecting) it in the Network Pane. You can also expand (collapse) all nodes at once by pressing the *expand (collapse) node list tool* in the Tool Bar just to the right of the node properties tool.

Unlike basic nodes, instance nodes don't have belief monitors associated with them, as they represent entire (sub)networks. Instead we must expand the instance node, whereby we get to see the list of nodes of the (sub)network that the instance node represents (see Figure 11).

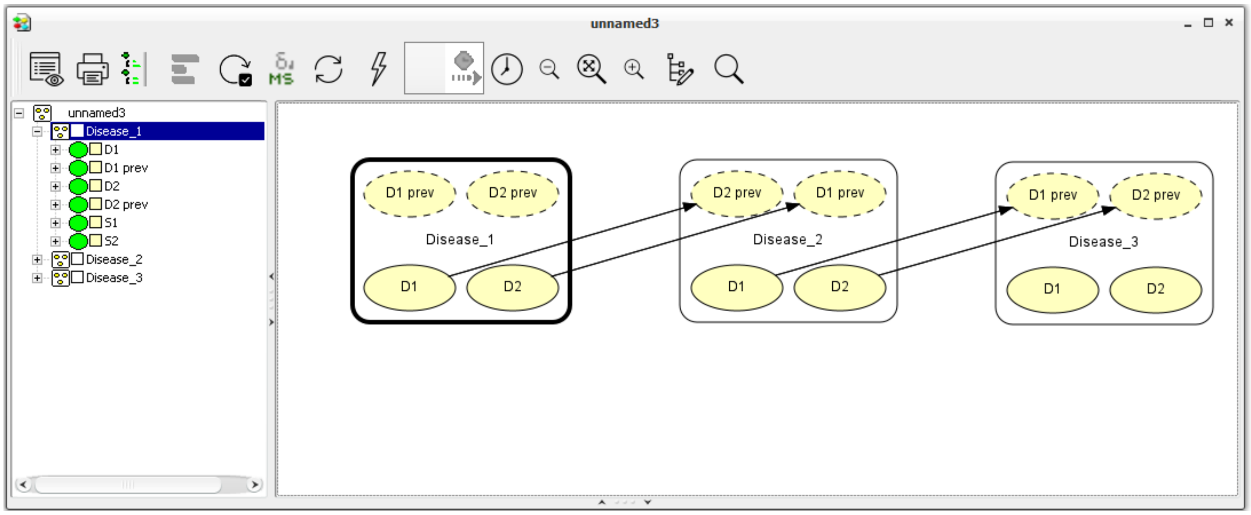


Figure 11: The nodes of the network represented by an instance node, get displayed in the Node List Pane when the instance node gets selected.

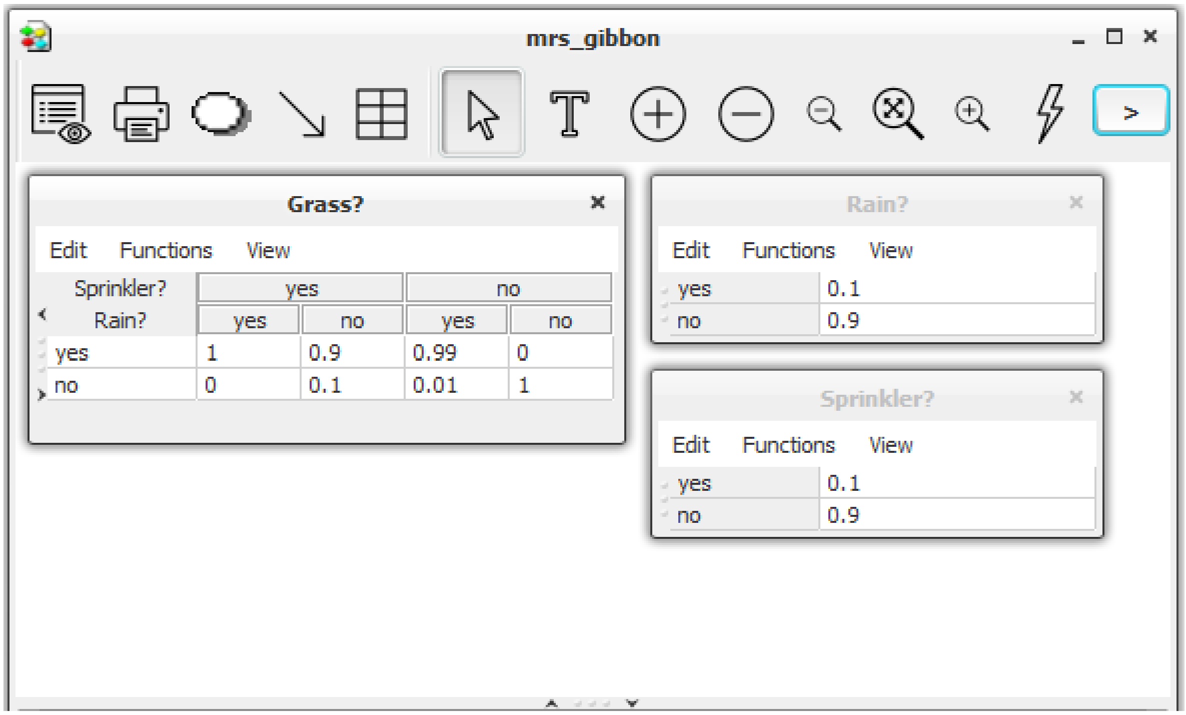
If the instances contains many nodes, locating a given node in the node list may be difficult. When this is the case, the instance can be *"traversed"* (page 121) by right-clicking on the node and selecting "Traverse Instance" (or selecting the node and choosing "Traverse Instance" from the network menu). This will open a window with compiled version of the class from which the instance node is created. In this window evidence can be inserted, and then transferred to any available instance. When the instances contains many nodes, it is possible to hide all private nodes from the node list to prevent the list from being to cluttered. This is done by selecting *Toggle Private Nodes* (page 242) in the View menu. For more details on the Run Mode, please consult the *How to Build BNs* (page 26) tutorial.

3.2.8 Node Table Tutorial

This tutorial explains the functionalities of node tables. To each chance node in a Bayesian network and each node in a (limited-memory) influence diagram is assigned a table, specifying a function associated with the node. Each decision node has a table specifying the initial policy for the decision variable represented by the node. These tables are referred to as *node tables*. The *types of the tables* (page 66) differ depending on the types of the nodes.

The HUGIN Graphical User Interface provides a number of powerful features for *displaying node tables* (page 70), including resizing, collapsing selected sets of columns, graphics display modes, etc. Furthermore, the user can choose between two different ways of organizing the tables:

- Framed tables (Figure 1)
- Tabbed tables (Figure 2)

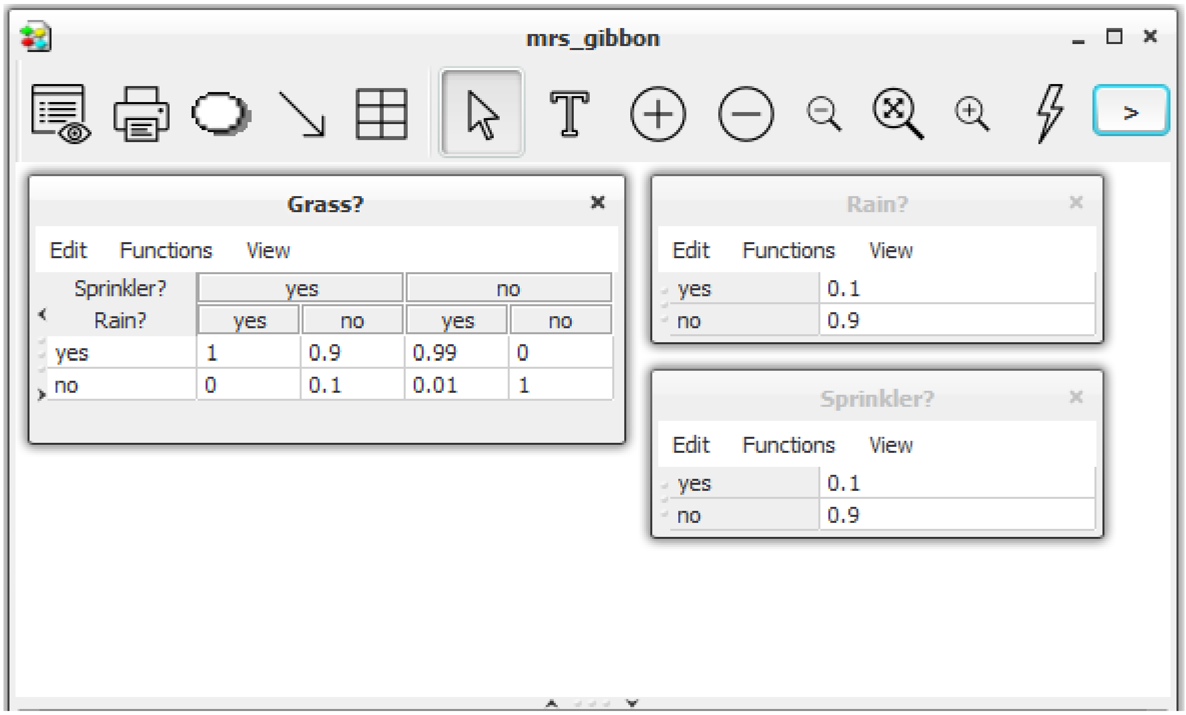


Sprinkler?	yes		no	
Rain?	yes	no	yes	no
yes	1	0.9	0.99	0
no	0	0.1	0.01	1

yes	0.1
no	0.9

yes	0.1
no	0.9

Figure 1: Tables displayed in framed mode



Grass?

Sprinkler?	yes		no	
Rain?	yes	no	yes	no
yes	1	0.9	0.99	0
no	0	0.1	0.01	1

Rain?

yes	0.1
no	0.9

Sprinkler?

yes	0.1
no	0.9

Figure 2: Tables displayed in tabbed mode

The framed mode has the advantage that several tables can be viewed at once, whereas the tabbed mode is a lot more compact (with several open tables) and it is easier to navigate the tables. The preferred table mode can be set in the *Preferences* (page 200), and it can be changed by selecting the “Toggle Table Mode” item in the *View Menu* (page 188) of a table.

The probabilities and utilities of discrete chance nodes and utility nodes, respectively, can be specified manually (i.e., directly through click-and-type operations) or they can be specified indirectly through specification of powerful *expressions* (page 150), which (if appropriate) can save a lot of work. The initial policy for a decision node can also be specified both manually and using expressions.

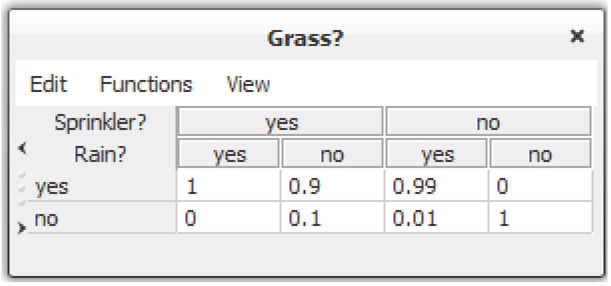
For all four kinds of tables, there is a menu bar containing an *Edit menu* (page 67), a *Functions menu* (page 68), and a *View menu* (page 69).

When working in *Edit Mode* (page 124) with the *Tables Pane* (page 239) open, the currently selected nodes will have their tables displayed in the Tables Pane.

Modes

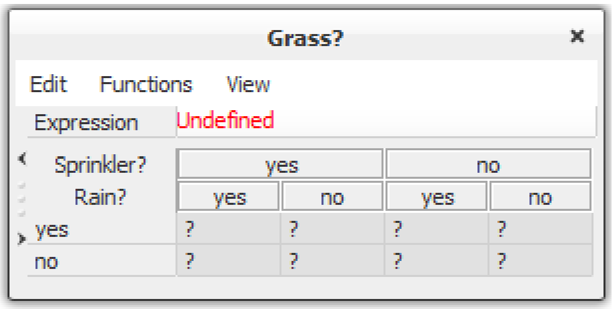
For *discrete chance nodes* (page 223), *discrete decision nodes* (page 224), and *utility nodes* (page 226) the node table has two different modes:

- *Manual mode* in which the numbers (probabilities or utilities) can be specified manually. This is the default mode, see Figure 1.
- *Expressions mode* in which the numbers can be specified through one or more mathematical expressions, see Figure 2.



Sprinkler?	yes		no	
Rain?	yes	no	yes	no
yes	1	0.9	0.99	0
no	0	0.1	0.01	1

Figure 1: A CPT in manual mode.



Expression Undefined

Sprinkler?	yes		no	
Rain?	yes	no	yes	no
yes	?	?	?	?
no	?	?	?	?

Figure 2: A CPT in expressions mode.

To switch between the two modes, select the appropriate item in the Functions menu of the node table, see Figure 3.

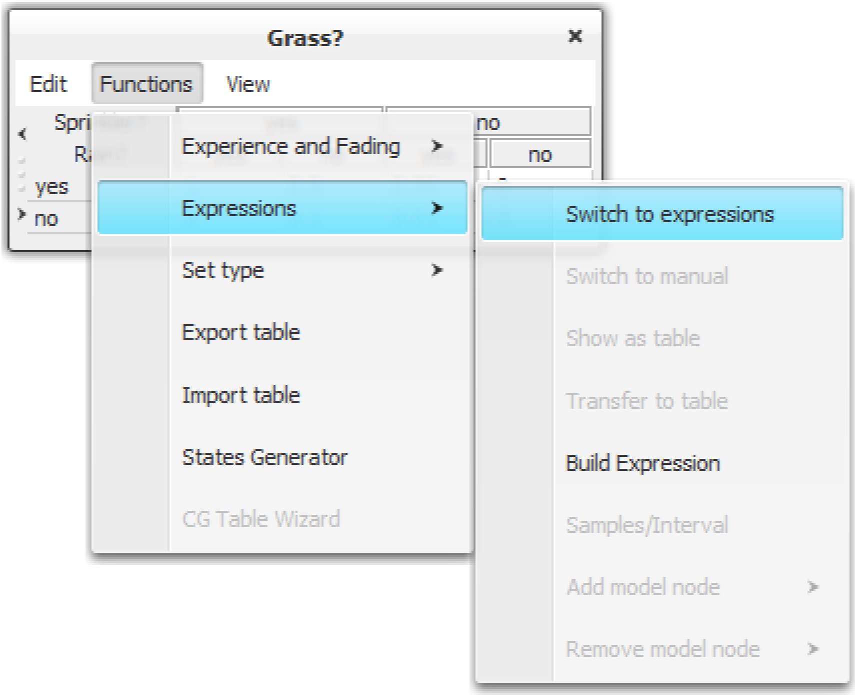
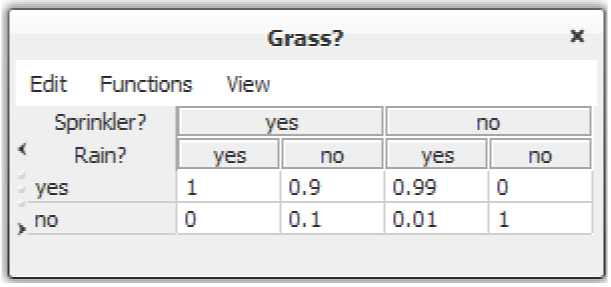


Figure 3: Selection of Expressions mode.

In Expressions mode there are one or more editable text fields in which expressions can be specified (one in each text field) either manually or through the *Expression Builder* (page 274). The number of text fields available depends on the number of states of the model nodes. If no model nodes have been specified, there will only be one text field (i.e., the same expression is used for all parent configurations).

Types of Tables

For *discrete chance nodes* (page 223) this function expresses a conditional probability distribution over the states of the node for each configuration of the states of the parents of the node, which are also discrete chance nodes. The conditional probability distribution (or marginal probability distribution in case of no parents) is represented as a *conditional probability table* (abbreviated CPT). Figure 4 shows an example of a CPT for the node Grass with parents Rain and Sprinkler.



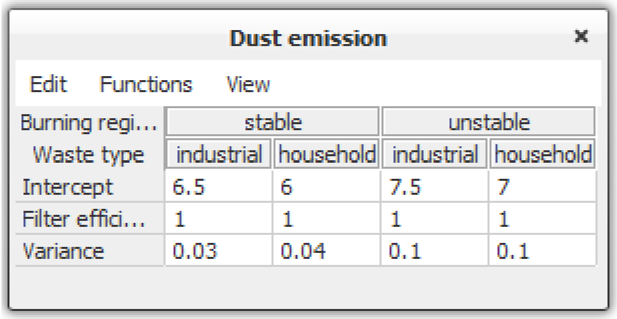
Sprinkler?	yes		no	
Rain?	yes	no	yes	no
yes	1	0.9	0.99	0
no	0	0.1	0.01	1

Figure 4: The CPT of node Grass with parents Rain and Sprinkler.

For each *continuous chance node* (page 224) with discrete parents I and continuous parents Z, the function expresses a (one-dimensional) Gaussian distribution conditional on the values of the parents:

$$P(Y|I=i, Z=z) = \mathcal{N}(\alpha(\rangle) + \beta(\rangle)^{\tau}\ddagger, \gamma(\rangle))$$

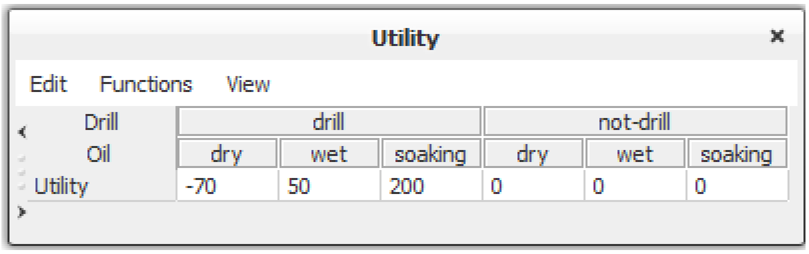
Note that the mean depends linearly on the continuous parents and that the variance does not depend on the continuous parents. However, both the linear function and the variance are allowed to depend on the discrete parents. (These restrictions ensure that exact inference is possible.) Thus, for each configuration of the discrete parents, a mean and a variance parameter must be specified, as well as a regression parameter for each continuous parent. Figure 5 shows the table for the continuous node Dust Emission (Emission) with discrete parents Burning Regimen (Burn) and Waste (Waste Type), and the continuous parent Filter Efficiency (Efficiency).



Burning regi...	stable		unstable	
Waste type	industrial	household	industrial	household
Intercept	6.5	6	7.5	7
Filter effici...	1	1	1	1
Variance	0.03	0.04	0.1	0.1

Figure 5: The table for continuous node Emission with parents Burn and Waste.

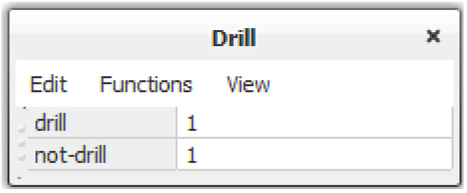
For each *utility node* (page 226), the function associated with the node expresses a utility function, where a utility value has to be specified for each configuration of its parents, which can be discrete chance and decision nodes. Figure 6 shows the table for the utility node Utility with the discrete chance node Oil and the discrete decision node Drill as parents.



Drill	drill			not-drill		
Oil	dry	wet	soaking	dry	wet	soaking
Utility	-70	50	200	0	0	0

Figure 6: The table for utility node Utility with parents Drill and Oil.

For each *decision node* (page 224) is associated a decision table, which is an encoding of the initial policy for the decision node. Figure 7 shows the table of a decision node, D, with decision options “drill” and “don’t drill”.



drill	1
not-drill	1

Figure 7: The decision table of a decision node D.

Edit Menu

The Edit menu provides the following functionalities (see Figure 8):

- *Copying* the selected cells of the table. Several cells can be selected by dragging the mouse cursor (i.e., move the mouse while keeping the left mouse button down). The selected cells will be highlighted.
- *Pasting* what has previous been copied. The destination cells of the paste operation are given as those covered by the rectangle extending right and down from the cell selected (i.e., the selected cell defines the upper-left cell of the destination cells).
- *Renaming / relabeling* of the node associated with the table. When selected, this menu item makes a dialog box appear through which the name and label of the node can be changed. Note that this dialog box is also available by right-clicking the title bar of the table.
- *Normalizing* the probability distributions of a CPT (i.e., making the probabilities in each distribution - one for each parent configuration - sum to 1).
- *Resetting* the values of all cells in the table to their default values (1 for CPTs and fading tables, and 0 for utility and experience tables). (See the section on *Learning - Adaptation* (page 244) for a description of experience and fading tables.)
- *Randomizing* the values of all cells of a CPT (i.e., assigning random values between 0 and 1 to all probabilities in each distribution, making sure that the values for each distribution sum to 1).
- *Creating an identity table* the values of each cell on the diagonal is assigned a value of 1 and every other cell is assigned a value of 0.

Please note that the copy and paste operations also work across applications (e.g., to/from the HUGIN node table from/to Excel, Word, Emacs, etc.).

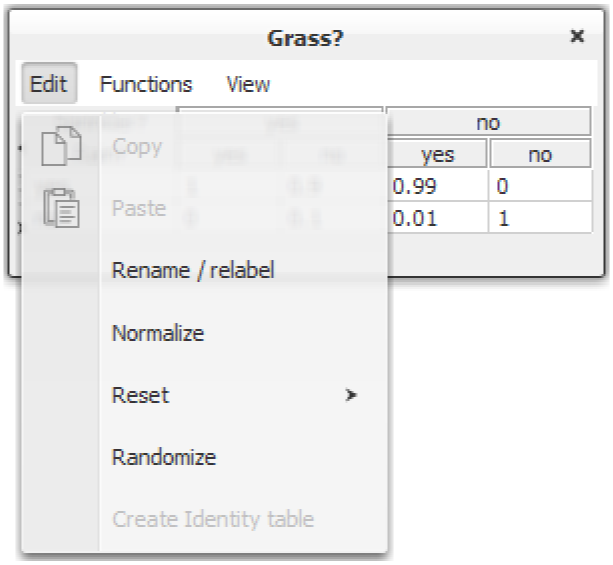


Figure 8: Edit menu for the node table.

Functions Menu

The Functions menu provides the following functionalities (see Figure 9): Creating and deleting experience and fading tables is performed via the Experience and Fading submenu (see Figure 10). (See the section on Learning - Adaptation for a description of experience and fading tables.) Specifying expressions (models) for how to generate the values of the cells of CPTs and utility tables. See section Expressions for details. Setting the type of the node to one of Labelled, Boolean, Numbered, or Interval if the node is a discrete chance node or a discrete decision node. See section Node Type for details on node types. Exporting the table (i.e., the values of the cells of the table) to either an x-delimited text file, where ‘x’ can be ‘,’, ‘<space>’, or ‘<tab>’.

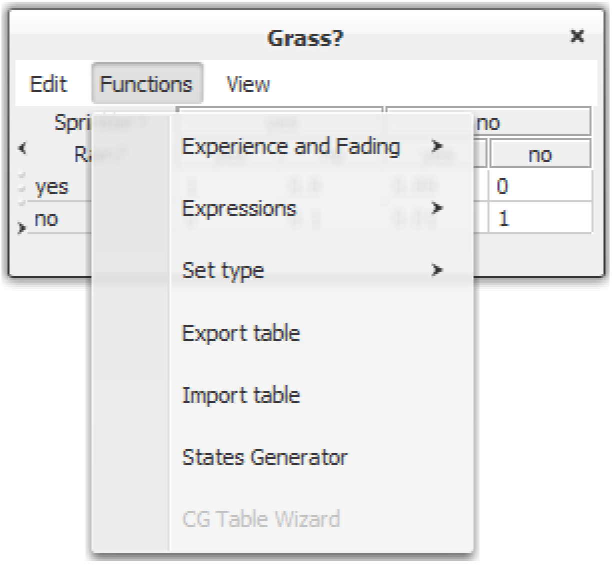
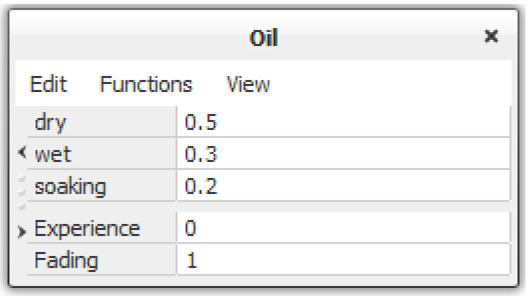


Figure 9: Functions menu for the node table.



dry	0.5
wet	0.3
soaking	0.2
Experience	0
Fading	1

Figure 10: Experience and fading tables created and shown.

View Menu

The View menu provides the following functionalities (see Figure 11):

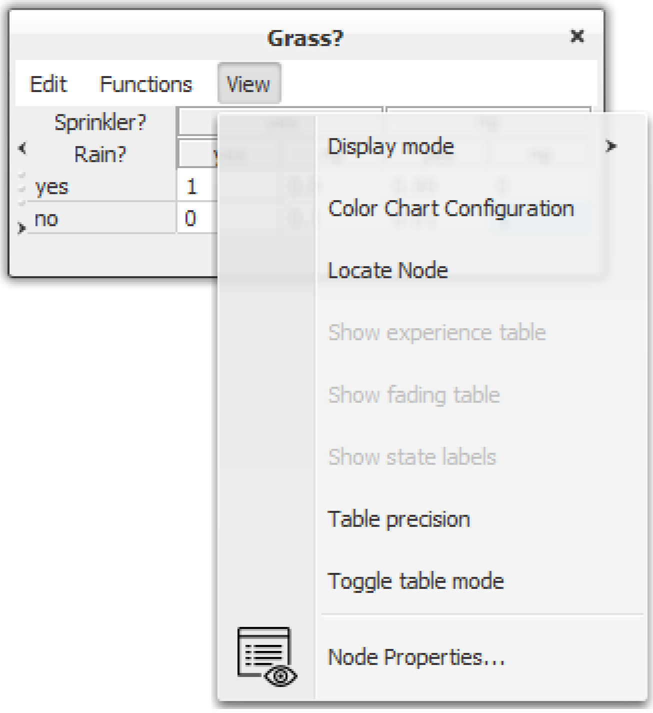


Figure 11: View menu for the node table.

Below are some test paragraphs.

- *Selecting display mode*. A number of different modes for displaying the numbers of the table are available:
  - *Normal mode* in which only the numbers are displayed.
  - *Bar mode* in which each number is overlaid by a bar of length corresponding to the number (see Figure 16).
  - *Pure bars mode* in which only the bars are shown (see Figure 17).
  - *Auto normalize* can be turned on, meaning that when dragging a bar the other bars belonging to the distribution are automatically adjusted to make sure the numbers (represented by the bar lengths) sum to 1.
- *Locate Node* selects and scrolls to the node whose table is viewed.

- *Show / hide experience table* toggles between showing or hiding the experience table, if any.
- *Show / hide fading table* toggles between showing or hiding the fading table, if any.
- *Show state labels* shows state labels for numbered or interval nodes (instead of state values). This allows for emulation of "ordinal" nodes. That is, a node can have the state labels "low", "medium", and "high", and these will have a natural ordering which can be used in expressions.
- *Table Precision* displays a dialog for setting the number of decimals shown in the table.
- *Toggle table* mode toggles between framed and tabbed tables.
- *Node Properties* shows the *properties* (page 216) for the node whose table is viewed.

See the *next section* (page 70) for more details on display modes.

Displaying a Node Table

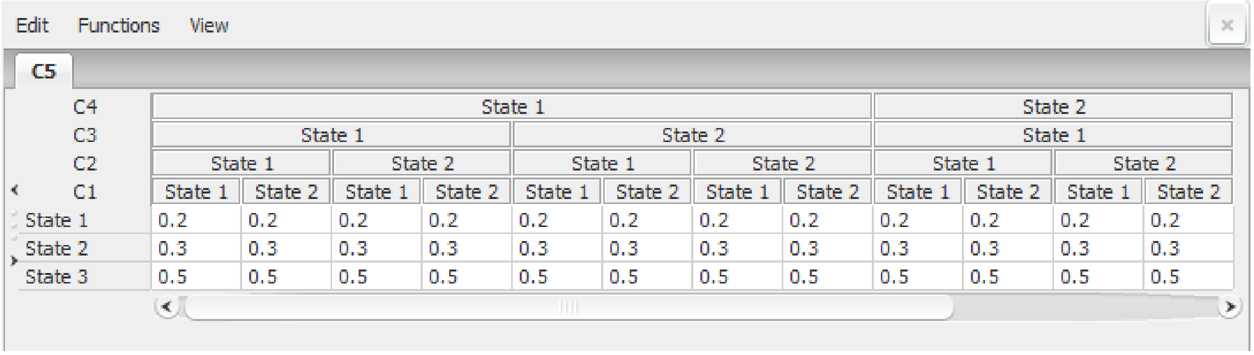
When the *Tables Pane* (page 239) is open, the tables for the nodes selected in the *Network Pane* (page 177) will be displayed. The HUGIN Graphical User Interface has a number of powerful features for displaying node tables:

- Resizing through dragging
- Re-ordering of parents
- Collapsing the display of subsets of parent configurations
- Displaying probabilities as numbers, bars, or both.

A node table frame (i.e., the internal window displaying the table) can be resized arbitrarily by clicking and holding (i.e., dragging) the left mouse at the border of the frame. The width of the cells of the table resizes correspondingly. The width of the first column (containing parent and state names) can be resized by dragging the left mouse button at the border between the first and the second column. This can be useful if the parent and/or state names are very long or very short.

The order of appearance of the parents in the node table can be changed in *Table tab* (page 234) of the Node Properties pane. This can be useful for grouping together sets of distributions.

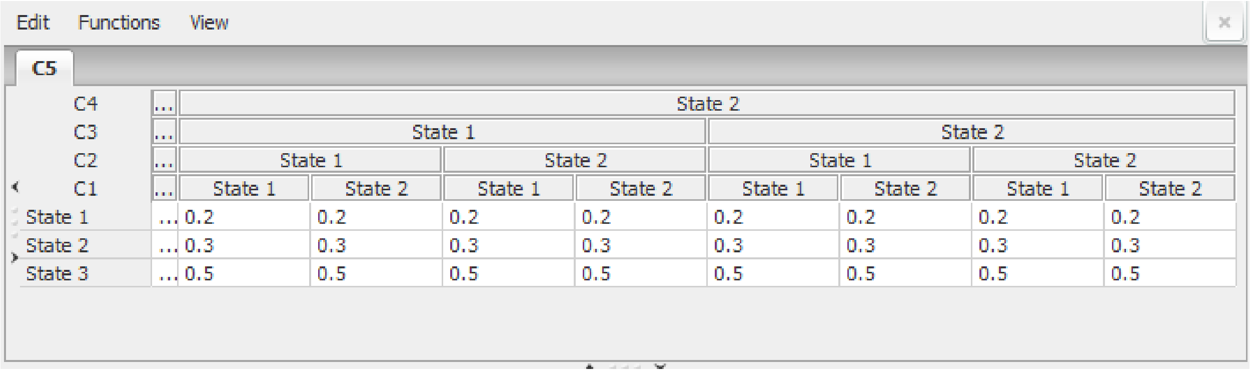
Also, for better viewing of selected sets of distributions, sets of columns (corresponding to sets of parent configurations) can be collapsed by double-clicking the appropriate labels for parent states. Figure 12 shows (part of) the table for node C5 with parents C1, C2, C3, and C4. Now, suppose we're interested in working with the probability distributions for C1 in state "State 2". One option would be to use the horizontal scroll bar. Another, much easier and more convenient approach would be to collapse the part of the table where C1 is in state "State 1". This can be achieved by double-clicking the text field labeled "State 1" in the C1 row.



C4	State 1								State 2			
C3	State 1				State 2				State 1			
C2	State 1		State 2		State 1		State 2		State 1		State 2	
C1	State 1	State 2	State 1	State 2	State 1	State 2	State 1	State 2	State 1	State 2	State 1	State 2
State 1	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
State 2	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
State 3	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

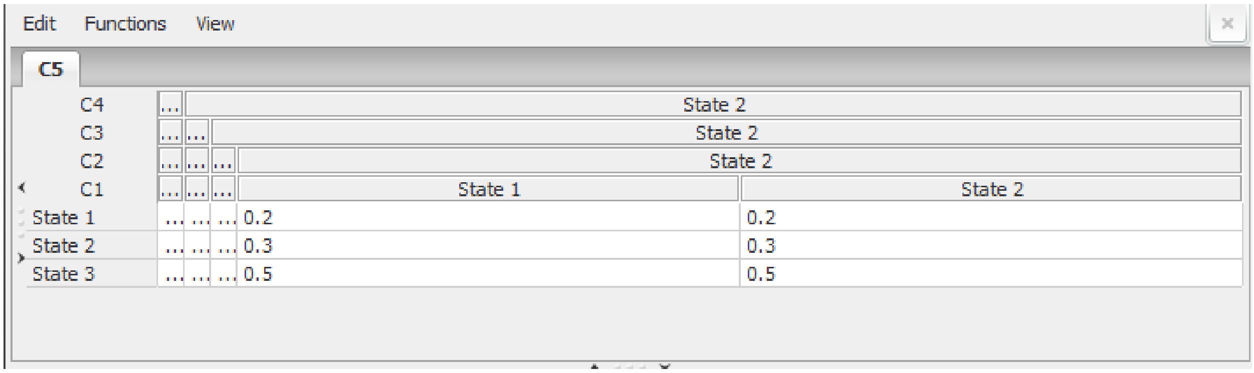
Figure 12: The table for node C5 with all columns collapsed for which the state of parent node C1 equals "State 2".

Figure 13 shows the resulting table. Note that the width of the cells has changed, allowing us to see all digits after the decimal point. Likewise, we can collapse other columns. In general, the cells in the parent states section can be double-clicked for all parent rows but the last, making the corresponding columns collapse. Figure 14 shows the table with all columns collapsed where the states for all parent nodes but C4 equals "State 1".



C4	...	State 2							
C3	...	State 1				State 2			
C2	...	State 1		State 2		State 1		State 2	
C1	...	State 1	State 2	State 1	State 2	State 1	State 2	State 1	State 2
State 1	...	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
State 2	...	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
State 3	...	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

Figure 13: The table for node C5 with all columns collapsed for which the state of parent node C1 equals "State 1".



C4	...	State 2			
C3	...	...	State 2		
C2	...	...	...	State 2	
C1	...	...	...	State 1	State 2
State 1	...	...	...	0.2	0.2
State 2	...	...	...	0.3	0.3
State 3	...	...	...	0.5	0.5

Figure 14: The table for node C5 with all columns collapsed for which the states of parent nodes C1, C2, and C3 equal "State 1"

A collapsed column can be expanded by double-clicking the top cell of the column. Please note that when changing the order of appearance of the parents (see above), any collapsed columns will be expanded. Sometimes, viewing or even specifying probability distributions graphically instead of numerically is preferable. The HUGIN Graphical User Interface allows the user to select among three different display modes: The numerical mode (denoted the Normal mode), the numerical and graphics mode (denoted the Bar mode), and the graphics mode (denoted the Pure bars mode). The desired mode can be selected under the View menu, see Figure 15.

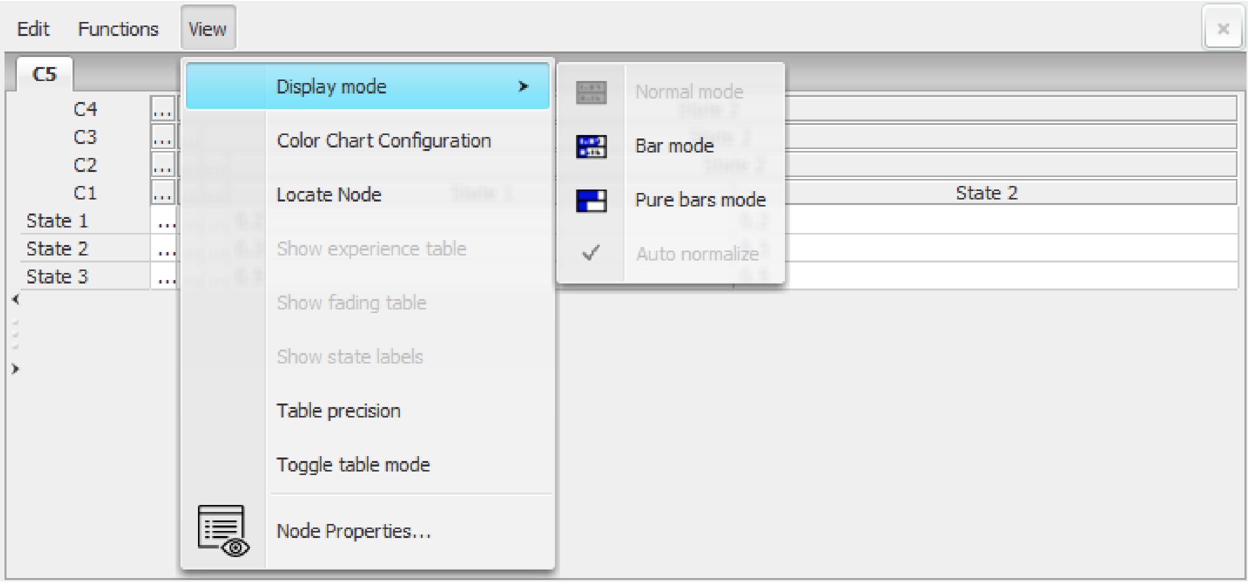
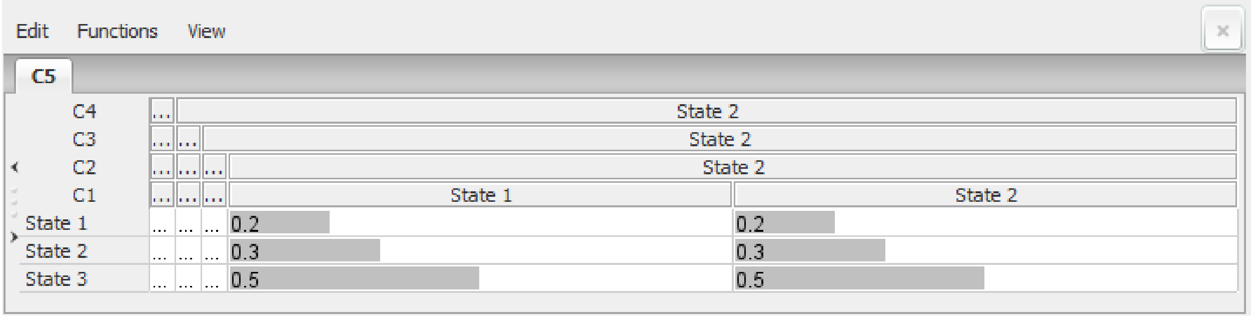


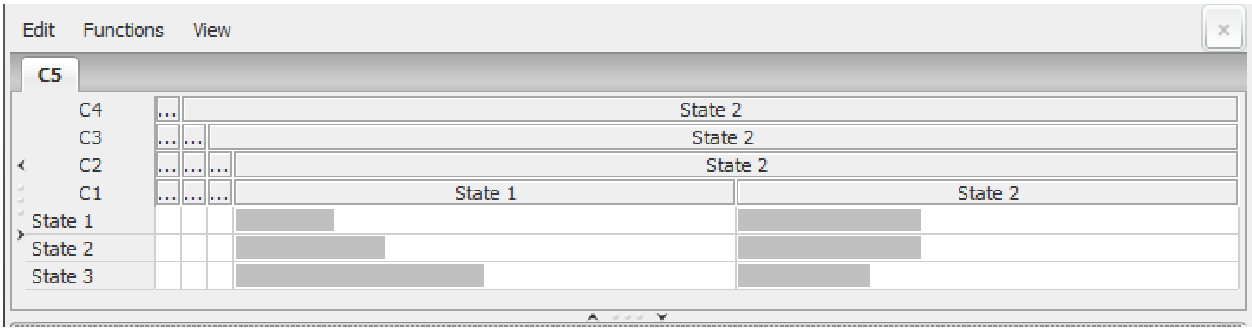
Figure 15: The probabilities of a table for a discrete chance node can be displayed numerically, graphically, or both numerically and graphically.

Figures 16 and 17 show the table of Figure 14 in Bar mode and Pure bars mode, respectively.



C4	...	State 2			
C3	...	...	State 2		
C2	...	...	...	State 2	
C1	...	...	...	State 1	State 2
State 1	...	...	...	0.2	0.2
State 2	...	...	...	0.3	0.3
State 3	...	...	...	0.5	0.5

Figure 16: The CPT for node C5 displayed in Bar mode.



C4	...	State 2			
C3	...	...	State 2		
C2	...	...	...	State 2	
C1	...	...	...	State 1	State 2
State 1					
State 2					
State 3					

Figure 17: The CPT for node C5 displayed in Pure Bars mode.

Graphical specification of probabilities is performed by dragging the bars with the left mouse button. When Bar mode or Pure bars mode have been selected, the distributions can be set to auto-normalize when the probabilities are changed.

This feature can be selected under the Display mode submenu of the View menu.

Please note that the graphical viewing and specification mode is available only in tables for discrete chance and decision nodes.

3.2.9 Table Generator Tutorial

This tutorial shows you how the Table Generator functionality can be used to simplify the specification of conditional probability tables (CPTs) for discrete chance nodes, utility tables for utility nodes, and initial policies for decision nodes. The tutorial describes how these tables can be described compactly using *models* and *expressions*. This is particularly useful when the conditional probability distribution for a variable follows (at least approximately) certain functional or distributional forms. In such cases it is cumbersome to specify the conditional probability table (CPT) manually.

Since the type of expressions available depends on the type of the node, it will be illuminating to discuss the different *node sub-types* (page 73) that are supported by the HUGIN Decision Engine.

A model consists of a list of discrete nodes and a set of expressions (one expression for each configuration of the states of the nodes). The list of nodes of a model is referred to as *model nodes* (page 74).

An expression is built using standard statistical distributions (e.g., Normal, Binomial, Beta, Gamma, etc.), arithmetic operators, standard mathematical functions (e.g., logarithmic, exponential, trigonometric, and hyperbolic functions), logical operators (e.g., and, or, if-then-else), and relations (e.g., less-than, equals).

Expressions can be constructed manually (*see syntax for expressions* (page 84)) or by the assistance of the *Expression Builder* (page 274), which guides the user through the construction, using series of dialog boxes.

Sub-Typing of Discrete Nodes

The different operators used in an expression have different return types and different type requirements for arguments. Thus, in order to provide a rich language for specifying expressions, it is convenient to have a classification of the discrete chance and decision nodes into different groups (see also section *Node Type* (page 216)):

- *Labelled* nodes can be used in equality comparisons and to express deterministic relationships. For example, a labelled node C1 with states “State 1” and “State 2” can appear in an expression like ‘if (C1 == “State 1”, Distribution (0.2, 0.8), Distribution (0.4, 0.6))’ for P(C2 | C1), where C2 is another discrete chance node with two possible states.
- *Boolean* nodes represent the truth values ‘false’ and ‘true’ (in that order) and can be used in logical operators. For example, for a Boolean node, B0, being a the logical OR of its (Boolean) parents, B1, B2, and B3, P(B0|B1, B2, B3) can be specified simply as ‘or (B1, B2, B3)’.
- *Numbered* nodes represent increasing sequences of numbers (integers or reals) and can be used in arithmetic operators, mathematical functions, etc.
- *Interval* nodes represent disjoint intervals on the real line and can be used in the same way as numbered nodes. In addition, they can be used when specifying the intervals over which a continuous quantity are to be discretized.

Numbered nodes and interval nodes are jointly referred to as numeric nodes.

Constant Values

The following kinds of constants can be used in expressions:

- *State labels* (i.e., sequences of characters encapsulated in quotation characters (“)).
- *Numeric values* (including integers, reals, and intervals of integers and reals). For example, a valid numeric expression could be “X + 3.4” (without the quotation characters; otherwise, it will be interpreted as a state label), where X is the name of a *numeric node* (page 73).
- *Infinity*. Positive infinity is denoted by “inf” (without the quotation characters), and negative infinity by “-inf”.
- *Boolean values*: “false” and “true” (without the quotation characters). Notice that the Boolean constants must be indicated using lower case.

Model Nodes

Quite often one needs different expressions depending on the states of one or more parent nodes. Using a number of nested if-then-else expressions is one way of coping with this. The resulting expression, however, often gets very complicated and hence difficult to evaluate by visual inspection and, thus, difficult to maintain.

To simplify complicated expressions, the notion of *model nodes* can be quite useful.

As mentioned above, a model for a CPT consists of a list of model nodes and an expression for each configuration of the states of the model nodes. That is, if there are no model nodes, the model contains a single expression.

The model nodes for a particular model constitute a subset of the parents of the node to which the model belongs. This subset is specified under the *Table tab* (page 234) of the Node Properties dialog box.

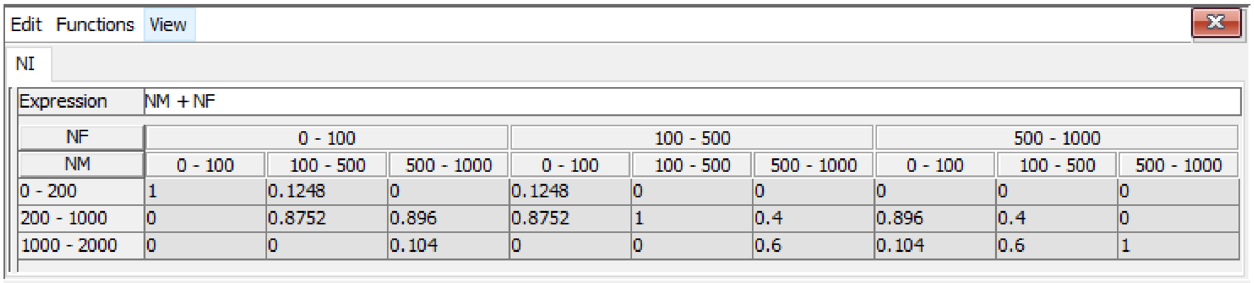
An example of the use of model nodes is given in the below example, *Discretization of a Random Variable* (page 76).

Simple Examples

Number of People

Assume that in some application we have probability distributions over the number of males and females, where the distributions are defined over intervals [0 - 100), [100 - 500), [500 - 1000), and that we wish to compute the distribution over the total number of individuals given the two former distributions. It is a simple but tedious task to specify P(NI | NM, NF), where NI, NM, and NF stands for number of individuals, number of males, and number of females, respectively. A much more expedient way of specifying this conditional probability distribution would be to let NM and NF be represented as interval nodes with states [0 - 100), [100 - 500), and [500 - 1000), and to let NI be represented as an interval node with states [0 - 200), [200 - 1000), and [1000 - 2000), for example, and then define P(NI | NM, NF) through the simple expression NI = NM + NF.

To specify that expression, we first select Expressions mode, see the *Node Table tutorial* (page 63). Next, we activate the expression text field by clicking it by the left mouse button. Finally, we type the string “NM + NF” (without the quotation characters), using the keyboard. Figure 1 shows the resulting table for node NI with this expression and the resulting CPT, where the numbers displayed are derived from the expression by selecting menu item “Show as table” from the “Expressions” submenu of the “Functions” menu.



Expression: NM + NF

NF	0 - 100			100 - 500			500 - 1000		
NM	0 - 100	100 - 500	500 - 1000	0 - 100	100 - 500	500 - 1000	0 - 100	100 - 500	500 - 1000
0 - 200	1	0.1248	0	0.1248	0	0	0	0	0
200 - 1000	0	0.8752	0.896	0.8752	1	0.4	0.896	0.4	0
1000 - 2000	0	0	0.104	0	0	0.6	0.104	0.6	1

Figure 1: A CPT specified via an expression. The CPT is specified for a discrete chance node NI that has parents NM and NF.

Fair or Fake Die?

As another example, consider the problem of computing the probabilities of getting n 6's in n rolls with a fair die and a fake die, respectively. A random variable, X, denoting the number of 6's obtained in n rolls with a fair die is binomially distributed with parameters (n, 1/6). Thus, the probability of getting k 6's in n rolls with a fair die is P(X = k), where P is a Binomial(n, 1/6). Assuming that for a fake die the probability of getting 6 eyes in one roll is 1/5, the probability of getting k 6's in n rolls with a fake die is Q(X = k), where Q is a Binomial(n, 1/5).

A Bayesian-network model of this problem is shown in Figure 2, where the node n6s (labeled "# 6's") depends on the number of rolls, represented by the node n_rolls (labeled "# rolls"), and on the probability of the die being fake, represented by the node fake_die (labeled "Fake die?"). Now, if we let n_rolls be a numbered node with states 1, 2, 3, 4, 5, let fake_die be a boolean node, and let n6s be a numbered node with states 0, 1, 2, 3, 4, 5, then P(n6s | n_rolls, fake_die) can be specified very elegantly using the expression P(n6s | n_rolls, fake_die) = Binomial (n_rolls, if (fake_die, 1/5, 1/6)).

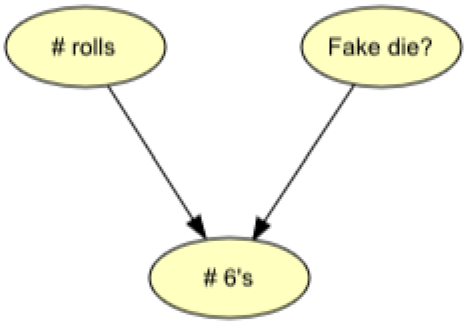
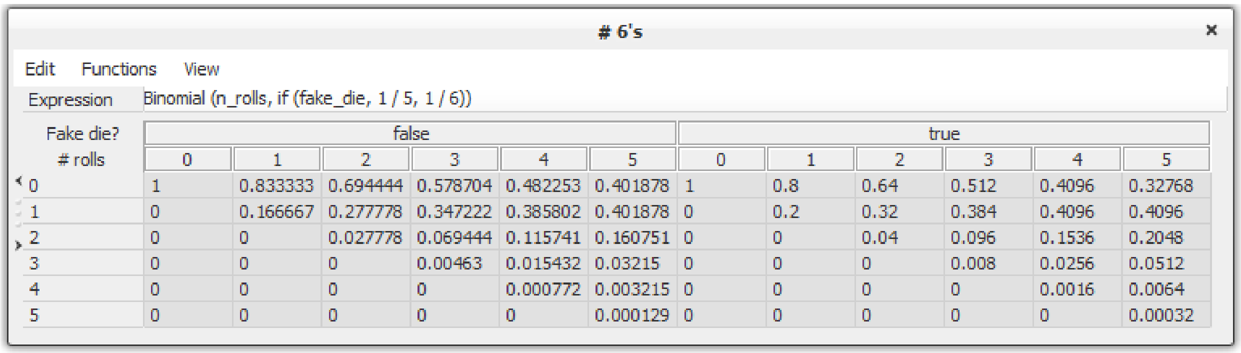


Figure 2: A Bayesian-network model for the fake die problem.

To specify that expression, we may proceed as in the *Number of People* (page 74) example or we may wish to use the *Expression Builder* (page 274) (activated by selecting the "Build Expression" item of the "Expressions" submenu of the "Functions" menu) . The result of the specification and the derived probabilities are shown in Figure 3.



Expression Binomial (n_rolls, if (fake_die, 1 / 5, 1 / 6))

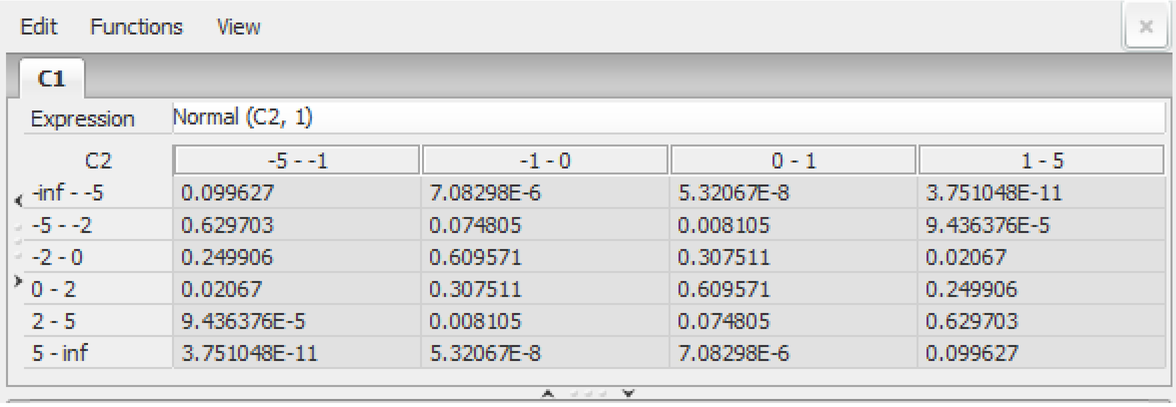
Fake die?	false						true					
# rolls	0	1	2	3	4	5	0	1	2	3	4	5
0	1	0.833333	0.694444	0.578704	0.482253	0.401878	1	0.8	0.64	0.512	0.4096	0.32768
1	0	0.166667	0.277778	0.347222	0.385802	0.401878	0	0.2	0.32	0.384	0.4096	0.4096
2	0	0	0.027778	0.069444	0.115741	0.160751	0	0	0.04	0.096	0.1536	0.2048
3	0	0	0	0.00463	0.015432	0.03215	0	0	0	0.008	0.0256	0.0512
4	0	0	0	0	0.000772	0.003215	0	0	0	0	0.0016	0.0064
5	0	0	0	0	0	0.000129	0	0	0	0	0	0.00032

Figure 3: The CPT for the fake die problem specified very compactly using a simple expression.

Notice that we could equivalently specify P(n6s | n_rolls, fake_die) = if (fake_die, Binomial (n_rolls, 1/5), Binomial(n_rolls, 1/6)).

Discretization of a Random Variable

Assume that P(C1 | C2) can be approximated by a Normal distribution with mean given by C2 and with variance 1, where C2 is an interval variable with states [-5,-1), [-1,0), [0,1), [1,5). If the discretization of C1 given by the intervals [-infinity,-5), [-5,-2), [-2,0), [0,2), [2,5), [5,infinity) is found to be suitable, then we can specify P(C1 | C2) simply as Normal(C2, 1), see Figure 4.

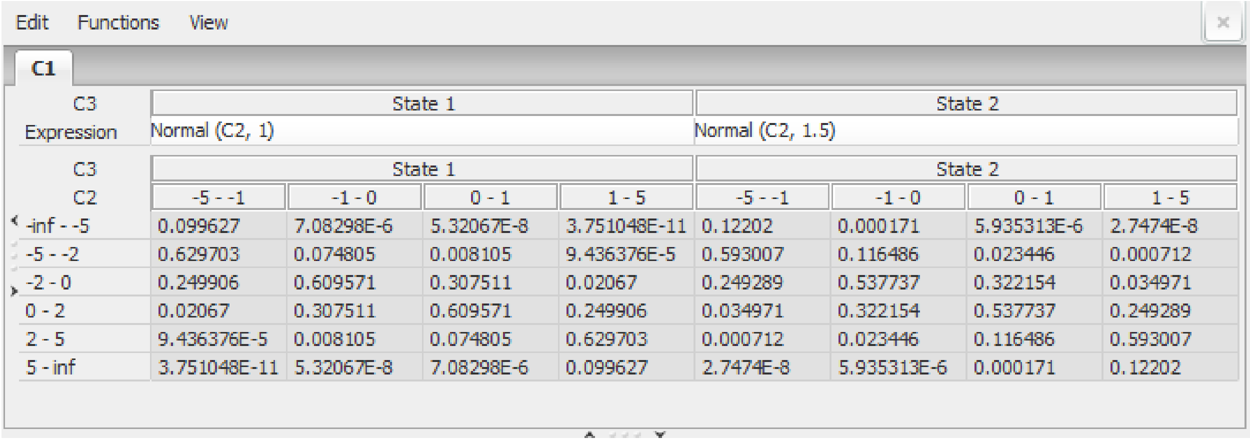


Expression Normal (C2, 1)

C2	-5 - -1	-1 - 0	0 - 1	1 - 5
-inf - -5	0.099627	7.08298E-6	5.32067E-8	3.751048E-11
-5 - -2	0.629703	0.074805	0.008105	9.436376E-5
-2 - 0	0.249906	0.609571	0.307511	0.02067
0 - 2	0.02067	0.307511	0.609571	0.249906
2 - 5	9.436376E-5	0.008105	0.074805	0.629703
5 - inf	3.751048E-11	5.32067E-8	7.08298E-6	0.099627

Figure 4: The CPT for variable C1 specified through discretization of a Normal distribution with mean given by the (interval) parent variable C2 and with variance 1.

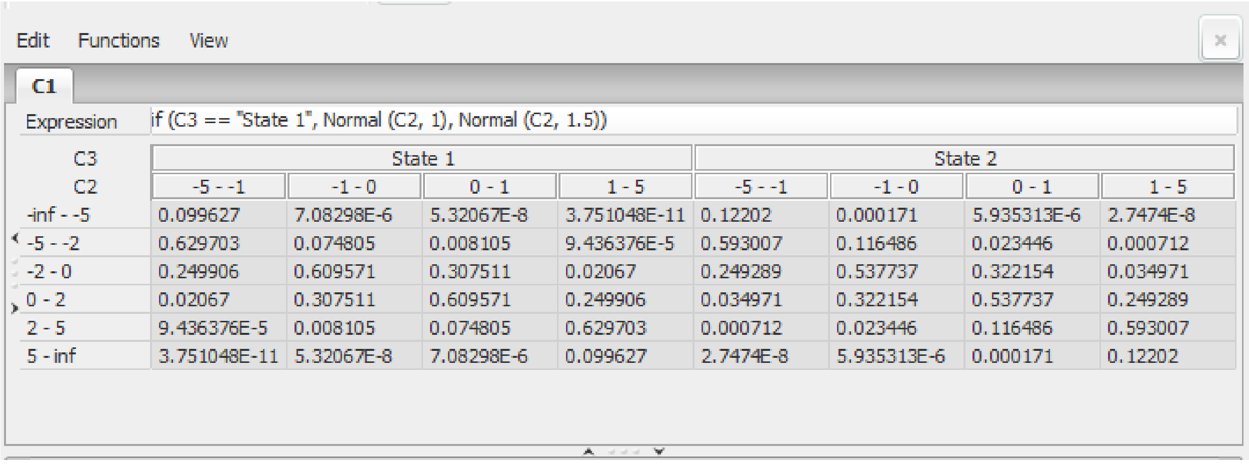
If, in addition, C1 has another parent, say C3, which is a labelled node with states, say, “State 1” and “State 2” and that the variance of the Normal distribution is 1 if C3 is in state “State 1” and 1.5 if C3 is in state “State 2”, then we can define C3 as a so-called model node, which allows us to specify different expressions for the different states of C3, see Figure 5.



C3	State 1				State 2			
Expression	Normal (C2, 1)				Normal (C2, 1.5)			
C3	State 1				State 2			
C2	-5 - -1	-1 - 0	0 - 1	1 - 5	-5 - -1	-1 - 0	0 - 1	1 - 5
-inf - -5	0.099627	7.08298E-6	5.32067E-8	3.751048E-11	0.12202	0.000171	5.935313E-6	2.7474E-8
-5 - -2	0.629703	0.074805	0.008105	9.436376E-5	0.593007	0.116486	0.023446	0.000712
-2 - 0	0.249906	0.609571	0.307511	0.02067	0.249289	0.537737	0.322154	0.034971
0 - 2	0.02067	0.307511	0.609571	0.249906	0.034971	0.322154	0.537737	0.249289
2 - 5	9.436376E-5	0.008105	0.074805	0.629703	0.000712	0.023446	0.116486	0.593007
5 - inf	3.751048E-11	5.32067E-8	7.08298E-6	0.099627	2.7474E-8	5.935313E-6	0.000171	0.12202

Figure 5: Similar to Figure 4, except that C1 has got a new parent, C3, defined as a model node, allowing the specification of different expressions for the different states of C3.

Notice that the use of model nodes is not strictly necessary, as we can alternatively condition on the states of the (model) node(s). The use of model nodes, however, often makes the specification much less cluttered and easier to read and maintain. For example, if we don't specify C3 as a model node, P(C1 | C2, C3) can be specified through the expression P(C1 | C2, C3) = if (C3 == "State 1", Normal (C2, 1), Normal (C2, 1.5)), see Figure 6.



Expression	if (C3 == "State 1", Normal (C2, 1), Normal (C2, 1.5))							
C3	State 1				State 2			
C2	-5 - -1	-1 - 0	0 - 1	1 - 5	-5 - -1	-1 - 0	0 - 1	1 - 5
-inf - -5	0.099627	7.08298E-6	5.32067E-8	3.751048E-11	0.12202	0.000171	5.935313E-6	2.7474E-8
-5 - -2	0.629703	0.074805	0.008105	9.436376E-5	0.593007	0.116486	0.023446	0.000712
-2 - 0	0.249906	0.609571	0.307511	0.02067	0.249289	0.537737	0.322154	0.034971
0 - 2	0.02067	0.307511	0.609571	0.249906	0.034971	0.322154	0.537737	0.249289
2 - 5	9.436376E-5	0.008105	0.074805	0.629703	0.000712	0.023446	0.116486	0.593007
5 - inf	3.751048E-11	5.32067E-8	7.08298E-6	0.099627	2.7474E-8	5.935313E-6	0.000171	0.12202

Figure 6: Similar to Figure 5, except that instead of specifying C3 as a model node the two expressions are merged into one expression, where we condition on the states of C3.

Operators and Functions

The basic operators and functions available for composing expressions are list below.

Binary Numeric Operators

The following binary (infix) operators can be applied to numeric expressions.

- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- ^ (power)

Examples:

- C1 + C2
- C1 ^ 3

where C1 and C2 are numeric nodes (i.e., numbered nodes and/or interval nodes).

Unary Numeric Operators

An numeric expression can be negated using the unary negation operator:

- - (negation)

Binary Comparison Operators

The following binary (infix) operators can be used for comparing labels (i.e., strings), numbers, and Booleans (both operands must be of the same type). Only the equality operators (i.e., = and !=) may be applied to labels and Boolean expressions. Each of the operators returns a Boolean value.

- = (equals)
- == (equals)
- != (not equals)
- <> (not equals)
- < (less than)
- > (greater than)
- <= (less than or equals)
- >= (greater than or equals)

Examples:

- C1 == C2
- C1 > 5

where C1 and C2 are numeric nodes (i.e., numbered nodes and/or interval nodes).

Min and Max Functions

The following functions compute the minimum or maximum of a list of numeric expressions.

- min(x1, x2, . . . , xn) returns the minimum of the argument expressions.
- max(x1, x2, . . . , xn) returns the maximum of the argument expressions.

Examples:

- min(C1, C2, 10)
- max(min(C1, C2, 10), max(C3, C4))

where C1, . . . , C4 are numeric nodes (i.e., numbered nodes and/or interval nodes).

Standard Mathematical Functions

The following list contains standard mathematical functions, which can be applied to a single numeric expression.

- log(x) returns the natural (i.e., base e) logarithm to x.
- log2(x) returns the base 2 logarithm to x.
- log10(x) returns the base 10 logarithm to x.
- exp(x) returns the exponential to x (i.e., e^x).
- sin(x) returns the sine of x.
- cos(x) returns the cosine of x.
- tan(x) returns the tangent of x.
- sinh(x) returns the hyperbolic sine of x.
- cosh(x) returns the hyperbolic cosine of x.
- tanh(x) returns the hyperbolic tangent of x.
- sqrt(x) returns square root of x.
- abs(x) returns the absolute value of x.

Examples:

- log(C1)
- abs(sin(C1))

where C1 is a numeric node (i.e., a numbered node or an interval node).

Floor and Ceiling Functions

The floor and ceiling functions round the result of real numeric expressions to integers.

- floor(x) returns the greatest integer less than or equal to x.
- ceil(x) returns the smallest integer greater than or equal to x.

Examples:

- floor(C1 * C2)
- ceil(C1 ^ 2)

where C1 and C2 are numeric nodes (i.e., numbered nodes and/or interval nodes).

Modulo Function

The modulo function gives the remainder of a division of two numeric expressions. Of course, the divisor expression must be non-zero.

- mod(x,y) returns x - y * floor(x / y), where x and y can be arbitrary real numbers with y != 0.

Example:

- mod(C1, C2 ^ 2)

where C1 and C2 are numeric nodes (i.e., numbered nodes and/or interval nodes).

If-Then-Else

Conditional expression (with three arguments) can be specified:

- if(px, tx, fx), where px must be a Boolean expression, and the second and third arguments must have the same type. If px evaluates to 'true', the value of expression tx is returned; otherwise, the value of expression fx is returned. The type of the if-expression is the type of tx and fx.

Examples:

- if(FakeDie, Binomial(n, 1/2), Binomial(n, 1/6))
- if(C1 == C2, Distribution(1, 2), Distribution(1, 3))

where FakeDie is a Boolean node, n is a numeric node, and C1 and C2 are nodes of arbitrary (but identical) type.

Logical Operators

The following standard logical operators are available. They all take Boolean expressions as arguments.

- and(x1, x2, . . . , xn) returns 'true' if all argument expressions evaluate to 'true'.
- or(x1, x2, . . . , xn) returns 'true' if at least one of the argument expressions evaluate to 'true'.
- not(x) returns 'true' if x evaluates to 'false'; otherwise, returns 'false'.

The evaluation of the argument expressions of 'and' is done sequentially, and the evaluation terminates whenever an argument evaluates to 'false'. Similarly, the evaluation of the argument expressions of 'or' terminates whenever an argument evaluates to 'true'.

Example:

- and(C1, or(C2, not(C3)))

where C1, C2 and C3 are Boolean nodes.

Continuous Statistical Distributions

A number of continuous statistical distributions are available. See *Continuous Distributions* (page 81) for details.

- Normal
- LogNormal
- Beta
- Gamma
- Exponential
- Weibull
- Uniform
- Triangular
- PERT

Discrete Statistical Distributions

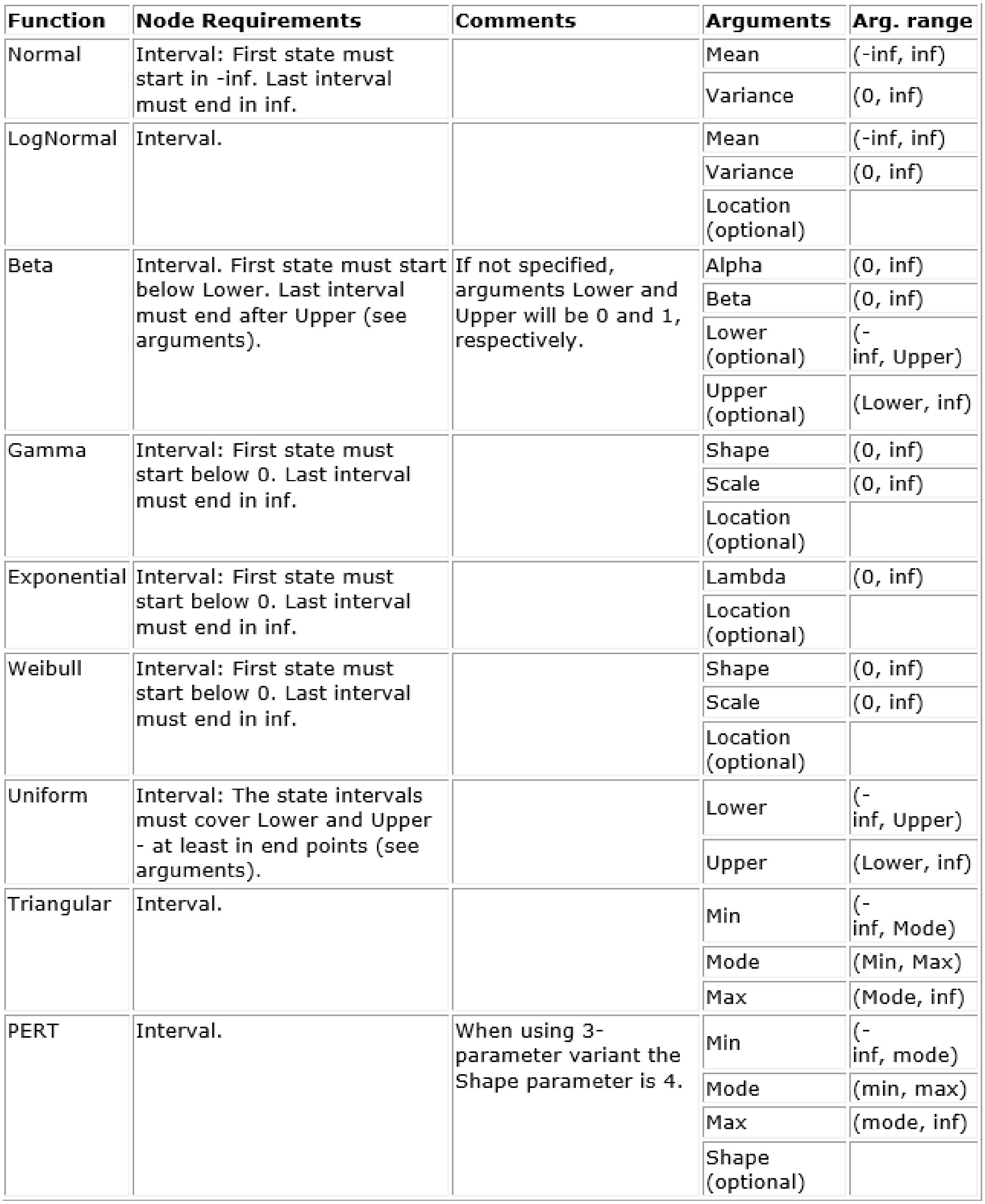
A number of discrete statistical distributions are available. See *Discrete Distributions* (page 83) for details.

- Binomial
- Poisson
- Negative Binomial
- Geometric
- Distribution
- Noisy OR

Statistical Distributions

Continuous Distributions

The following continuous distribution functions are available for interval nodes only.



Function	Node Requirements	Comments	Arguments	Arg. range
Normal	Interval: First state must start in -inf. Last interval must end in inf.		Mean	(-inf, inf)
			Variance	(0, inf)
LogNormal	Interval.		Mean	(-inf, inf)
			Variance	(0, inf)
			Location (optional)	
Beta	Interval. First state must start below Lower. Last interval must end after Upper (see arguments).	If not specified, arguments Lower and Upper will be 0 and 1, respectively.	Alpha	(0, inf)
			Beta	(0, inf)
			Lower (optional)	(-inf, Upper)
			Upper (optional)	(Lower, inf)
Gamma	Interval: First state must start below 0. Last interval must end in inf.		Shape	(0, inf)
			Scale	(0, inf)
			Location (optional)	
Exponential	Interval: First state must start below 0. Last interval must end in inf.		Lambda	(0, inf)
			Location (optional)	
Weibull	Interval: First state must start below 0. Last interval must end in inf.		Shape	(0, inf)
			Scale	(0, inf)
			Location (optional)	
Uniform	Interval: The state intervals must cover Lower and Upper - at least in end points (see arguments).		Lower	(-inf, Upper)
			Upper	(Lower, inf)
Triangular	Interval.		Min	(-inf, Mode)
			Mode	(Min, Max)
			Max	(Mode, inf)
PERT	Interval.	When using 3-parameter variant the Shape parameter is 4.	Min	(-inf, mode)
			Mode	(min, max)
			Max	(mode, inf)
			Shape (optional)	

Example:

- Normal(C1, C2) - specifies the conditional probabilities for an interval node from a Normal distribution with mean given by the value of the numeric parent node C1 and variance given by another numeric parent node C2.

Truncation operator:

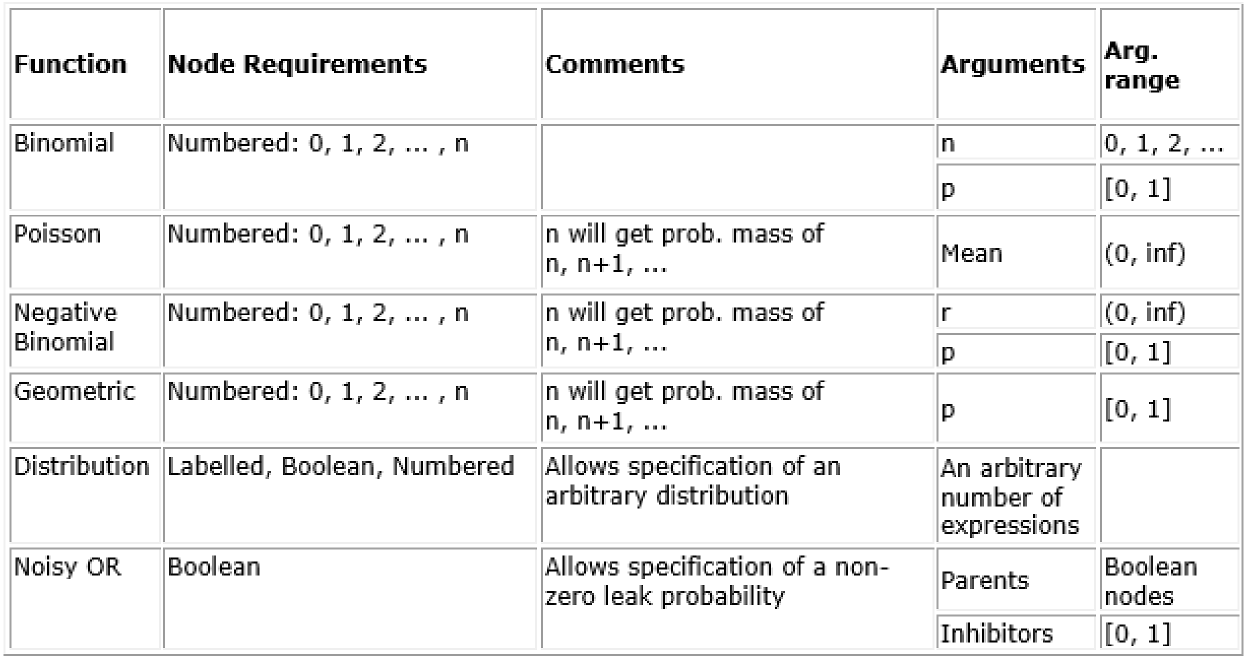
In addition a truncation operator can be applied to a continuous statistical distribution in order to form a truncated distribution. The operator takes either two or three arguments. When three arguments are specified, the first and third arguments must be numeric expressions denoting, respectively, the left and right truncation points, while the second argument must denote the distribution to be truncated. Either the first or the third argument can be omitted. Omitting the first argument results in a right-truncated distribution, and omitting the third argument results in a left-truncated distribution.

Example:

- truncate (-4, Normal (0, 1), 4) - specification of a truncated normal distribution. The distribution is truncated at the left at -4 and at the right at 4.
- truncate (4, Normal (0, 1)) - A left truncated (at -4) normal distribution.

Discrete Distributions

A variety of discrete distribution functions can be specified. There are four standard statistical distribution functions, which all must be specified for numeric nodes. The special function called 'Distribution' allows one to specify arbitrary distribution functions, where an expression must be specified for each possible outcome of the variable in question.



Function	Node Requirements	Comments	Arguments	Arg. range
Binomial	Numbered: 0, 1, 2, ... , n		n	0, 1, 2, ...
			p	[0, 1]
Poisson	Numbered: 0, 1, 2, ... , n	n will get prob. mass of n, n+1, ...	Mean	(0, inf)
Negative Binomial	Numbered: 0, 1, 2, ... , n	n will get prob. mass of n, n+1, ...	r	(0, inf)
			p	[0, 1]
Geometric	Numbered: 0, 1, 2, ... , n	n will get prob. mass of n, n+1, ...	p	[0, 1]
Distribution	Labelled, Boolean, Numbered	Allows specification of an arbitrary distribution	An arbitrary number of expressions	
Noisy OR	Boolean	Allows specification of a non-zero leak probability	Parents	Boolean nodes
			Inhibitors	[0, 1]

Examples:

- Binomial(4, 0.2)
- NegativeBinomial(sqrt(10), 0.4)
- NoisyOR(C1, 0.1, C2, 0.05, true, 1) - specifies that a Boolean effect node is 'true' with probability 1 - 0.1 if the Boolean cause (parent) node C1 is 'true' and C2 is 'false', 1 - 0.05 if the Boolean cause (parent) node C2 is 'true' and C1 is 'false', and 1 - 0.1*0.05 if both C1 and C2 are 'true'. Further, it specifies that the effect node is 'false' with probability 1 if both C1 and C2 are 'false' (i.e., a leak probability of zero has been specified in this example).

- Distribution(if(C1 = “State 1”, 3, 5), 2) - specifies the distribution for a binary node with a Labelled parent node C1.

Syntax for Expressions

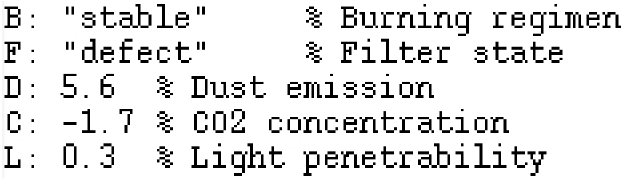
```
<Expression> ::= <Simple expression> <Comparison> <Simple expression> |

             <Simple expression>



<Simple expression> ::= <Simple expression> <Plus or minus> <Term> |
                    <Plus or minus> <Term> |
                    <Term>

<Term> ::= <Term> <Times or divide> <Exp factor> |
               <Exp factor>

<Exp factor> ::= <Factor> ^ <Exp factor> |
                                 <Factor>


<Factor> ::= <Unsigned number> |
                 <Node name> |
                 <String> |
                 false |
                 true |
                 (<Expression>) |
                 <Operator> (<Expression sequence>)


<Expression sequence> ::= <Empty> | <Expression> [, <Expression>]*

<Comparison> ::= == | = | != | <> | < | <= | > | >=

<Plus or minus> ::= + | -

<Times or divide> ::= * | /

<Operator> ::= truncate | Normal | LogNormal | Beta | Gamma | Exponential | Weibull |␣
↪Uniform | Triangular | PERT
       Binomial | Poisson | NegativeBinomial | Geometric | Distribution |
       NoisyOR | min | max | log | log2 | log10 | exp |
       sin | cos | tan | sinh | cosh | tanh |
       sqrt | abs | floor | ceil | mod |
       if | and | or | not
```

3.2.10 The Case and Data File Formats

The HUGIN Graphical User Interface supports case and data files to save and load cases and data. A case file consists of a single case, i.e., a set of evidence on a subset of the nodes in a network, whereas a data file consists of a set of cases. The format of a case file is different from the format of a data file.

The Case File Format

The HUGIN Graphical User Interface supports saving and loading of case files. A case file contains a description of a single evidence scenario such as this



```
B: "stable"     % Burning regimen
F: "defect"     % Filter state
D: 5.6  % Dust emission
C: -1.7 % CO2 concentration
L: 0.3  % Light penetrability
```

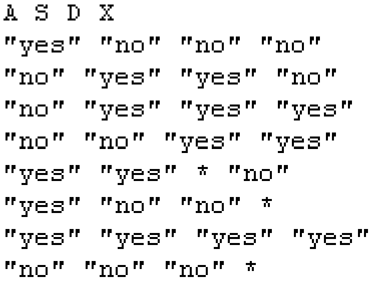
Figure 1: A sample case file.

Case files are useful for storing a set of evidence to be propagated in the network.

A more elaborate description case files can be found *here* (page 388).

The Data File Formats

The HUGIN Graphical User Interface supports saving and loading of data files. A data file contains a set of cases as this



```
A S D X
"yes" "no" "no" "no"
"no" "yes" "yes" "no"
"no" "yes" "yes" "yes"
"no" "no" "yes" "yes"
"yes" "yes" * "no"
"yes" "no" "no" *
"yes" "yes" "yes" "yes"
"no" "no" "no" *
```

Figure 2: A sample data file.

Data files are useful for storing a set of cases to be propagated in the network or more commonly to be used for learning.

A more elaborate description data files can be found *here* (page 389).

3.2.11 Case Generator Tutorial

To generate (simulated) cases based on the current conditional probability distributions (CPTs) select the Generate Cases item in the File menu as shown in figure 1.

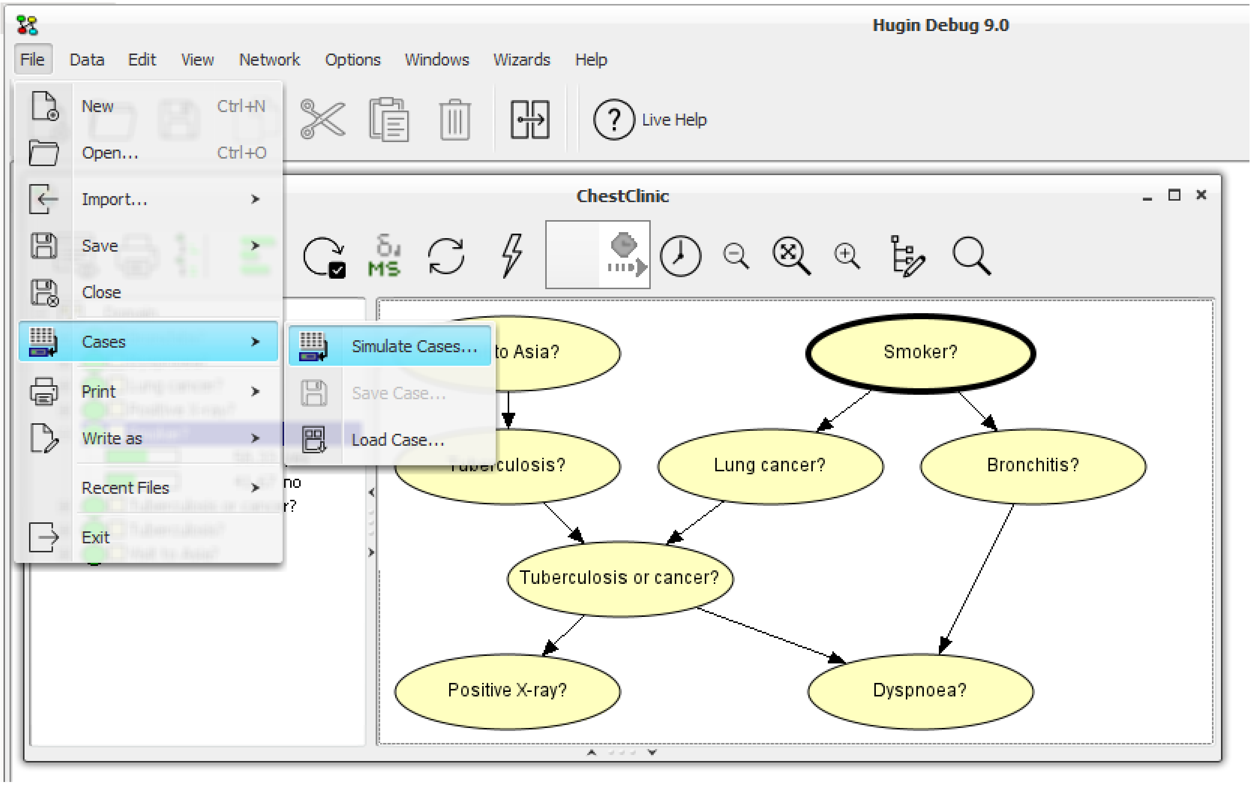
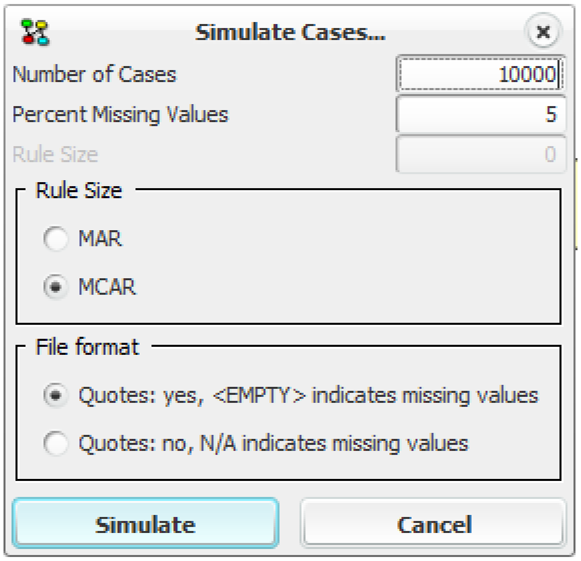


Figure 1: The Case Generator Button.

By default, the HUGIN Graphical User Interface chooses to create 10,000 cases with 5 percent missing values. The user can change these values. To generate the cases one of the algorithm: MCAR or MAR must be selected. Figure 2 shows the Case Generator Window.



Number of Cases 10000

Percent Missing Values 5

Rule Size 0

Rule Size

○ MAR

● MCAR

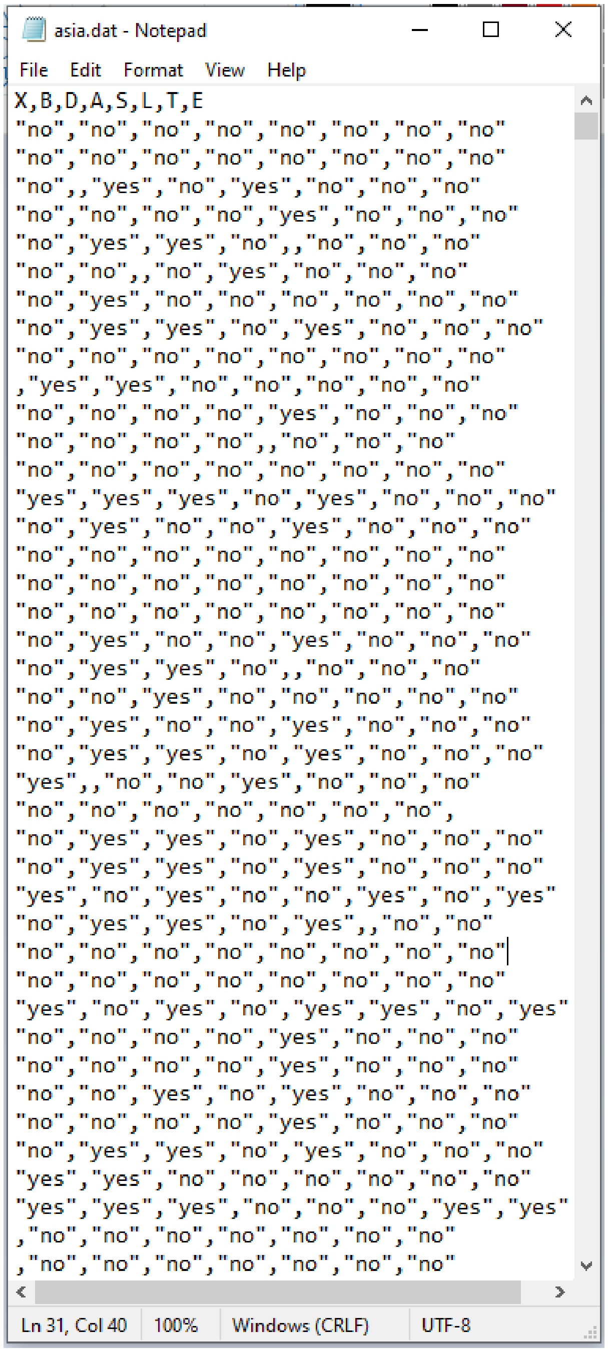
File format

● Quotes: yes, <EMPTY> indicates missing values

○ Quotes: no, N/A indicates missing values

Figure 2: The Case Generator Window.

Now, the HUGIN Graphical User Interface will generate a set of cases and store these in a plain text file. Figure 3 shows the content of such a file where the cases are generated based on the `Chest Clinic` network.



```
X,B,D,A,S,L,T,E
"no","no","no","no","no","no","no","no"
"no","no","no","no","no","no","no","no"
"no",,"yes","no","yes","no","no","no"
"no","no","no","no","yes","no","no","no"
"no","yes","yes","no",,"no","no","no"
"no","no",,"no","yes","no","no","no"
"no","yes","no","no","no","no","no","no"
"no","yes","yes","no","yes","no","no","no"
"no","no","no","no","no","no","no","no"
,"yes","yes","no","no","no","no","no"
"no","no","no","no","yes","no","no","no"
"no","no","no","no",,"no","no","no"
"no","no","no","no","no","no","no","no"
"yes","yes","yes","no","yes","no","no","no"
"no","yes","no","no","yes","no","no","no"
"no","no","no","no","no","no","no","no"
"no","no","no","no","no","no","no","no"
"no","no","no","no","no","no","no","no"
"no","yes","no","no","yes","no","no","no"
"no","yes","yes","no",,"no","no","no"
"no","no","yes","no","no","no","no","no"
"no","yes","no","no","yes","no","no","no"
"no","yes","yes","no","yes","no","no","no"
"yes",,"no","no","yes","no","no","no"
"no","no","no","no","no","no","no",
"no","yes","yes","no","yes","no","no","no"
"no","yes","yes","no","yes","no","no","no"
"yes","no","yes","no","no","yes","no","yes"
"no","yes","yes","no","yes",,"no","no"
"no","no","no","no","no","no","no","no"
"no","no","no","no","no","no","no","no"
"yes","no","yes","no","yes","yes","no","yes"
"no","no","no","no","yes","no","no","no"
"no","no","no","no","yes","no","no","no"
"no","no","yes","no","yes","no","no","no"
"no","no","no","no","yes","no","no","no"
"no","yes","yes","no","yes","no","no","no"
"yes","yes","no","no","no","no","no","no"
"yes","yes","yes","no","no","no","yes","yes"
,"no","no","no","no","no","no","no"
,"no","no","no","no","no","no","no"
```

Figure 3: The Content of a Case File.

3.2.12 The Net Language

The HUGIN Development Environment uses a special-purpose language, called the net language, for description of networks and LIMID models. This language allows the user to create complete descriptions of Bayesian networks and LIMID models, containing specifications of the structure of the network model, the conditional probability, the policy, and the utility functions.

This chapter describes the third revision of the net language. This revision is substantially different from the first revision, and somewhat different from the second revision. The reason is that the first revision of the language used a fixed format (i.e., the semantics of the different elements were determined by their position within the description). This implied that it was impossible to extend this language in such a way that descriptions in the old language retained their meaning in the new language. The second revision was designed with that goal in mind. This third revision contains a minor modification of the second revision, enabling specification of object-oriented networks (i.e., a net file now can contain *instance nodes* (page 224), and means for specifying *interface nodes* (page 229)).

Nodes

The basic element of a network model is the *node*. In ordinary Bayesian networks, a node represents a random variable (discrete or continuous). In a LIMID model, a node may also represent a decision, controlled by the *decision* maker, or a utility function, which is used to assign preferences to different configurations of variables. In object-oriented networks, there is an additional *category of nodes* (page 209), namely *instance nodes*, representing *instances* of other networks; that is, an instance node represents a sub-network in the network in which it appears.

Example 1

The following node description is taken from the "Chest Clinic" example *(Lauritzen & Spiegelhalter 1988)* (page 541)

```
node T
{
  states = ("yes" "no");
  label = "Has tuberculosis?";
  position = (25 275);
}
```

- This describes a binary random variable named *T*, with states labeled "yes" and "no". The description also gives the label and position, which are used by the HUGIN Graphical User Interface.

A node description is introduced by one of the keywords: [<prefix>] **node**, **decision**, **utility**, or **instance** where the optional prefix on **node** is either **discrete** or **continuous** (omitting the prefix causes discrete to be used as default). The keywords are followed by a name that must be unique within the model. For instance nodes further follows the name of the network (or class) of which it is an instance, and the bindings of the input nodes of the instance node. To be precise, the syntax of is

```
instance <name> : <class> ({<input> = <node>}*)
```

where <class> is the name of the network of which <name> is an instance, and where {<input> = <node>}* denotes a comma separated list of zero or more elements of the kind <input> = <node>, where <input> is the name of an input node in <class> and <node> is the name of a node appearing in the network or an output node of an instance of a network included in the network. Note that <input> and <node> must be type consistent; i.e., be of the same category, kind, and subtype, and have identical state labels/values if they are discrete.

Example 2

The following specifies an instance node with name Person_1. This node represents an instance of a network named Person, and the nodes named I and E have been bound to the input nodes named Income and Education, respectively, of the instance of Person used in our network.

```
instance Person_1 : Person (Income = I, Education = E)
{
    label = "Bill Yates";
    position = (101 156);
}
```

- The description also gives the label and position, which are used by the HUGIN Graphical User Interface.

Then, for all kinds of nodes, follows a sequence of name/value pairs of the form

```
<name> = <value>;
```

enclosed in braces.

The example shows the field names currently defined in the net language for nodes: *states*, *label*, and *position*. All of these fields are optional; if any field is absent, a default value is supplied.

- *states* specifies the states of the node (here, the decisions of a decision node are also referred to as states); the states are indicated by a list of strings. The list must be non-empty. The strings in the list comprise the labels of the individual states; the labels need not be unique (can even be empty strings) for the node. Only the length of the list (i.e., the number of states) is relevant to the HUGIN Decision Engine; the state labels are, however, used by the companion system, the HUGIN Graphical User Interface. The default value is a list of length one, containing an empty string (i.e., the node will have one state). The *states* field is not allowed for utility and continuous nodes.
- *label* is a string that is used by the HUGIN Graphical User Interface when displaying the nodes. The label is not used by the inference engine. The default value is the empty string.
- *position* is a list of integers (the list must have length two). It indicates the position within the graphical display of the network by the HUGIN Graphical User Interface. The position is not used by the inference engine. The default position is at (0,0).

Apart from these fields, you can specify your own fields for nodes. These can be used for a specific application needing some extra information about the nodes.

Example 3

In this situation the T node has being assigned the application specific field *MY_APPL_my_field*.:

```
node T
{
    states = ("yes" "no");
    label = "Has tuberculosis?";
    position = (25 275);
    MY_APPL_my_field = "1000";
}
```

The value of such application specific fields can only be text strings encapsulated in quote characters (”) (see section *“Lexical Matters”* (page 96) for precise definition of text strings).

It would be regarded as good style to start field name with an application specific prefix to avoid confusion (in example 3 the *MY_APPL* prefix).

Example 4

In the HUGIN Graphical User Interface some extra fields are used to save descriptions of both nodes and their states. These are the fields prefixed with *HR*.:

```
node T
{
    states = ("yes" "no");
    label = "Has tuberculosis?";
    position = (25 275);
    HR_State_0 = "Yes, the patient HAS tuberculosis.";
    HR_State_1 = "No, the patient has NOT tuberculosis.";
    HR_Desc = "Represents the fact that the patient has\
tuberculosis or not.";
}
```

The Structure of the Model

The structure (i.e., the edges of the underlying graph) is specified indirectly. We have two kinds of edges: *directed* and *undirected* edges.

Example 5

This is a typical specification of directed edges:

```
potential ( A | B C ) { }
```

This specifies that node *A* has two parents: *B* and *C*. That is, there is a directed edge from *B* to *A*, and there is a directed edge from *C* to *A*.

The model may also contain undirected edges. Such a model is called a *chain graph* model.

Example 6

```
potential ( A B | C D ) { }
```

This specifies that there is an undirected edge between *A* and *B*. Moreover, as usual, it specifies that both *A* and *B* have *C* and *D* as parents.

If there are no parents, the vertical bar may be omitted.

A maximal set of nodes, connected by undirected edges, is called a *chain graph component*.

Not all graps are permitted. The following restrictions are imposed on the structure of the network.

The graph may not contain any (directed) cycles.

Example 7

The following specification is not allowed, because of the cycle $A \rightarrow B \rightarrow C \rightarrow A$.

```
potential ( B | A ) { }
potential ( C | B ) { }
potential ( A | C ) { }
```

However, the following specification is legal.

```
potential ( B | A ) { }
potential ( C | B ) { }
potential ( C | A ) { }
```

Example 8

The following specification is not allowed either, since there is a cycle $A \rightarrow B \rightarrow C\ A$ (the edge between A and C counts as “bidirectional”).

```
potential ( B | A ) { }
potential ( C | B ) { }
potential ( A C ) { }
```

However, the following specification is legal.

```
potential ( A | B ) { }
potential ( C | B ) { }
potential ( A C ) { }
```

Continuous chance nodes are not allowed in LIMID models, i.e., there cannot be continuous nodes in a net also containing utility or decision nodes.

Utility nodes may not have any children in the graph. This implies that utility nodes may only appear to the left of the vertical bar (never to the right).

Undirected edges can only appear between discrete chance nodes.

Continuous nodes can only have continuous nodes as children.

If a decision node appears to the left of the vertical bar, it must appear alone. In this case, so-called *informational* links are specified; such links specify which variables are known when the decision is to be made.

Example 9

Assume we want to specify a LIMID with two decisions, *D1* and *D2*, and with three discrete chance variables, *A*, *B*, and *C*. First, *A* is observed; then, decision *D1* is made; then, *B* is observed; finally, decision *D2* is made. This sequence of events can be specified as follows:

```
potential ( D1 | A) { }
potential ( D2 | D1 B ) { }
```

Finally, no node may be referenced in any potential-specification before it has been declared by a node-, decision-, or a utility-specification.

Potentials

We also need to specify the quantitative part of the model. This part consists of conditional probability functions for random variables, policies for decision nodes and the values a utility function may assume. Thus, we distinguish between *discrete probability*, *continuous probability*, and *utility potentials*.

All types of potentials are different in the numerical specification between the braces of the potential-specification.

Example 10

The following description is taken from the "Chest Clinic" example and specifies the conditional probability table of the discrete variable T.

```
potential ( T | A )
{
    data = (( 0.05 0.95 )          %  A=yes
            ( 0.01 0.99 ));        %  A=no
}
```

This specifies that the probability of tuberculosis given a trip to Asia is 5 %, whereas it is only 1 % if the subject has not been to Asia. The *data* field may also be specified as an unstructured list of numbers.

```
potential ( T | A )
{
    data = ( 0.05 0.95           %  A=yes
             0.01 0.99 );        %  A=no
}
```

As the example shows, the numerical data is specified through the *data* field of a potential-specification. This data has the form of a list of real numbers. The structure of the list must either correspond to that of a multi-dimensional table with node list comprised of the parent nodes followed by the child nodes, or it must be a flat list with no structure at all. The "layout" of the *data* list is *row-major* (see section *"Row-major Representation"* (page 96)).

Example 11

```
potential ( D E F | A B C ) { }
```

The *data* field of this potential-specification corresponds to a multi-dimensional table with dimension list <*A*, *B*, *C*, *D*, *E*, *F*>.

The *data* field of a utility potential has only the dimension of the nodes on the right side of the vertical bar.

Example 12

The following description is taken from the "Oil Wildcatter" example and shows a utility potential. *Drillpay* is a utility node while *Oil* is a discrete chance node with three states and *Drill* is a decision node with two states.

```
potential (Drillpay | Oil Drill)
{
    data = (( -70 0 )         %  dr
            ( 50 0 )          %  wt
```

(continues on next page)

(continued from previous page)

```
            ( 200 0 ));       %  sk
}
```

The *data* field of this potential-specification corresponds to a multi-dimensional table with dimension list < *Oil*, *Drill*>.

The table in the *data* field of a continuous probability potential has the dimensions of the discrete chance nodes to the right of the vertical bar. All the discrete chance nodes must be listed first on the right side of the vertical bar (then follows the continuous nodes). However, the items in the multi-dimensional table are no longer values but instead *continuous distribution functions*. Currently, only Gauss normal distribution can be used. A normal distribution can be specified by its mean and variance. In the following example, a continuous probability potential is described.

Example 13

Suppose *A* is a continuous node with parents *B* and *C* which are both discrete. Also, both *B* and *C* have two states: *B* has states *b1* and *b2* while *C* has states *c1* and *c2*.

```
potential (A | B C)
{
    data = (( normal ( 0, 2 )       %  b1  c1
              normal ( 3, 2 ) )     %  b1  c2
            ( normal ( 1, 2 )       %  b2  c1
              normal ( 2, 2 ) ));   %  b2  c2
}
```

The *data* field of this potential-specification is a table with the dimension list < *B*, *C*>. Each entry contains a probability distribution for the continuous node A.

All entries in the above example contains a Gauss normal distribution (the only continuous distribution currently available). A normal distribution is specified with the keyword **normal** followed by a list of two parameters. The first parameter is the mean and the second is the variance of the normal distribution.

Example 14

In this example, suppose *A* is a continuous node with one discrete parent *B* and one continuous parent *C*. *B* has two states *b1* and *b2* and *C* has a normal distribution.

```
potential (A | B C)
{
    data = ( normal ( 1 + C, 2 )              %  b1
             normal ( 1 + 2 * C, 2 ) );       %  b2
}
```

The *data* field of this potential-specification is a table with the dimension list <*B*> (*B* is the only discrete parent which is then listed first on the right side of the vertical bar). Each entry again contains a continuous distribution function for *A*. The influence of *C* on A now comes from the use of *C* in an expression specifying the mean parameter of the normal distributions.

Only the mean parameter of a normal distribution can be specified as a an expression. The variance parameter must be a numeric constant. The operators allowed in the expression are +, -, and * (addition, subtraction, and multiplication).

Since a decision node has no function assigned, it cannot have a data field. Thus, the decision potential specification does not really specify a potential but is rather a trick for specification of informational links.

If the *data* field is omitted from a potential-specification, a list of ones is supplied for discrete probability potentials, whereas a list of zeros is supplied for utility potentials. For a continuous probability potential, a list of normal distributions with both mean and variance set to 0 is supplied.

The values of the *data* field of discrete probability potentials may only contain non-negative numbers. In the specification of a normal distribution for a continuous probability potential, only non-negative numbers are allowed for the variance parameter. There is no such restriction on the values of utility potentials or the mean parameter of a normal distribution.

Global Information

Information pertaining to the Bayesian network or LIMID model as a whole can be specified at the beginning of the description, initiated by the keyword **class** (in the second revision of the net language, this keyword was **net**) - see Example 15. Notice that, except for the first line (i.e., “class <net name>”), the entire specification must be enclosed in curly braces (this syntax differs from the syntax used in the second revision).

Example 15

```
class test
{
    inputs = ();
    outputs = ();
    node_size = (80 40);
    ...
}
```

This specifies lists of names of nodes of the network that serves as input nodes and output nodes, respectively. It also specifies that nodes are drawn in the HUGIN Graphical User Interface with a width of 80 pixels and a height of 40 pixels.

Currently, only the *inputs*, *outputs*, and *node_size* fields have been defined for **net**-specifications. However, as with nodes, you can add all the additional fields you want.

Example 16

The newest version of the HUGIN Graphical User Interface uses a series of application specific fields. Some of them are shown here:

```
class
{
    inputs = ();
    outputs = ();
    node_size = (80 40);
    HR_Grid_X = "10";
    HR_Grid_Y = "10";
    HR_Grid_GridSnap = "1";
    HR_Grid_GridShow = "0";
    HR_Font_Name = "Arial";
    HR_Font_Size = "-12";
    HR_Font_Weight = "400";
    HR_Font_Italic = "0";
```

(continues on next page)

(continued from previous page)

```
    HR_Propagate_Auto = "0";
    ...
}
```

The HUGIN Graphical User Interface uses the prefix HR on all fields.

Lexical Matters

A name has the same structure as an identifier in the C programming language. This means that a name is a non-empty sequence of letters and digits, beginning with a letter. In this context, the underscore character (_) is considered a letter. The case of letters is significant. The sequence of letters and digits forming a name extends as far as possible; it is terminated by the first non-letter/digit character (for example, braces or whitespace).

A string is a sequence of characters not containing a quote character (“) or a newline character; its start and end are indicated by quote characters.

A number is comprised of an optional sign, followed by a sequence of digits, possibly containing a decimal point character, and an optional exponent field containing an ‘E’ or ‘e’ followed by a possibly signed integer.

Comments can be placed in a net description anywhere (except within a name, a number, or other multi-character lexical elements). It is considered equivalent to whitespace. A comment is introduced by a percent character (%) and extends to the end of the line.

Row-Major Representation

This section describes the row-major “layout” of a table. If you do not have interest in this particular subject, you can just ignore it.

To find a value corresponding to a specific configuration in the row-major representation of a table, we index the values from 0 to s-1 where s is the number of values in the list. Suppose that the corresponding list of nodes is (A_1, A_2,…, A_n) and that node A_i has s_i states indexed from 0 to s_i-1. Then,

$$s = \prod_{i=1}^{n} s_i$$

What we want is the index x of a configuration (a_1, a_2,…, a_n). Now, suppose that the state index of a_i is j_i $(0 \leq ji \leq si - 1)$. Then, x can be computed as

$$x = \sum_{i=1}^{n} a_i j_i$$

where

$$a_i = \begin{cases} a_i j_i & \text{if } i < n, \\ 1 & \text{if } i = n. \end{cases}$$

3.2.13 Structure Learning Tutorial

The HUGIN Graphical User Interface provides the user with powerful structure learning capabilities (i.e., learning the structure of a Bayesian-network model from data (a set of cases)). Structure learning can be performed via the *Learning Wizard* (page 267) (which allows data to be read from data files, to be preprocessed, etc) or by activating one of the structure learning algorithms directly. In this tutorial, the structure learning algorithms are activated directly.

There are several algorithms to choose from, however this example will only look at the *NPC algortihm* (page 246) and the *PC algorithm* (page 246). In this tutorial we shall assume that you already know how these algorithms work. Refer to the "Help" button on the bottom left of the window for any details on the algorithms.

Consider the data file `asia.dat` sampled from the `Chest Clinic` network shown in Figure 1.

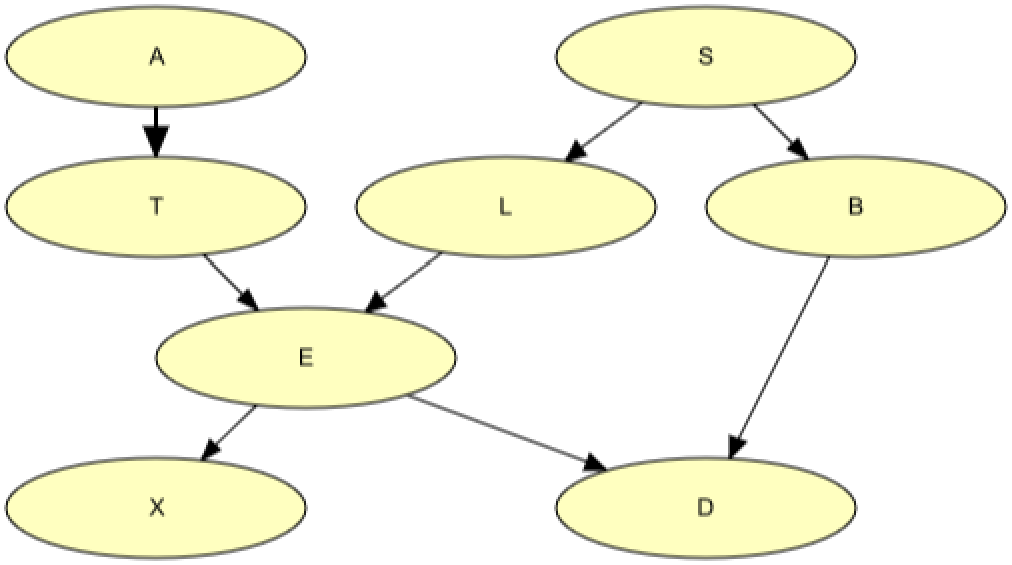
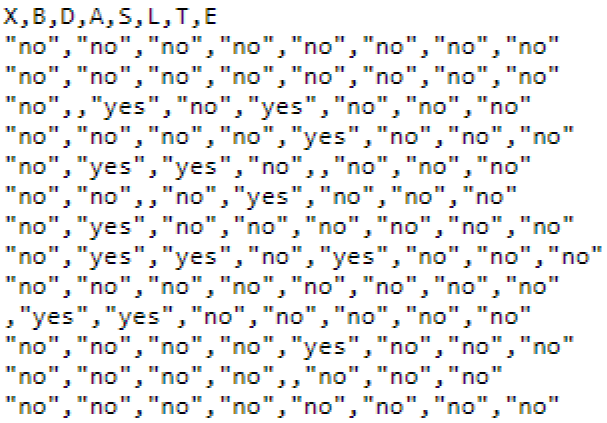


Figure 1: The structure from which the data were sampled.

Figure 2 shows the first few lines of the data file, which consists of 10000 cases altogether.



```
X,B,D,A,S,L,T,E
"no","no","no","no","no","no","no","no"
"no","no","no","no","no","no","no","no"
"no",,"yes","no","yes","no","no","no"
"no","no","no","no","yes","no","no","no"
"no","yes","yes","no",,"no","no","no"
"no","no",,"no","yes","no","no","no"
"no","yes","no","no","no","no","no","no"
"no","yes","yes","no","yes","no","no","no"
"no","no","no","no","no","no","no","no"
,"yes","yes","no","no","no","no","no"
"no","no","no","no","yes","no","no","no"
"no","no","no","no",,"no","no","no"
"no","no","no","no","no","no","no","no"
```

Figure 2: First few lines of the data file.

The structure learning functionality is available under the “File” menu and through the structure learning button of the tool bar of the *Main Window* (page 186). The structure learning button is shown in Figure 3.

The structure learning functionality is available under the “Wizards” menu as shown in Figure 3. This wizard will take you through a series of steps required in order to use the feature.

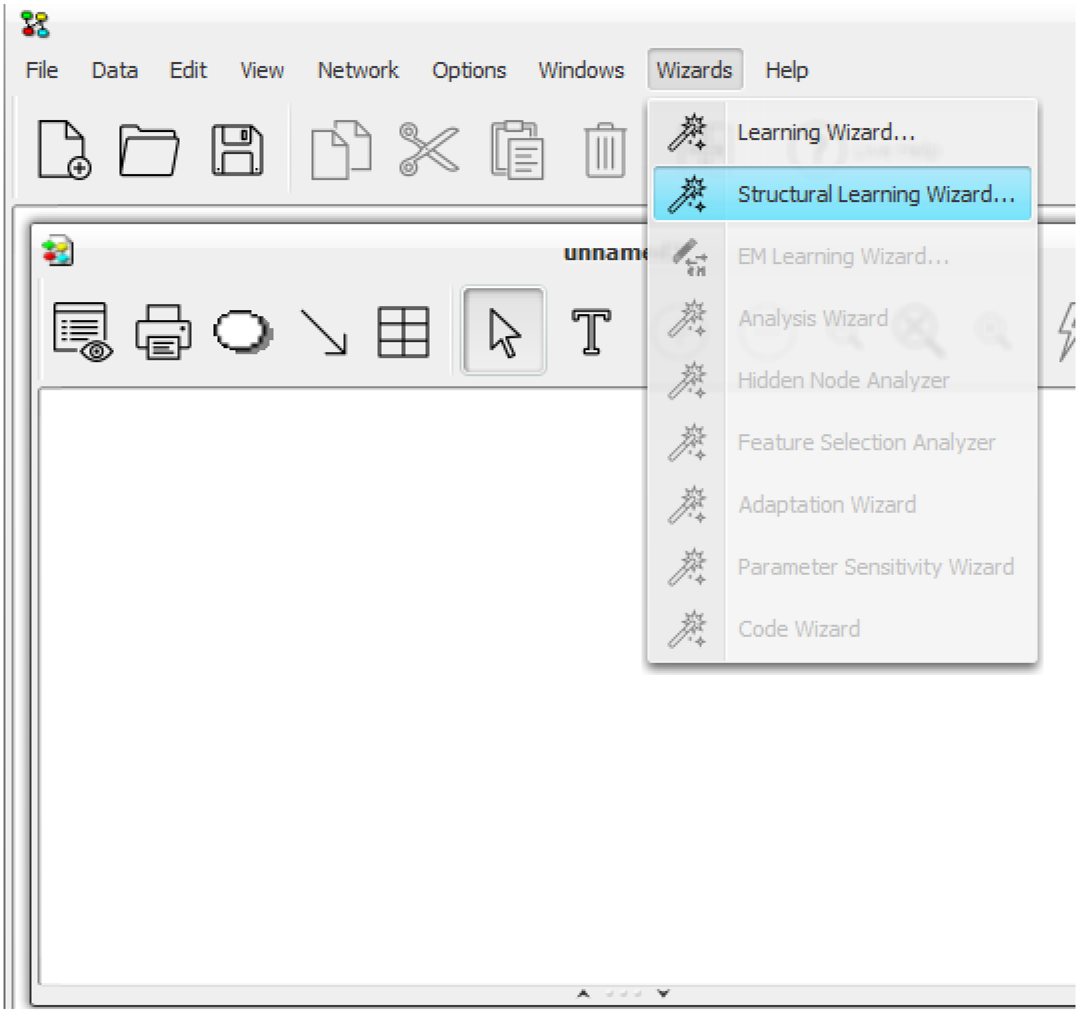


Figure 3: The option to activate the structure learning functionality.

When said option is chosen, the structure learning dialog appears (see Figure 4).

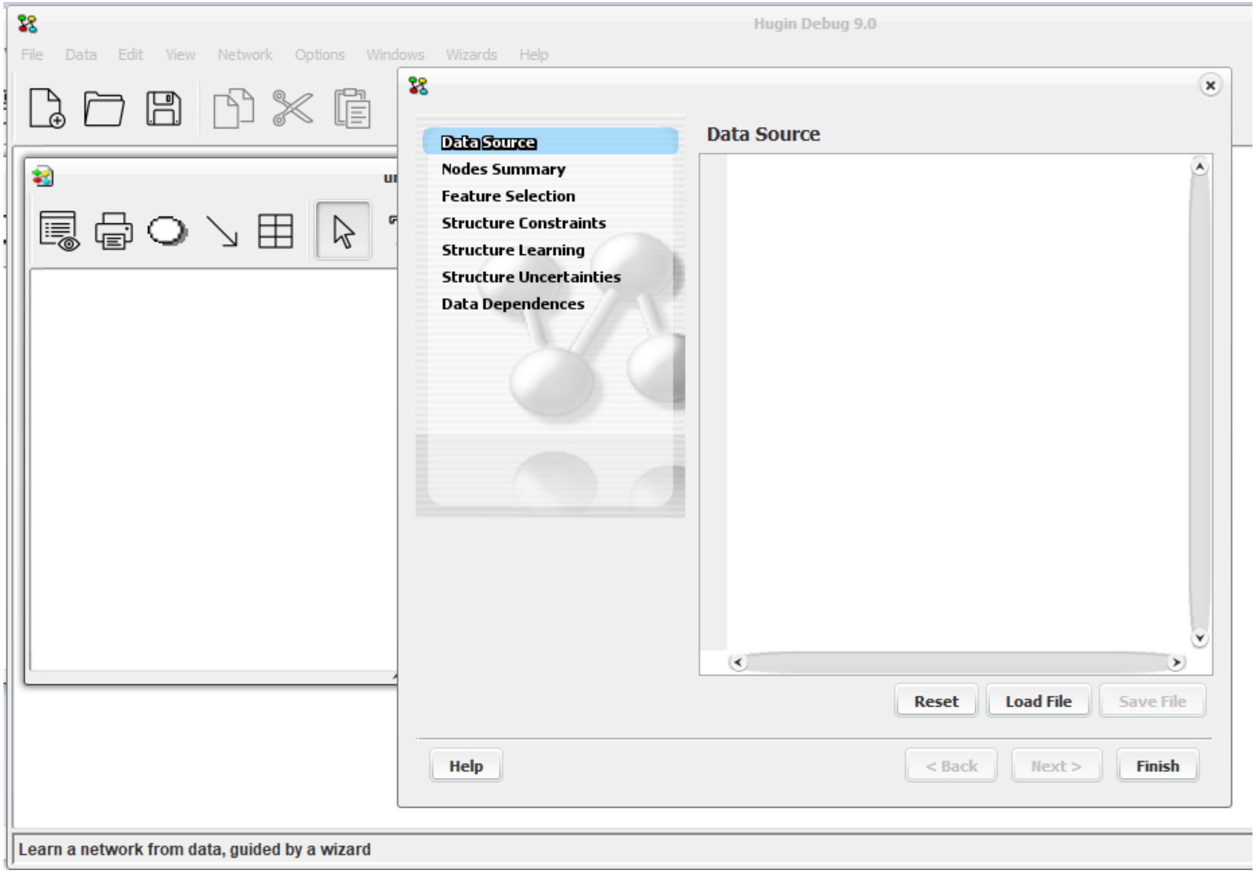
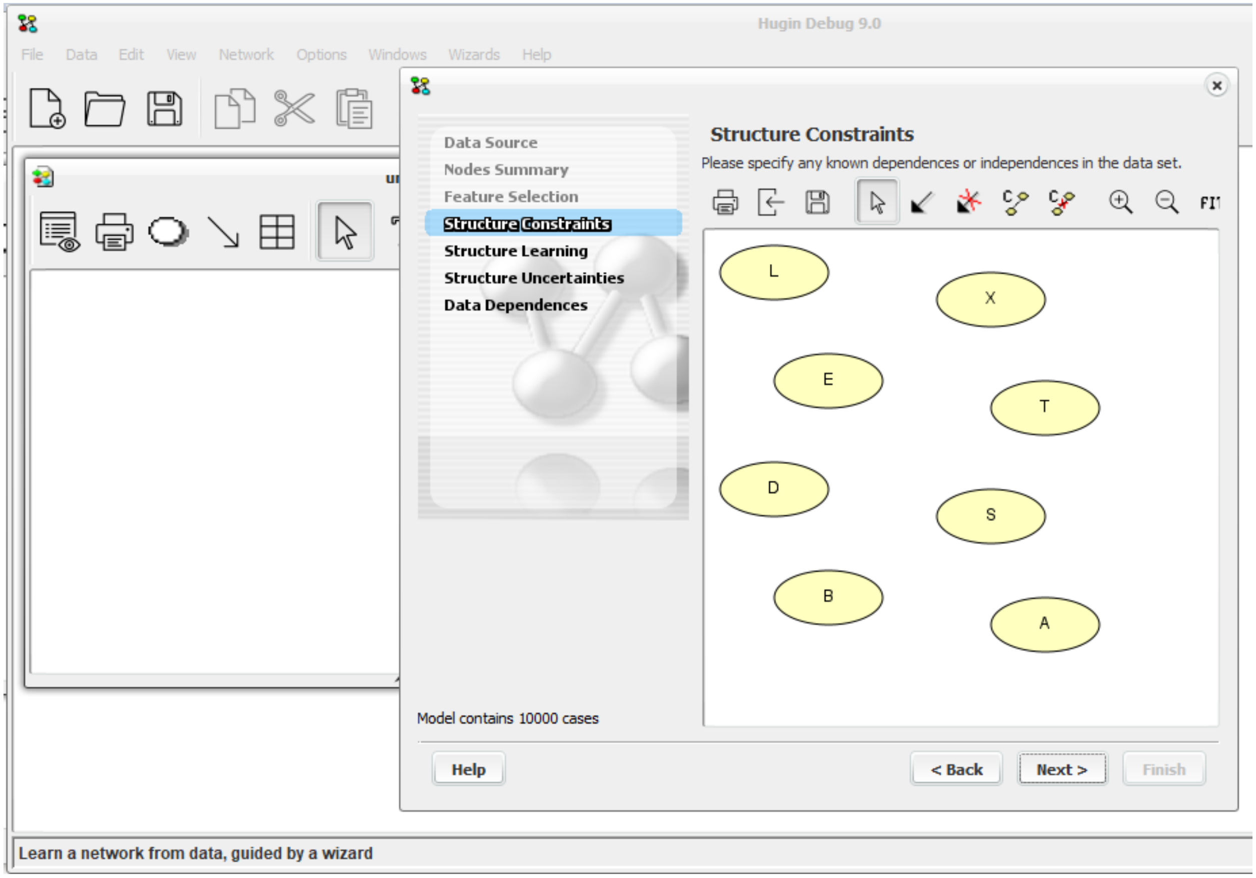


Figure 4: The structure learning dialog.

Press the “Load File” button and choose a file from which the structure is to be estimated. When a file has been selected the “Next” button gets enabled. When the “Next” button is clicked, the wizard shows you a node summary of your file and if you click “Next” again you can use the Feature Selection tool to narrow the set of nodes. For this example, we just click “Next” again without selecting any nodes. Next you need to specify any structure constraints, meaning any known dependencies or independencies in the data set, see Figure 5. Once again we just click “Next”.



Structure Constraints

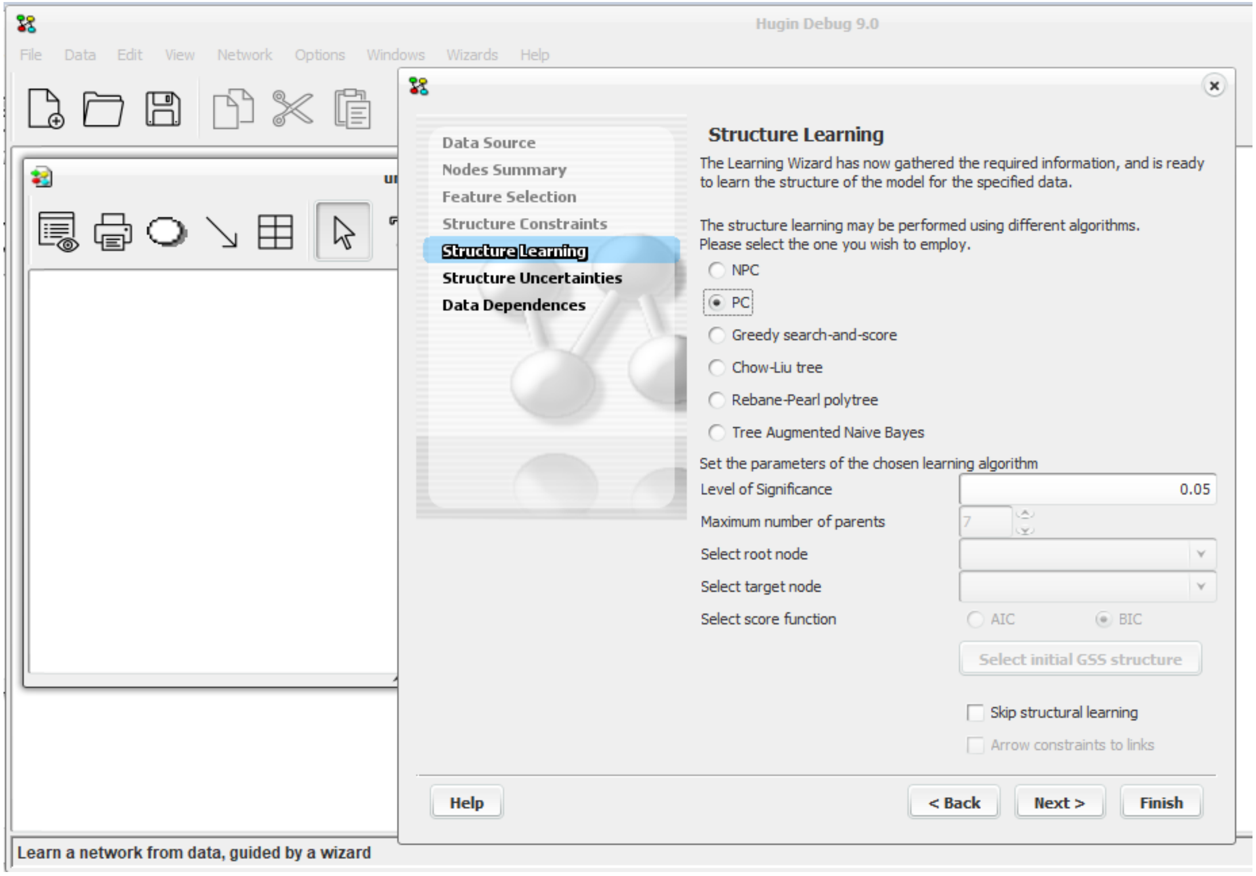
Please specify any known dependences or independences in the data set.

Model contains 10000 cases

Learn a network from data, guided by a wizard

Figure 5: The Structure Constraints dialog.

On the next page you need to specify an algorithm for the structure learning to perform.



Structure Learning

The Learning Wizard has now gathered the required information, and is ready to learn the structure of the model for the specified data.

The structure learning may be performed using different algorithms. Please select the one you wish to employ.

○ NPC
● PC
○ Greedy search-and-score
○ Chow-Liu tree
○ Rebane-Pearl polytree
○ Tree Augmented Naive Bayes

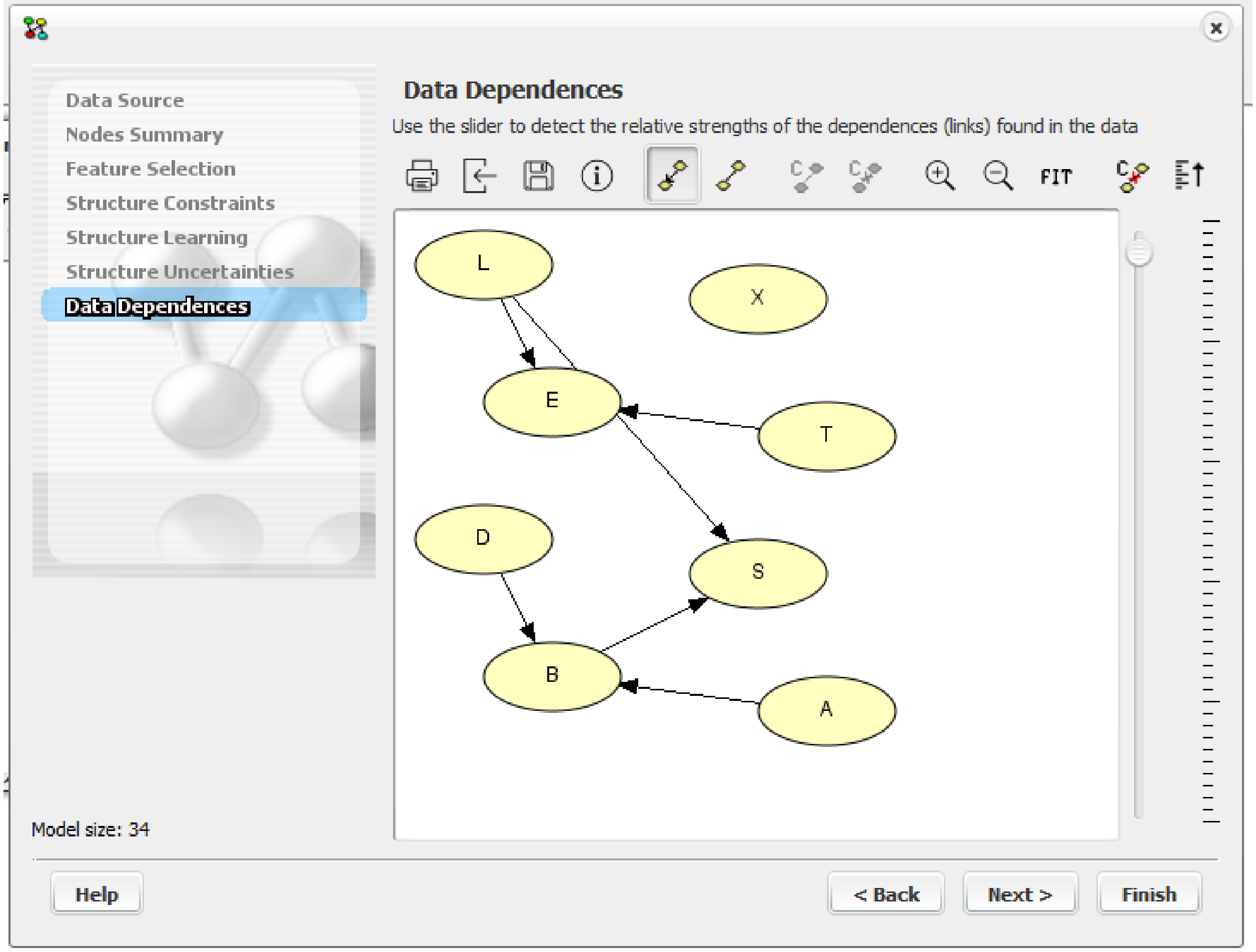
Set the parameters of the chosen learning algorithm

Level of Significance 0.05
Maximum number of parents 7
Select root node
Select target node
Select score function ○ AIC ● BIC

□ Skip structural learning
□ Arrow constraints to links

Figure 6: The Structure Learning dialog

First chose the PC algorithm and assert the *level of significance* (page 120) for the statistical independence tests performed during structure learning. Then click the “Next” button to the Data Dependencies dialog where you at last can detect the relative strengths of the dependencies (links) found in the data. You may use the slider to the right on the window.



Data Dependences

Use the slider to detect the relative strengths of the dependences (links) found in the data

Model size: 34

Figure 7: The Data Dependences dialog

Once finished press “Next” one last time to start the structure learning algorithm and load the network to the network pane. Here you should be able to manoeuvre the nodes to look like figure 8.

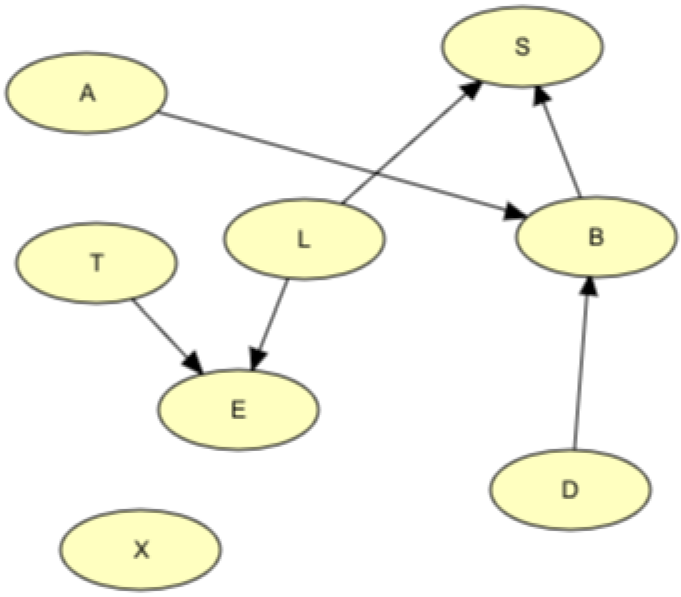


Figure 8: The structure learned by the PC algorithm

If the PC algorithm was selected, the result should appear as in Figure 8. Obviously, the structure of the original network has not been recalled perfectly. The problem is that variable E depends deterministically on variables T and L through a logical OR (E stands for Tuberculosis or Lung cancer). This means that (i) T and X are conditionally independent given E, (ii) L and X are conditionally independent given E, and (iii) E and X are conditionally independent given L and T. Thus, the PC algorithm concludes that there should be no links between T and X, between L and X, and between E and X. Obviously, this is wrong. The same reasoning leads the PC algorithm to leave D unconnected to T, L, and E. Also, as the PC algorithm directs links more or less randomly (respecting, of course, the conditional independence and dependence statements derived from data), the directions of some of the links are wrong.

If the NPC algorithm *NPC algortihm* (page 246) was selected and there are uncertain links or links for which the directionality could not be determined for sure, the user will be presented with an intuitive graphical interface for resolving these structural uncertainties (see the description of the NPC algorithm for details). Figure 9 shows the (intermediate) result of the NPC structure learning algorithm applied to the same data.

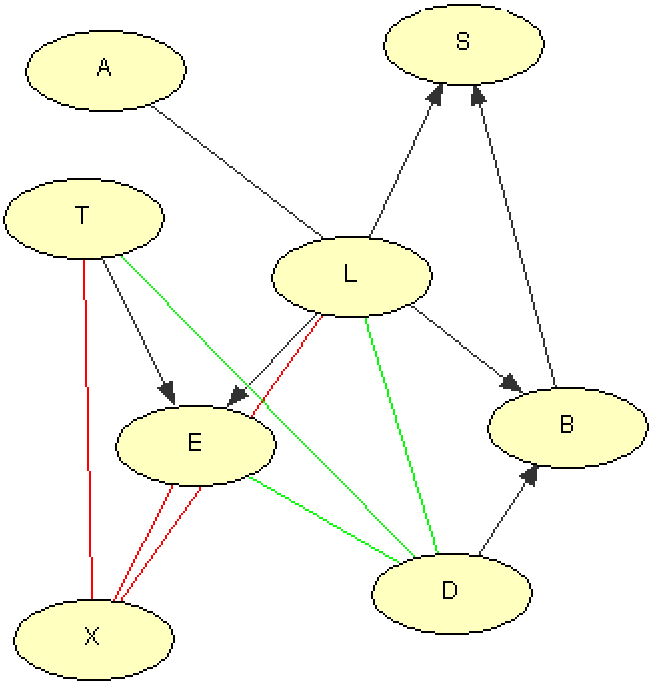


Figure 9: The intermediate structure learned by the NPC algorithm

Depending on the choices made by the user, many different final structures (including the original structure shown in Figure 1) can be generated from this intermediate structure.

The HUGIN Graphical User Interface implements a number of different algorithms for learning the structure of a Bayesian network from data, see Structure Learning for more information.

Once the structure of the Bayesian network has been generated, the conditional probability distributions of the network can be estimated from the data using the *EM-learning algorithm* (page 352), see the *EM tutorial* (page 105).

3.2.14 EM Learning Tutorial

> It often happens that many (or all) of the probability distributions of the variables in a Bayesian network are unknown, and that we want to learn these probabilities (parameters) from data (i.e., series of observations obtained by performing experiments, from the literature, or from other sources). An algorithm known as the *EM (Expectation-Maximization) algorithm* (page 352) is particularly useful for such parametric learning, and it is the algorithm used by HUGIN for learning from data. EM tries to find the model parameters (probability distribution) of the network from observed (but often not complete) data.

The *EM Algorithm* (page 352) is well-suited for parameter estimation in batch (i.e., estimation of the parameters of conditional probability tables from a set of cases) whereas the *Adaptation Algorithm* (page 244) is well-suited for sequential parameter update.

Experience Tables In EM Learning

Experience Tables (page 231) are used to specify prior experience on the parameters of the conditional probability distributions. Notice that EM parameter estimation is disabled for nodes without an experience table.

Example EM Learning

We use the example from the *Experience Tables* (page 231) to illustrate the use of EM learning. The Bayesian network is shown in Figure 1.

Go to `Chest Clinic` to learn more about the domain of this network.

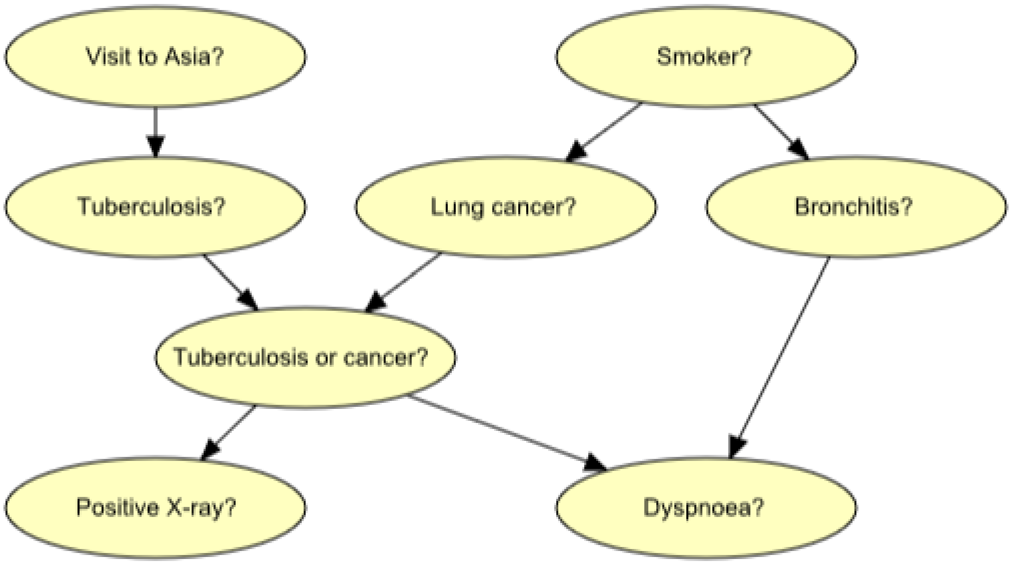
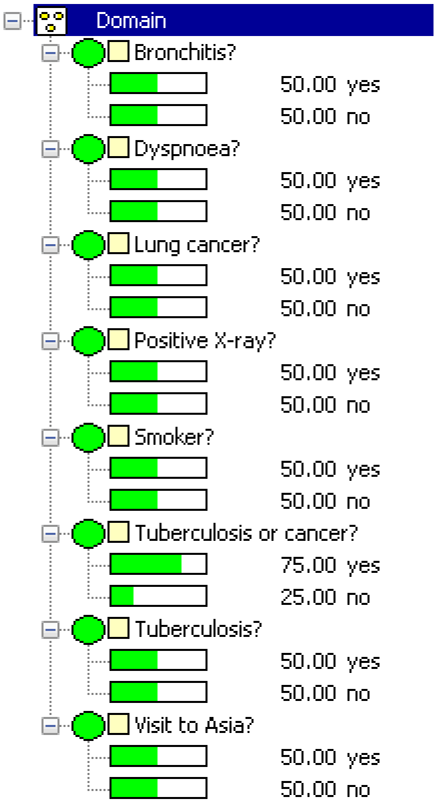


Figure 1: Bayesian-network representation of “Chest Clinic”.

To illustrate the use of EM learning, we assume that the structure of the network is known to be as shown in Figure 1. In addition, assume that the conditional probability distribution of “Tuberculosis or cancer” given “Has tuberculosis” and “Has lung cancer” is known to be a disjunction (i.e., the child is in state yes if and only if at least one of the parents is in state yes).

We do not assume any prior knowledge on the remaining set of distributions. Thus, we set all entries of the conditional probability distributions to one except for the “Tuberculosis or cancer” node. This will turn all the probability distributions for all but the “Tuberculosis or cancer” node into uniform distributions because HUGIN will normalize the values of the distribution tables if the sum of the column values differs from one. A uniform distribution signifies ignorance (i.e., we have no a priori knowledge about the probability distribution of a variable). Figure 2 shows the initial marginal probability distributions prior to parameter estimation.



```
Domain
  Bronchitis?
    50.00 yes
    50.00 no
  Dyspnoea?
    50.00 yes
    50.00 no
  Lung cancer?
    50.00 yes
    50.00 no
  Positive X-ray?
    50.00 yes
    50.00 no
  Smoker?
    50.00 yes
    50.00 no
  Tuberculosis or cancer?
    75.00 yes
    25.00 no
  Tuberculosis?
    50.00 yes
    50.00 no
  Visit to Asia?
    50.00 yes
    50.00 no
```

Figure 2: Probability distributions after setting all values in the conditional probability tables to 1.

Now we will try to learn the probabilities from a data file associated with this network. The EM-Learning Wizard will help guide you through the process. Go to Edit Mode and select the EM-Learning Wizard from the Wizard menu as shown in Figure 3.

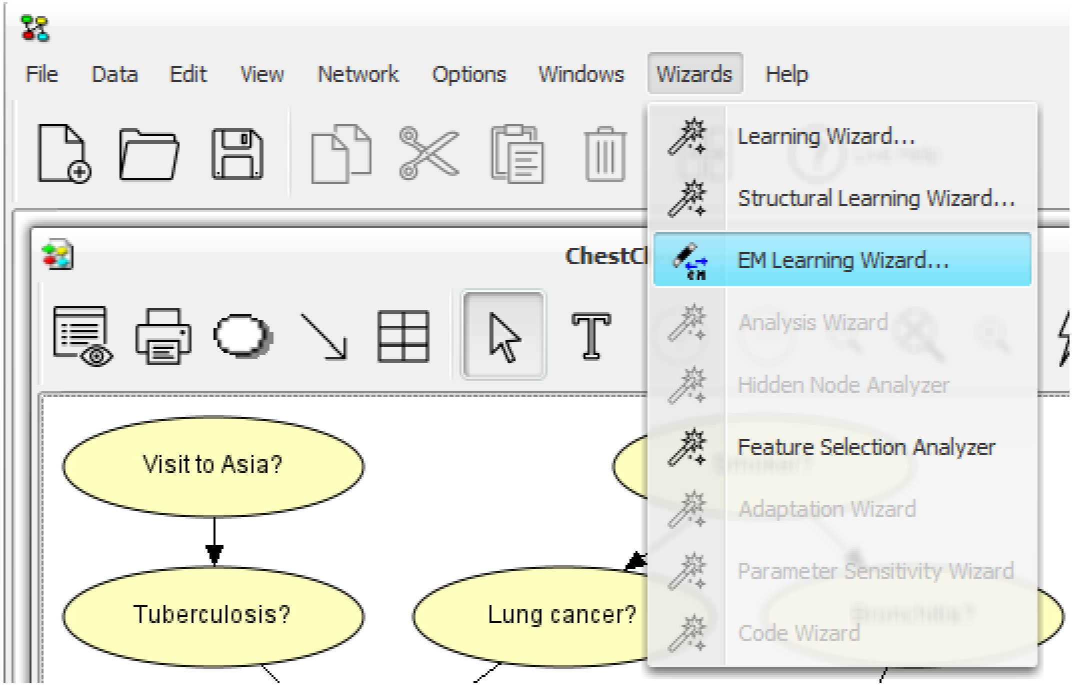
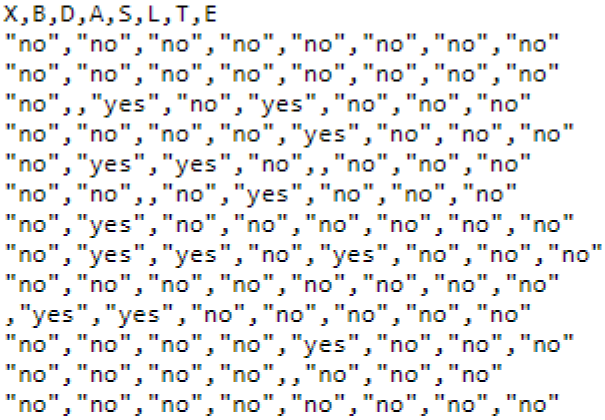


Figure 3: The option to activate the EM-Learning functionality.

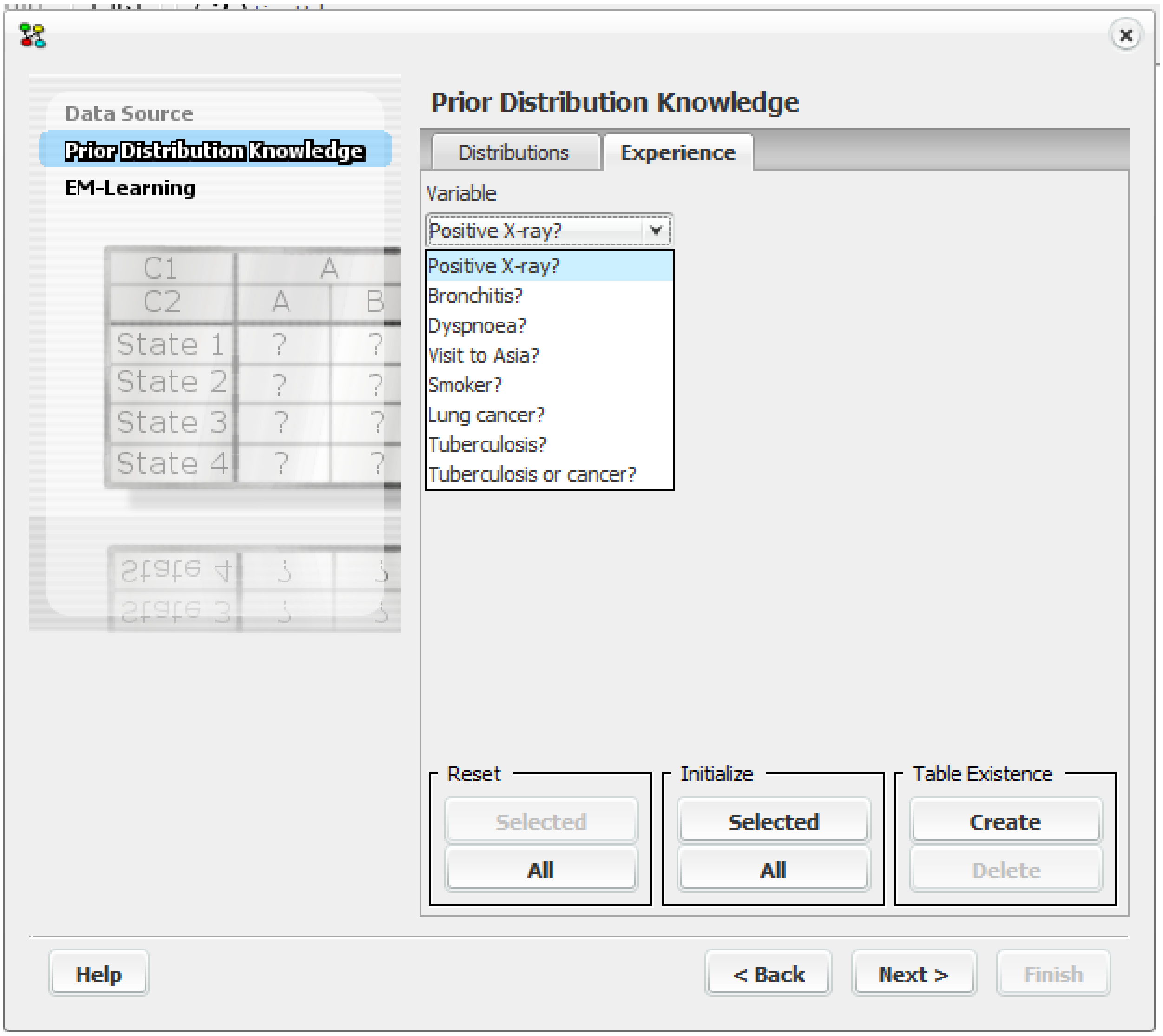
Click the “Load File” button and choose a file from which the conditional distribution probabilities are to be learned. Choose the `asia.dat` file, which is located in the same directory as the network. The first few lines of the file are shown below.



```
X,B,D,A,S,L,T,E
"no","no","no","no","no","no","no","no"
"no","no","no","no","no","no","no","no"
"no",,"yes","no","yes","no","no","no"
"no","no","no","no","yes","no","no","no"
"no","yes","yes","no",,"no","no","no"
"no","no",,"no","yes","no","no","no"
"no","yes","no","no","no","no","no","no"
"no","yes","yes","no","yes","no","no","no"
"no","no","no","no","no","no","no","no"
,"yes","yes","no","no","no","no","no"
"no","no","no","no","yes","no","no","no"
"no","no","no","no",,"no","no","no"
"no","no","no","no","no","no","no","no"
```

Figure 4: The first few lines of the asia.dat file.

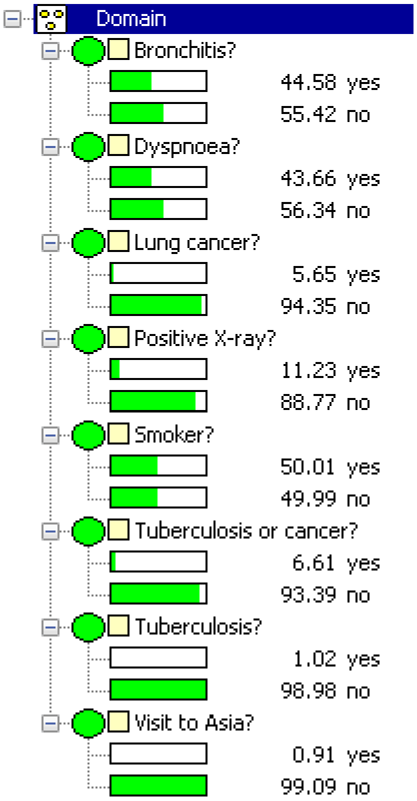
The first line is the name of the nodes as in the network, the rest are the evidence for each experiment/observation. After the file is selected click the “Load” button and then the “Next” button to open the prior distribution knowledge dialog as shown in Figure 5. Now you must create *Experience Tables* (page 231) for all variables except Turberculosis or cancer, by selecting each of them and pressing the “Create” button under Table Existence as shown in Figure 6.



Variable: Positive X-ray?

Figure 5: the Prior Distribution Knowledge dialog

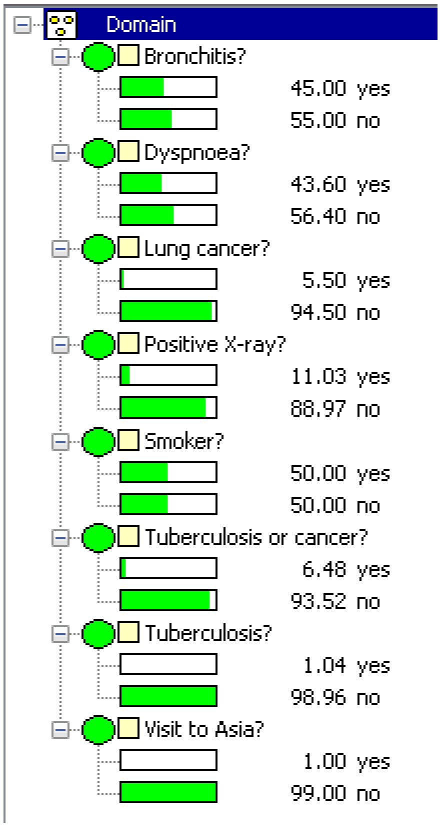
Once you have created an experience table for all variables, click the “All” button under Initialize. This will start the EM-Learning algorithm. Based on the data, the EM-algorithm computes the conditional probability distribution for each node. Figure 5 shows the new conditional probability distribution after EM-learning finishes. As can be seen from Figure 6, all the conditional probability distribution values have changed to reflect all the cases in the case file.



```
Domain
  Bronchitis?
    44.58 yes
    55.42 no
  Dyspnoea?
    43.66 yes
    56.34 no
  Lung cancer?
     5.65 yes
    94.35 no
  Positive X-ray?
    11.23 yes
    88.77 no
  Smoker?
    50.01 yes
    49.99 no
  Tuberculosis or cancer?
     6.61 yes
    93.39 no
  Tuberculosis?
     1.02 yes
    98.98 no
  Visit to Asia?
     0.91 yes
    99.09 no
```

Figure 6: The estimated marginal probability distributions.

Figure 7 shows the marginal probability distributions for the network from which the set of cases was generated. Note that the probability distribution shown here is for the original `Chest Clinic` network.



```
Domain
  Bronchitis?
    45.00 yes
    55.00 no
  Dyspnoea?
    43.60 yes
    56.40 no
  Lung cancer?
     5.50 yes
    94.50 no
  Positive X-ray?
    11.03 yes
    88.97 no
  Smoker?
    50.00 yes
    50.00 no
  Tuberculosis or cancer?
     6.48 yes
    93.52 no
  Tuberculosis?
     1.04 yes
    98.96 no
  Visit to Asia?
     1.00 yes
    99.00 no
```

Figure 7: marginal probability distributions of the original network

The EM algorithm can both estimate the conditional probability tables of discrete chance nodes and the conditional probability densities of continuous chance node.

3.2.15 Adaptation Tutorial

Adaptation is the process of refining the (conditional) probabilities specified for a Bayesian network by taking into consideration real experiment outcomes or cases. For example, assume we have a Bayesian network relating the causal relationships between a set of diseases and a set of symptoms. Every time a patient is diagnosed, the information about his/her symptoms and the diseases he/she is suffering from can be used to adapt the network's probabilities. Using the HUGIN Graphical User Interface, we can adapt a network by using experience tables and/or fading tables. Below is a detailed discussion of how to perform adaptation in the HUGIN Graphical User Interface, including an overview of the computations performed. The mathematics is presented for those who want to get a deeper understanding of the adaptation process and can be skipped without any discontinuity in the reading. The *Adaptation Algorithm* (page 244) is well-suited for sequential parameter update whereas the *EM Algorithm* (page 352) is well-suited for parameter estimation in batch (i.e., estimation of the parameters of conditional probability tables from a set of cases).

Experience Tables In Adaptation

Experience Tables (page 231) are used to specify prior experience on the parameters of the conditional probability distributions. Notice that adaptation is disabled for nodes without an experience table and parent configurations that have a zero experience count.

Example of Adaptation

We use the example from the *Experience Tables* (page 231) to illustrate the use of adaptation. The Bayesian network is shown in Figure 1. download `Chest Clinic` to learn more about the domain of this network.

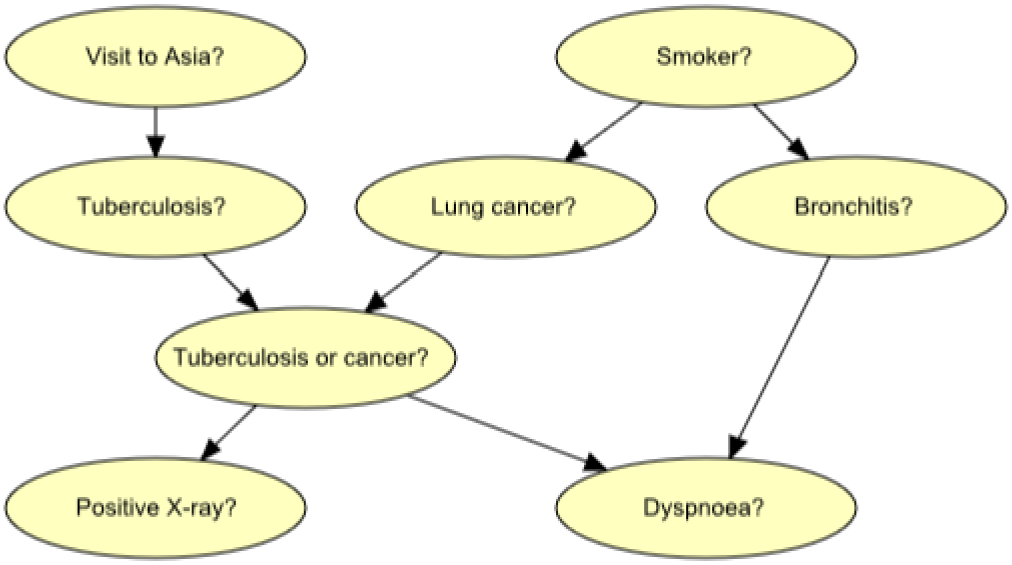


Figure 1: Bayesian-network representation of “Chest Clinic”.

The conditional probability distribution (represented by a conditional probability table (CPT)) of the variable Smoker (S) prior to any adaptation is (0.5,0.5). We assume that the experience count for Smoker has been set to 10 as in the example. Now compile the network, enter the observation (evidence) that variable S is in state 0 (“yes”), and click the adaptation button . Click the adaptation button four more times, denoting that this observation has been made five times.

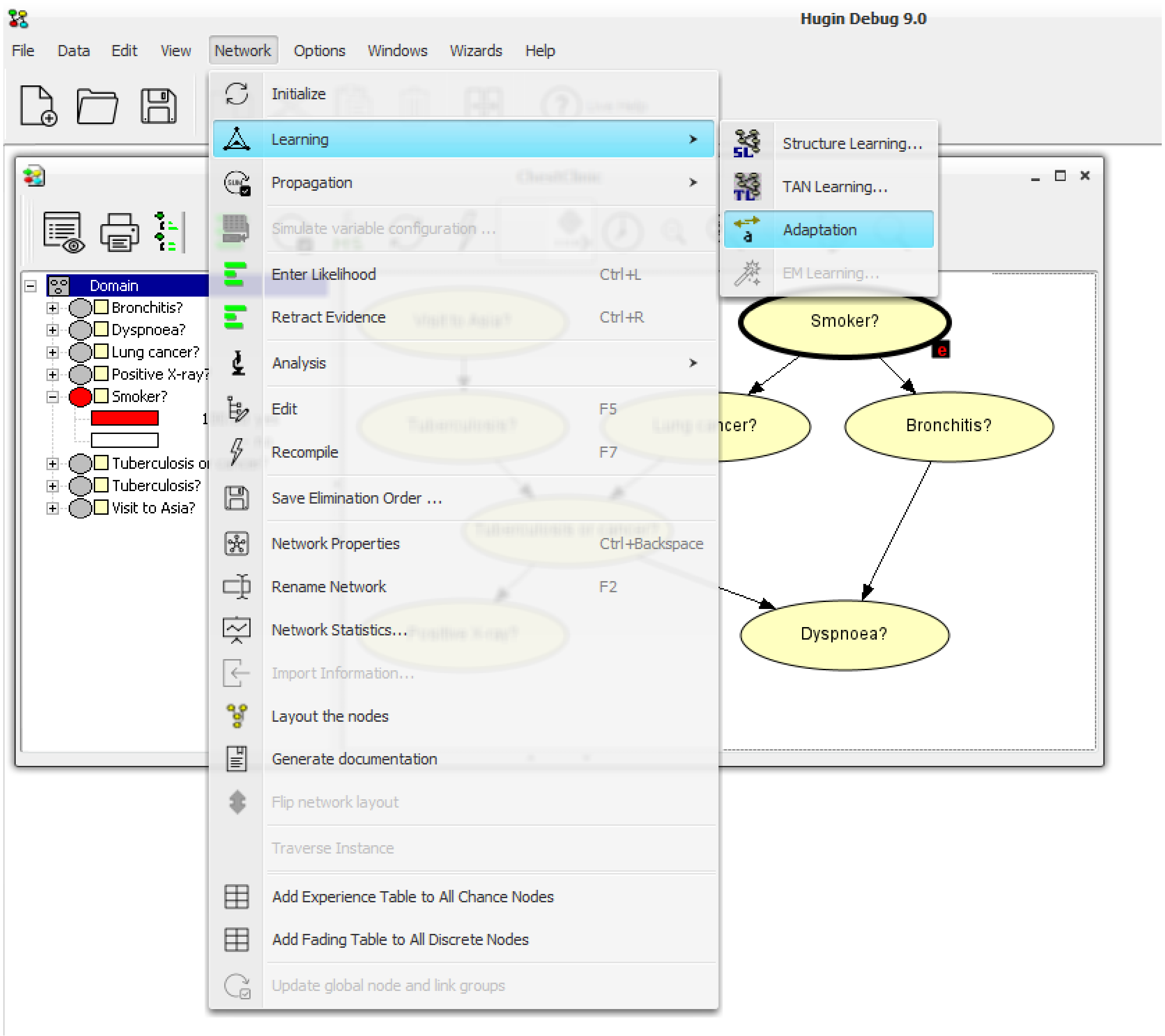


Figure 1: The adapdation option.

Now, if we reinitialize the network, we will see that the CPT of variable S has changed from (0.5,0.5) to (0.667,0.333). Also, all the variables that are not independent of S have different probability distributions. In general, the CPTs of the variables ancestral to the observed variables may be updated when the adaptation button is clicked, provided their experience tables exist and the counts are greater than zero. Note that deleting or resetting the experience tables does not affect the CPTs of the variables.

To understand why the values have changed, it's easier to see the experience table again. If we go to Edit Mode and see the experience table for the "Smoker?" variable, we will see that the current experience count is 15, corresponding to an initial count of 10 and having made 5 observations of the variable. Before adaptation began we had an experience count of 10, and the probability distribution was (0.5,0.5), i.e., 5 of them were "yes"; and the other 5 were "no".

The additional 5 counts all pertain to state "yes". Therefore, the adapted probability distribution of "Smoker?" becomes

$$P(Smoker = ''yes'') = \frac{N(''yes'')}{(N(''yes'') + N(''no''))} = 10/15 = 0.667,$$

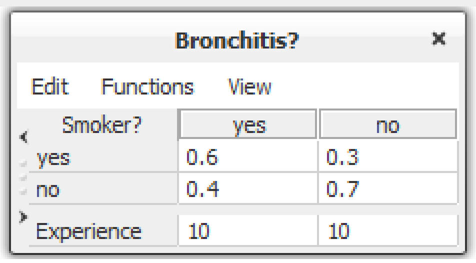
where N("x") denotes the number of times we experienced state "x". The result agrees with the values we saw. Below is a definition of adaptation, which gives a concise description of what we have been doing so far.

To summarize, an adaptation step consists of entering evidence, propagating, and updating (adapting) the conditional probability tables and the experience tables.

To make the experiments specified below, please close the network file, and choose “No” when the Hugin Graphical User Interface shows the prompt to save the file, so that we can begin with the original file.

In the network of Figure 1, it is justified to add experience tables to all the variables in the domain except for “Tuberculosis or cancer”, since this variable is a *logical OR* and no experience can be gained on logical OR variables. After we have added experience tables to all variables (by choosing “Add Experience Table to All Discrete Chance Nodes”), we have to modify the experience counts because they are initially all zero. In our case, we add experience tables to all variables in the network (except the “Tuberculosis or cancer” variable) and set the initial experience counts to 10. The network is now ready for adaptation.

The experience table of a variable represents the experience counts of the parent configurations, as mentioned earlier. Figure 3 shows the CPT and the experience table for “Has Bronchitis”.



Smoker?	yes	no
yes	0.6	0.3
no	0.4	0.7
Experience	10	10

Figure 3: The CPT and experience table for “Has Bronchitis” before adaptation.

The interpretation of Figure 3 is that we have observed the value of the “Has Bronchitis” 20 times, 10 of them when “Smoker?” was in state “yes” and 10 when “Smoker?” was in state “no”. Combining this information with the CPT of “Has Bronchitis”, we can get information about the initial experience of each individual state. Since there are 10 observations of “Has Bronchitis” when “Smoker?” was in state “yes” and

$$P(HasBronchitis = ''yes''|Smoker? = ''yes'') = 0.6,$$

we have

$$N(HasBronchitis = ''yes''|Smoker? = ''yes'') = 0.6 * 10 = 6,$$

where we have used a new notation for N to conform with conditional probability notation. Similarly, for Smoker?=”no” we have

$$N(HasBronchitis = ''yes''|Smoker? = ''no'') = 0.3 * 10 = 3,$$

giving the total individual state experience counts of

$$N(''yes'') = 6 + 3 = 9$$

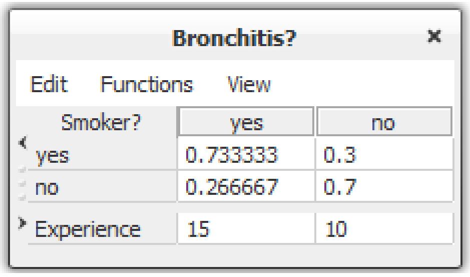
and .. math:: N(“No”) = 20 - 9 = 11.

With these values we can now compile the network and do some adaptation. Adaptation requires at least one variable with an experience table in order to adapt the network. Note that it is not necessary to have non-zero experience counts or to enter identical experience counts for every parent configuration. For instance, the initial experience counts could be set to “10,0”. Note also that we don’t have to worry about the computations which we have been doing so far, as they are only intended for users who want a deeper understanding of the adaptation process.

Let’s say we have observed five cases for which

- Variable S (Smoker?) is in state 0 (“yes”)
- Variable B (Has Bronchitis) is in state 0 (“yes”)

Then propagate this evidence. Next click the adaptation button five times as we have done before. Each time the adaptation button is clicked the probability of this observation increases. Now reinitialise the network and observe the conditional probability distribution of “Smoker?” as we did before. As before, the conditional probability distribution has changed from (0.5,0.5) to (0.667,0.333) since the adaptation we just performed is the same as before in the context of variable “Smoker?”. As for “Has Bronchitis?”, the new CPT and the updated experience table is shown in Figure 4.



Smoker?	yes	no
yes	0.733333	0.3
no	0.266667	0.7
Experience	15	10

Figure 4: The CPT and experience table for “Has Bronchitis” after adaptation.

The right hand columns of Figures 3 and 4 (i.e., P(Has Bronchitis = “yes” | Smoker? = “no”) and P(Has Bronchitis = “no” | Smoker? = “no”)) haven’t changed. This is because in all the five adaptation steps that we ran, evidence was entered that variable “Smoker?” was in state “yes”, giving no new information about the states of “Has Bronchitis” when the state of “Smoker?” is “no”. We can see similar information also from Figure 4, where the experience under “yes” has increased by 5 while the one under “no” remained the same.

On the other hand, the left side column (i.e., P(Has Bronchitis = “yes” | Smoker? = “yes”) and P(Has Bronchitis = “no” | Smoker? = “yes”)) has changed, showing that some experience has been gained. Using the above results from the computation before adaptation was performed, we have

$$N(HasBronchitis = "yes" | Smoker? = "yes") = 6 + 5 = 11$$

$$N(Smoker? = "yes") = 15$$

$$P(HasBronchitis = "yes" | Smoker? = "yes") = 11/15 = 0.7333$$

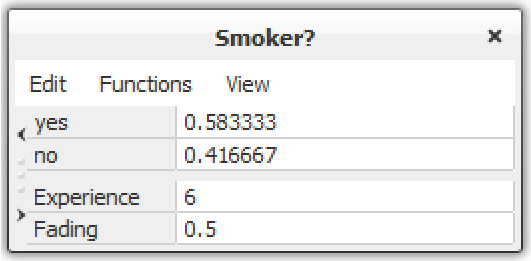
agreeing with the results displayed in the HUGIN Graphical User Interface.

Fading Tables

The adaptation procedures we have gone through so far gave equal weight to recent experiences and older ones (that is why we were just adding and dividing as if they were observations done at the same time). Often, however, old observations do not count as much as more recent ones. Thus we have to *unlearn* or forget some of them. This is the same as saying that new observations (evidence) are more important than older evidence and hence should have more weight in the adaptation process. We can make HUGIN respond to such situations by using fading tables.

Fading tables are meaningless without experience tables (i.e., in order to forget something we must have had remembered it first). In order to implement fading, we have to introduce a *fading factor*, which is the rate at which previous observations are forgotten. A fading factor of 0 means that there is no adaptation and a fading factor of 1 means there is no fading.

The following is a description of how to use fading tables in the HUGIN Graphical User Interface and how they work. Close the network we have been working with so far without saving it or adjust the experience counts to 10 and set the conditional probabilities back to (0.5,0.5). The steps needed to add a fading table to a variable are similar to those for adding an experience table to the variable. To add a fading table to a discrete chance variable select the variable, do a right click, and select “Add Fading Table” in the “Experience/Fading Table Operations” submenu. Then select “Show fading table” item of the View menu of the node table for the “Smoker?” node and enter the fading factor in the fading table section of the node table. The fading factor should be a number between 0 and 1. If we enter 0 or a number greater than 1, no matter how many times we try to adapt the network, nothing will change. If we choose 1, it will work as if there was no fading taking place. Note that if a variable does not have an experience table, then it is not possible to add a fading table to the variable. For this example, enter a fading factor of 0.5 to the “Smoker?” variable. Now enter the evidence that Smoker? is in state 0 (“yes”). Click the adapt button once and switch to Edit Mode to see the updated CPT and experience table of “Smoker?”. The results are shown in Figure 5.



yes	0.583333
no	0.416667
Experience	6
Fading	0.5

Figure 5: The CPT, experience table, and fading table for “Smoker?” after adaptation.

One thing that you might have noticed is that the experience count has decreased from 10 to 6. How come the experience count decreased while we were adding some experience to the network? To explain this, let’s consider the fading factor. After one adaptation run the experience counts and probabilities can be computed as

$$N(Smoker? = ”yes”) = factor * N(previous\_yes) + 1 = 0.5 * 5 + 1 = 3.5$$

$$N(Smoker? = ”no”) = factor * N(previous\_no) = 0.5 * 5 = 2.5$$

$$N(experience) = 3.5 + 2.5 = 6$$

$$P(Smoker? = ”yes”) = 3.5/6 = 0.583333$$

$$P(Smoker? = ”no”) = 2.5/6 = 0.416667,$$

where we again use the somewhat informal N notation. N(*previous_yes*) and N(*previous_no*) are the previous experience counts in the “yes” and “no” states respectively and N(*experience*) is the total experience count for the concerned variable. To add fading tables to all discrete chance variables in the network click the right mouse button somewhere in the Network Pane, choose “Add Fading Table to All Discrete Chance Nodes” in the “Experience/Fading Table Operations” submenu, and add the fading factor for each variable.

CHAPTER

FOUR

MANUAL

This manual describes the HUGIN Graphical User Interface. All topics covered by this manual can be found in the index below.

You can also go to the Main Window topic, the Edit Mode topic, or the Run Mode topic and use their clickable images to find the information that you need:

- *Main Window* (page 186)
- *Edit Mode* (page 124)
- *Run Mode* (page 123)

4.1 Functions and Tools

4.1.1 Gaussian (Normal) Distribution Function

Continuous chance nodes (page 224) in a HUGIN network can be specified to have a *Gaussian (or normal) distribution function*. This text does not cover a mathematically description of the Gaussian distribution function; it only describes how it can be used in HUGIN network nodes.

The Gaussian distribution function is defined on the complete real axis. Any Gaussian distribution function can be specified by its mean and variance parameter. In HUGIN, a continuous chance node can have a single Gaussian distribution function for each configuration of its discrete parents states (both discrete chance nodes and decision nodes). If a continuous chance node has one or more continuous chance node parents, the mean can be linearly dependent on the states of these continuous parents.

In general, the distribution for a continuous variable Y with discrete parents I and continuous parents Z is a (one-dimensional) Gaussian distribution conditional on the values of the parents:

$$P(Y|I=i, Z=z) = N(\alpha(i) + \beta(i)^\tau z, \gamma(i))$$

Figure 1 shows an example of the specification of a Gaussian distribution function of a continuous chance node (C4) having one discrete chance node (C1) and two continuous chance nodes (C2 and C3) as parents.

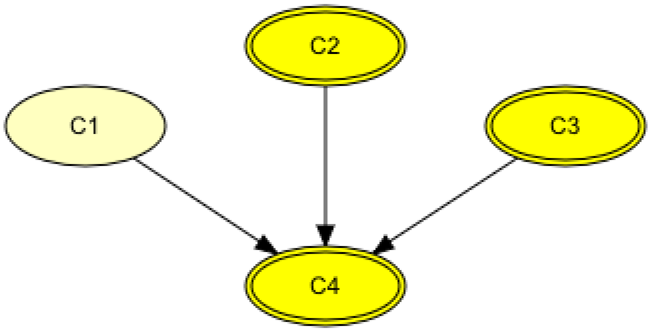
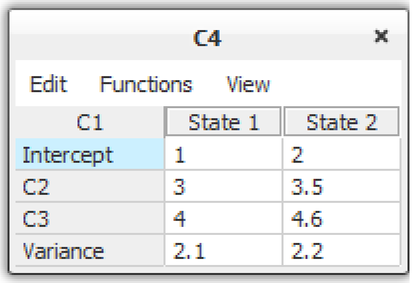


Figure 1: An example of a network where the continuous chance node C4 has one discrete chance node (C1) and two continuous chance nodes (C2 and C3) as parents.



C1	State 1	State 2
Intercept	1	2
C2	3	3.5
C3	4	4.6
Variance	2.1	2.2

Figure 2: The specification of a Gaussian distribution function for C4 that has one discrete chance node (C1) and two continuous chance nodes (C2 and C3) as parents.

This specification gives a conditional Gaussian distribution function for each of the states of C1 (where *N(m,v)* is the Gaussian distribution function with mean m and variance v):

- P(C4 | C1=”State 1”, C2 = x, C3 = y) = N(1 + 3x + 4y, 2.1)
- P(C4 | C1=”State 2”, C2 = x, C3 = y) = N(2 + 3.5x + 4.6y, 2.2)

The mean of each distribution function for C4 is thus a sum of a specified mean parameter and a weighted sum over the values of the continuous parents, where the weights are given by the numeric values in the C2 and the C3 rows of the table in Figure 2.

Only the mean depends linearly on the continuous parent nodes. The variance is constant for each configuration of the states of the discrete parents.

See *Introduction to Bayesian Networks* (page 19) for an example of the modeling of a real-world problem involving several discrete and continuous variables.

4.1.2 Continuous-Discrete Node Value Transfer Function

The *Continuous-Discrete node Value Transfer function (CDVT function)* is available in *Analysis Menu* (page 288) under the *Network Menu* (page 185). In essence the CDVT function implements a modelling technique to circumvent the limitation of no discrete children of continuous nodes in CLG Bayesian networks. A discrete interval node is linked to a continuous node such that the expression generating the distribution of the discrete node is equal to the posterior marginal of the parent continuous node. In this way the discrete node becomes a discretize representation of the continuous node. The discrete node has no table as the table is regenerated using the Table Generator functionality. The states of the discrete node are determined by the user prior to compiling the network. Remember to add -infinity and infinity as lower- and upper limits in the discrete interval node.

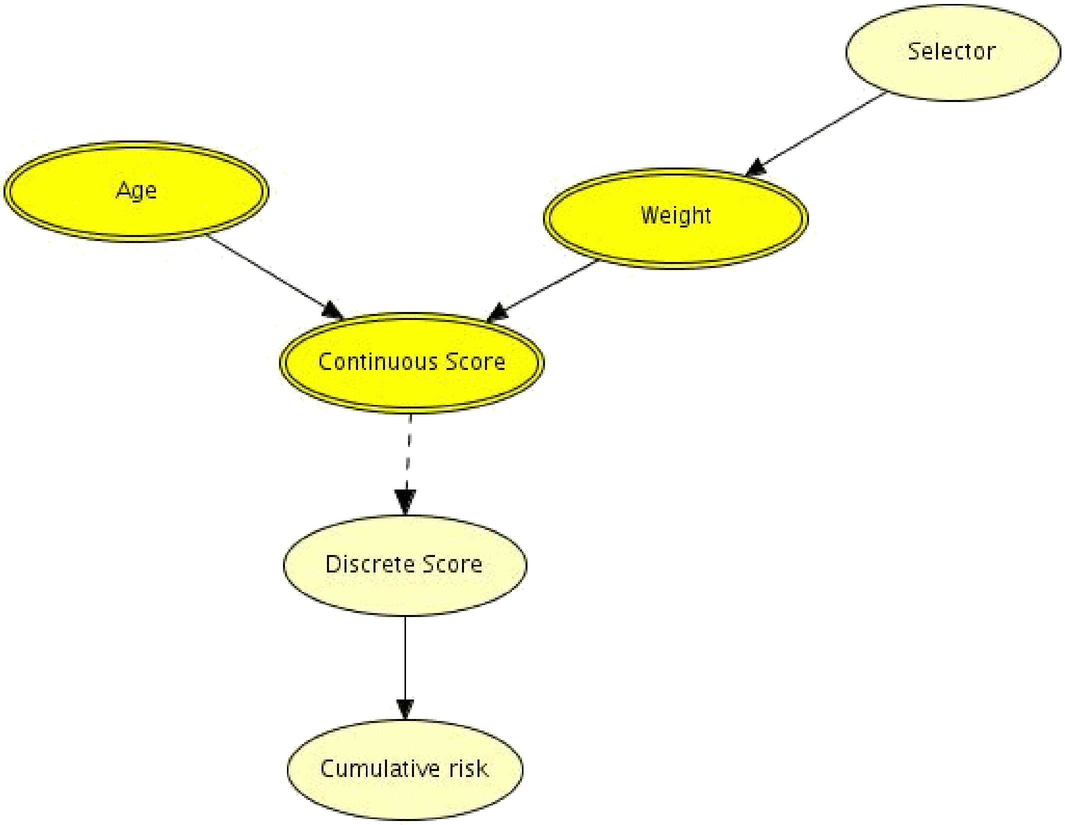
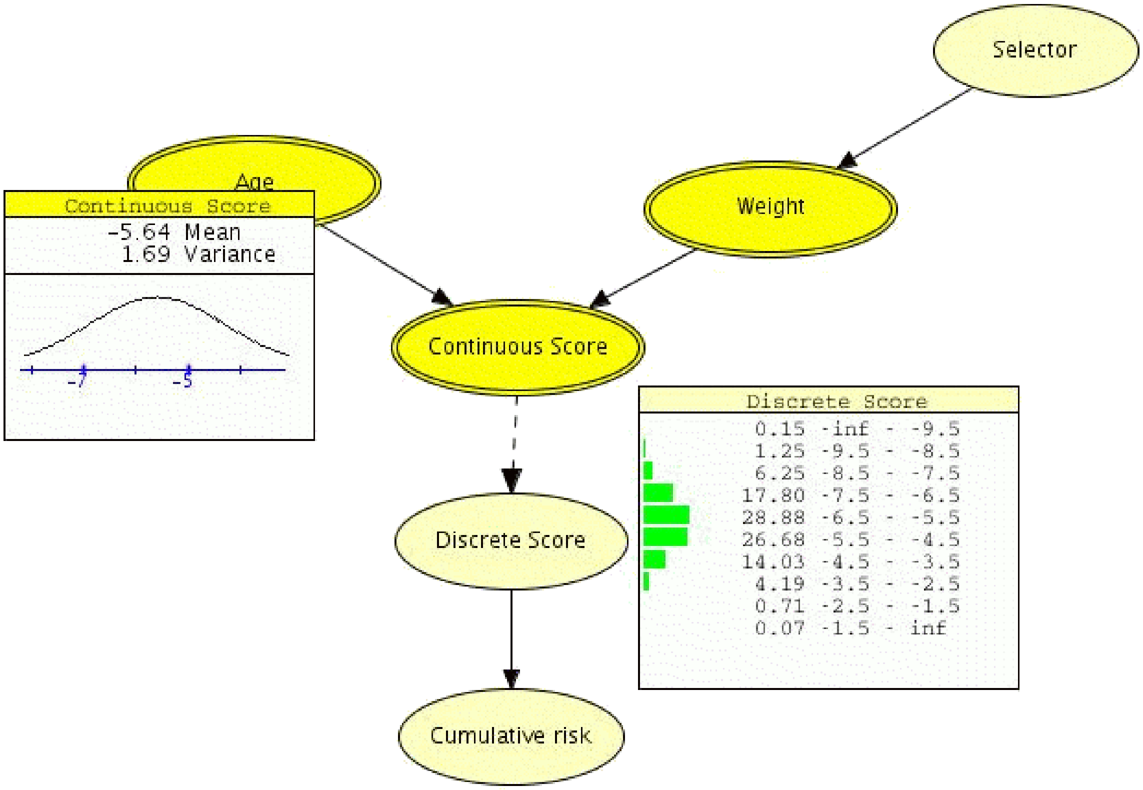


Figure 1: The dashed arrow specifies a CVDT function link.

Figure 1 shows a link between a continuous and discrete nodes pair created by the CDVT function.



Continuous Score

```
-5.64 Mean
 1.69 Variance
```

Discrete Score

```
 0.15 -inf - -9.5
 1.25 -9.5 - -8.5
 6.25 -8.5 - -7.5
17.80 -7.5 - -6.5
28.88 -6.5 - -5.5
26.68 -5.5 - -4.5
14.03 -4.5 - -3.5
 4.19 -3.5 - -2.5
 0.71 -2.5 - -1.5
 0.07 -1.5 - inf
```

Figure 2: The Discrete Score is a discrete approximation of the Continuous Score computed using Table Generator functionality.

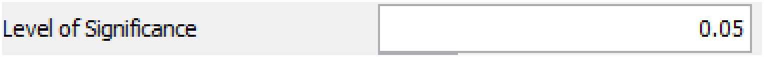
Figure 2 illustrates how the density function of a continuous node can be approximated by a discrete distribution using the CDVT function.

The CDVT function has some major limitations such as evidence can only be propagated along the direction of the CDVT link, only sum-normal equilibrium is supported and no type of analysis is supported on a network with CDVT links.

The discretization link should be used with extreme care.

4.1.3 Level of Significance

The structure learning algorithms are based on making dependence tests that calculate a test statistic which is asymptotically chi-squared distributed assuming (conditional) independence. If the test statistic is large for a given independence hypothesis, the hypothesis is rejected; otherwise, it is accepted. The probability of rejecting a true independence hypothesis is given by the *level of significance*.



Level of Significance 0.05

Figure 1: Text box in which the user specifies the level of significance to be used by the structure learning algorithms.

The level of significance can be entered into this text field. The default value is set to 0.05, which should be appropriate for most learning sessions, but it can take on any value in the open interval (0;1). In general, the higher the significance level the more links will be included in the learned structure. Reducing the level of significance will usually also reduce the running time of the structure learning algorithms.

See also the *PC algorithm* (page 246) and the *NPC algorithm* (page 246).

4.1.4 Traverse Instance

The "Traverse Instance" function can be invoked by selecting an instance node in run mode, and selecting either "Traverse Instance" from the Network Menu, or from the popup-menu (brought up by a right-click with the mouse). This will open a new window containing a compiled version of the class from which the instance is derived (See figure 1).

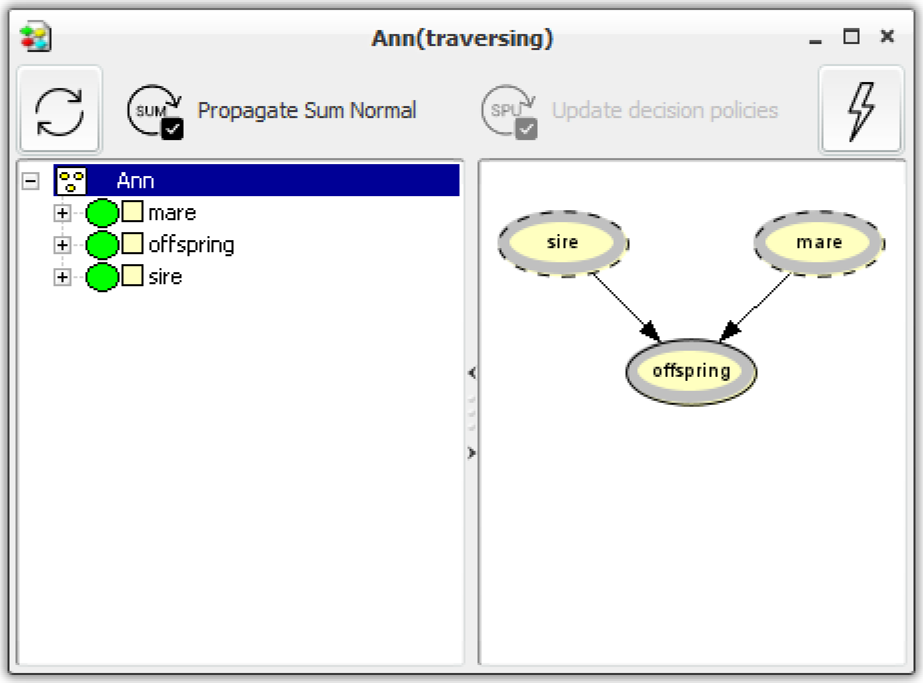


Figure 1: A traversal frame for the Ann class in the Studfarm network.

The functionality of the traversal frame is very limited. Only operations concerning insertion and propagation of evidence are included.

Traversing instances can be very useful when working with OOBNs. In the main network (i.e., the network with the instance nodes which are to be traversed) there is no way of examining the structure of the instance nodes. The only way of interacting with the nodes in the instance is by clicking on the nodes in the *belief bar* (page 243), and this contains no structural information (except for encapsulating nodes in instances). By traversing the frame, this structure is revealed while still being able to insert and retract evidence. Obviously, recursive traversal is possible as well, so that any instances present in the traversal frame can themselves be traversed.

4.1.5 Simulation

Based on the joint probability distribution induced by the model and propagated evidence, the HUGIN Graphical User Interface generates an instantiation of all (uninstantiated) variables in the model.

4.1.6 Markov Blanket

The markov blanket for a node can be displayed by selecting the node and then click the *Show Markov Blanket* option in the the *d-separation* (page 288) pane.

The Markov Blanket of a node A is the set consisting of the the parents of A, the children of A, and the nodes sharing a child with A.

If all variables in the Markov blanket for A are instantiated, then A is d-separated from the rest of the network.

For example the markov blanket for the node *I* in Figure 1 is *{C, E, H, K, L}*, which can be seen in Figure 2.

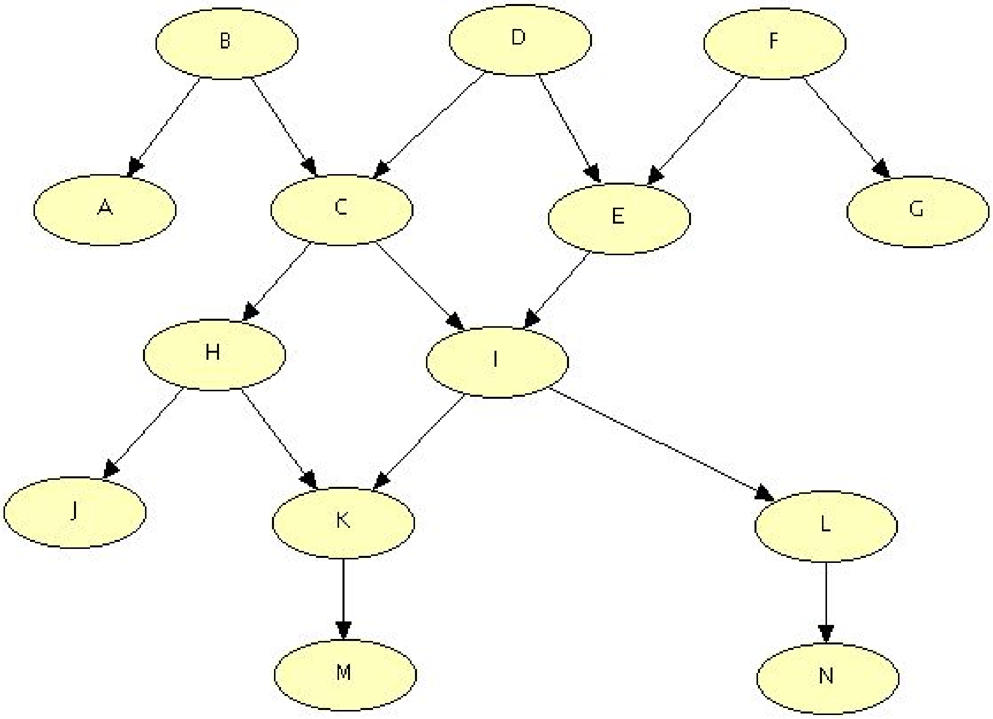


Figure 1: Network.

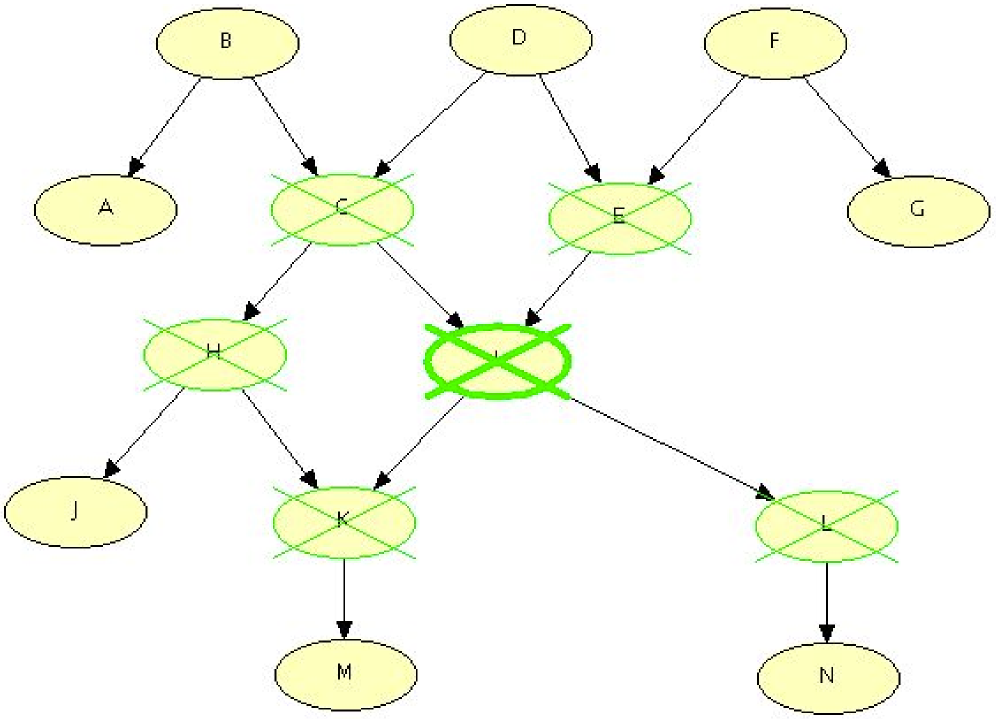


Figure 2: Markov blanket for node I.

4.1.7 Run Mode

To use HUGIN networks for computations, the HUGIN Graphical User Interface must be working in Run Mode. Figure 1 shows a *Network Window* (page 182) in Run Mode. In Run Mode, the network window is split into a *Node List Pane* (page 243) (unless this has been disabled via the Toggle Belief Bar item of the *View Menu* (page 188)) on the left and a *Network Pane* (page 177) on the right, showing the network. In the Node List Pane one can see the states of the nodes and their beliefs (also, expected utilities in LIMIDs), probabilities and expected utilities for *decision nodes* (page 224) and mean & variance for *continuous chance nodes* (page 224)). The states of a node appears when *expanding* (page 231) it.

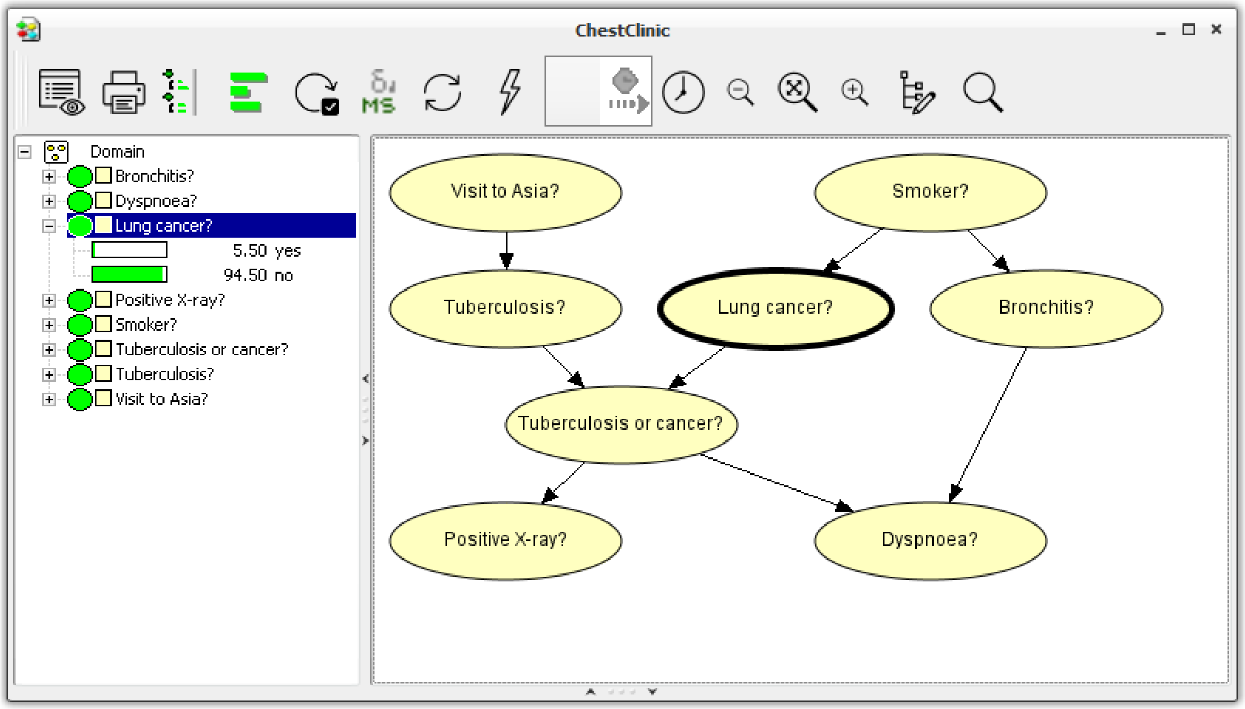


Figure 1: The network window of a HUGIN network running in Run Mode.

To switch from *Edit Mode* (page 124) to Run Mode, click the *compile* (page 126) button in the Edit Mode Tool Bar or select the Compile item of the *Network Menu* (page 185). Figure 2 shows the compile button.

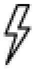
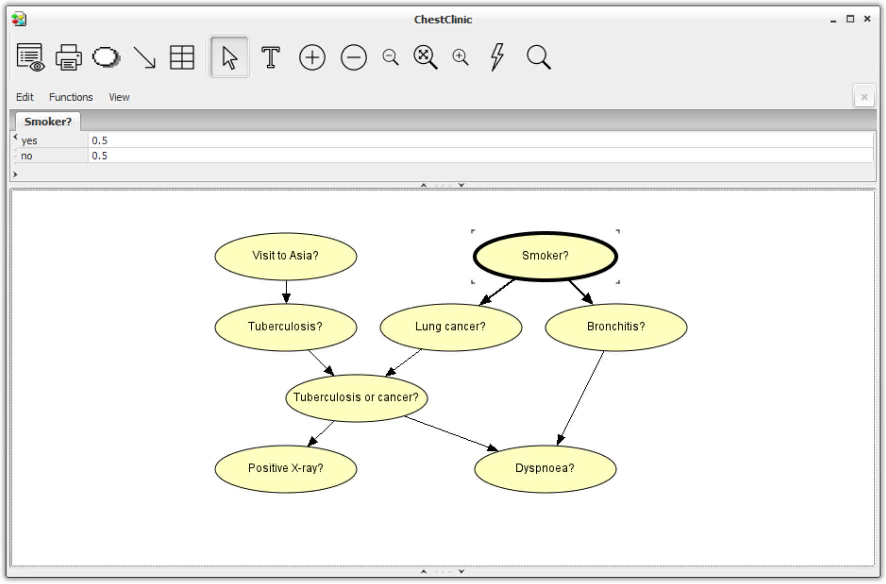


Figure 2: The compile tool button from the network window Tool Bar.

4.1.8 Edit Mode

When you are constructing your HUGIN network, you are running in Edit Mode. In Figure 1 you can see a *network window* (page 182) of a HUGIN network running in Edit Mode. In Edit Mode, the network window is split up in a *Tables Pane* (page 239) containing the table of the currently selected node. Below the Tables Pane is the *Network Pane* (page 177) where the graph describing the network is found.



Smoker?	
yes	0.5
no	0.5

Figure 1: The *network window* (page 182) of a HUGIN network running in Edit Mode.

To switch from *Run Mode* (page 123) to Edit Mode, press the Edit Mode tool button (see Figure 2).

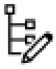


Figure 2: The Edit Mode tool button from the Run Mode Tool Bar.

4.1.9 Compress

To optimize a HUGIN network you can also use the compression facility activated by the Compress check box found in the *Compilation* (page 126) tab of the *Network Properties* (page 178) dialog box.

Compression will optimize the way the tables of the internal representation are stored. Basically, you can save space any time a table entry has a zero value (probability). This means that the more zero-probabilities you find in the tables of your HUGIN network, the more effect will you get from using the compression option.

You can compress even more using *approximation* (page 125) (if you are willing to work with less precision).

4.1.10 Approximate

If you need to (want to) make your HUGIN network even faster than what you can achieve by ordinary *compression* (page 125), you can use the approximation option.

Approximation is activated in the *Compilation* (page 126) tab of the *Network Properties* (page 178) dialog box. This function works by eliminating the smallest table values of the internal network representation (making them zero), so that the compression function will be more efficient (compression can save space when there are a lot of zero probabilities).

To specify which table values to eliminate, you must set an epsilon value which specifies how much probability mass to eliminate. Then, small table values are set to zero - from the smallest end, until you have removed an amount of *epsilon* probability mass.

You should be careful when using approximation. It can cause conflicts when rare cases occur, since small (though existing) probability mass has been eliminated.

4.1.11 Compile

The *compile* function is activated from *Edit Mode* (page 124) either by pressing the compile tool button (see Figure 1) or by selecting the Run item of the *Network Menu* (page 185).

Figure 1: The compile tool button from the *Network Window* (page 182) Tool Bar.

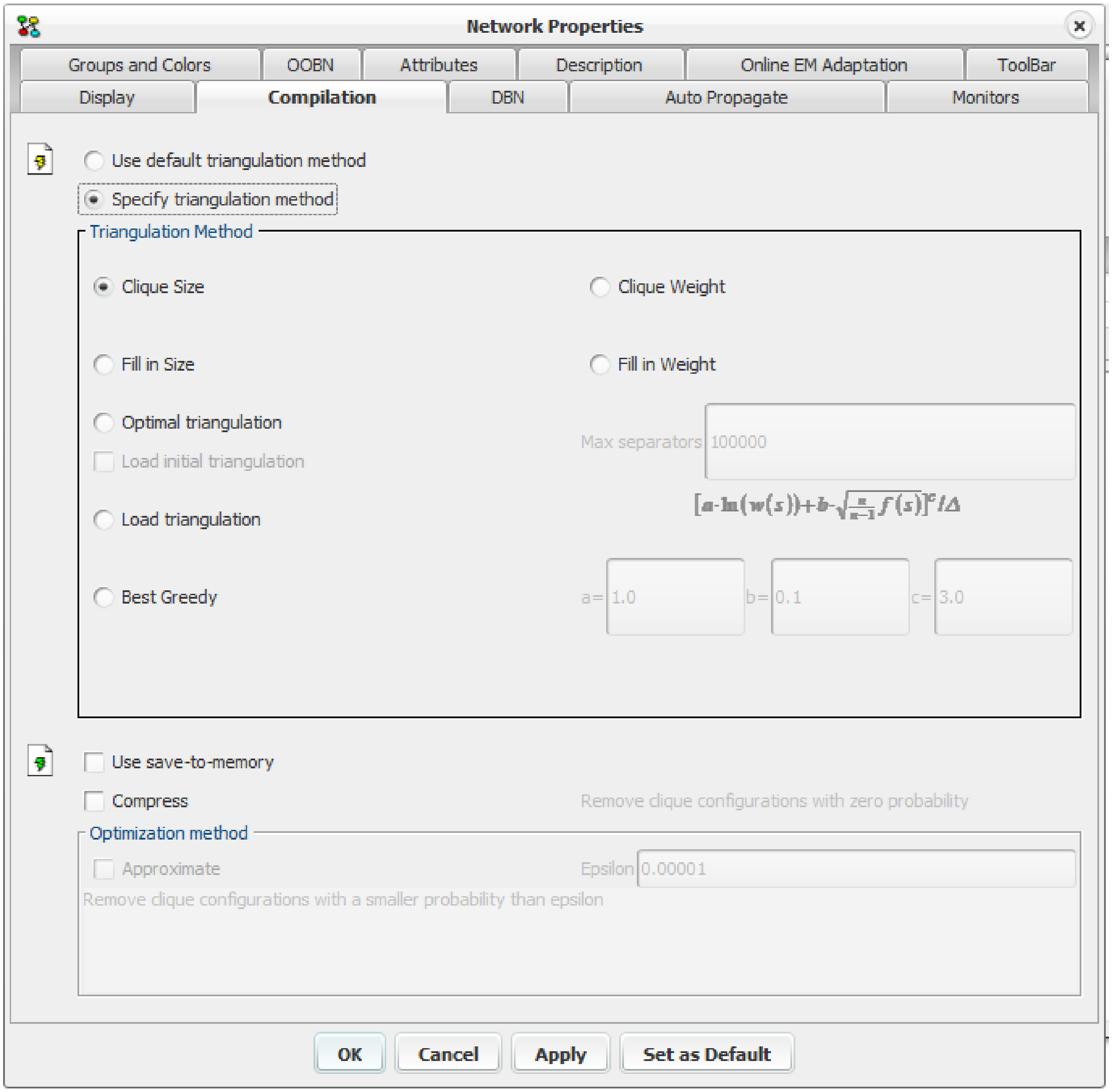
You can also activate the compile function from *Run Mode* (page 123). Here, the Run Mode item of the Network Menu is replaced with the Recompile item.

When activating the compile function from Edit Mode, the current network is compiled and Run Mode is entered.

When activating the compile function from Run Mode, the current network is *recompiled* (page 128) (this can sometimes be useful). If the color of the compile button changes, then this indicates that a recompilation of the network may be require in order to capture any changes made to the class of the network or to classes of instance nodes in the network. Notice that a compilation of a *(limited-memory) influence diagram* (page 345) does not compute updated policies. To update policies it is necessary to perform a *Single Policy Update* (page 258) after compiling the network.

The compilation of a HUGIN network involves a transformation of the network to a *junction tree* (page 254) in which the *propagations* (page 245) can take place.

You can specify certain properties on the compilation in HUGIN from the *Network Properties* (page 178) item of the Network Menu. Figure 2 shows the Compilation tab of the Network Properties dialog box.



○ Use default triangulation method

● Specify triangulation method

Triangulation Method

● Clique Size　○ Clique Weight

○ Fill in Size　○ Fill in Weight

○ Optimal triangulation　Max separators 100000

□ Load initial triangulation

$[a \cdot \ln(w(s)) + b \cdot \sqrt{\frac{n}{n-1}} f(s)]^c / \Delta$

○ Load triangulation

○ Best Greedy　a= 1.0　b= 0.1　c= 3.0

□ Use save-to-memory

□ Compress　Remove clique configurations with zero probability

Optimization method

□ Approximate　Epsilon 0.00001

Remove clique configurations with a smaller probability than epsilon

Figure 2: The Network Properties dialog box showing the Compilation tab.

Generally, it is NP hard (mathematical way to say that it is impossible within reasonable time) to find the best internal representation of a HUGIN network (the junction tree representation). This is basically because it is NP hard to triangulate a graph optimally.

In many cases, however, an optimal (or near-optimal) triangulation can in fact be found. From version 5.4 and onwards, The HUGIN Decision Engine offers a method for optimal triangulation, which in many cases results in junction trees of much lower complexity than those generated using the other (heuristic) triangulation methods provided. The optimal triangulation method works by searching for minimal separators. For some networks this results in extremely many possible optimal separators.

Note on Optimal triangulation options:

Note:

- *Max separators* - To control the running time of the search for an optimal triangulation, the maximum number of minimal separators to generate can be specified. If this value is set to zero, an unlimited number of separators are allowed to be generated.
- $\left[a \cdot \ln(w(s)) + b \cdot \sqrt{\frac{n}{n-1} f(s)}\right]^c / \Delta$ - at each triangulation step a separator that minimize this formula is selected. By specifying values for a, b and c we can tweak the selection criteria - this may be beneficial when searching for triangulations for large networks.
- *a* - specify how to favor weight of the current separator.
- *b* - specify how to favor added weight of extra fill-ins imposed by current separator.
- *c* - influence tolerance for the selection of large separators, delta is the difference in size between the entire graph and the largest sub-graph induced by split using this separator.
- *Load initial triangulation* - instead of using greedy selection of minimal separators to find the initial triangulation, check this field to load an initial triangulation from file (i.e. load a node elimination order from file). **If a good triangulation is already known, then this can be used to speed up the optimal triangulation process significantly!**

The other triangulation methods listed in the Compilation tab are different heuristic methods, which in many cases perform quite well, but, as mentioned above, can result in junction trees that are far from optimal, complexity-wise. Which of the heuristics are the better one depends on the network at hand - so if your network is complex and seems to have poor performance with the current heuristic triangulation method, you might try one of the other heuristics or try the optimal triangulation method.

Finally, a triangulation can be loaded from file (i.e. load a node elimination order from file).

The save-to-memory operation stores a copy of the initial clique potentials in memory. This implies a faster initialization process which may improve efficiency of performing multiple propagations (e.g., as part of the Analysis Wizard).

You can sometimes save some space by using the optimization facilities. These are activated in the bottom half of the Compilation tab. You can choose between ordinary *compression* (page 125) and compression involving *approximation* (page 125). The latter will cause less precision while the first will not.

If you select Approximate, you must also specify an epsilon value stating how much loss of precision you will allow. The larger the epsilon value, the less the precision (read more about this in the *Approximate section* (page 125)).

4.1.12 Recompile

The recompile function is activated from *Run Mode* (page 123) either by pressing the compile tool button (see Figure 1) or by selecting the Recompile item of the *Network Menu* (page 185).

Figure 1: The compile tool button from the *Run Mode* (page 123) Tool Bar.

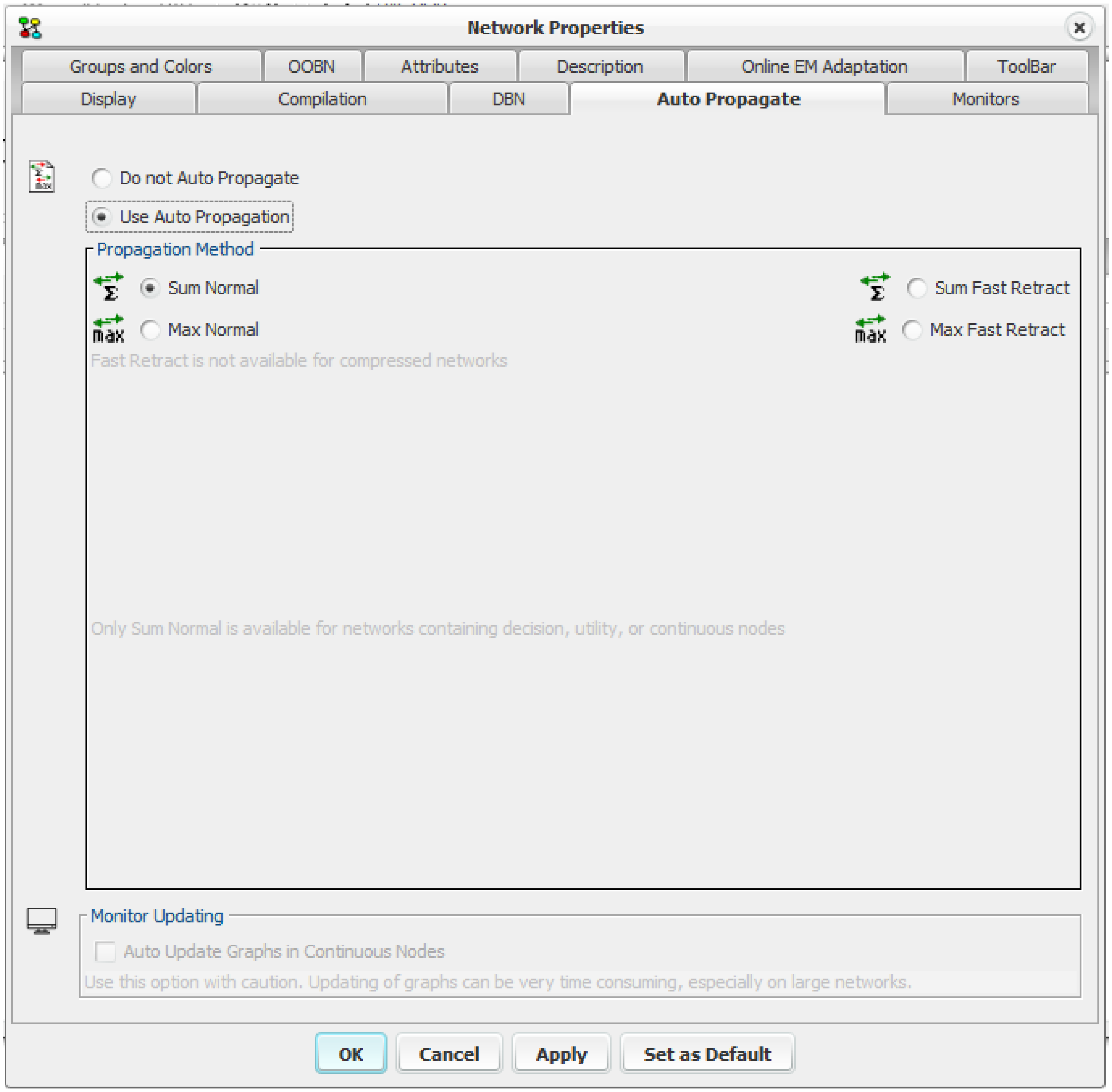
You can use the recompile function to make a new compilation when a network has been saved with evidence and reloaded to get the initial beliefs (if a network was saved with evidence, the shown beliefs will be those computed with this evidence.

See also *Compile* (page 126).

4.1.13 Auto Propagation

Auto propagation can be enabled or disabled in the Auto Propagate tab of the *Network Properties* (page 178) dialog box (see Figure 1).

Auto propagation makes HUGIN propagate automatically with a specific *propagation method* (page 245) every time you enter new evidence.



○ Do not Auto Propagate

● Use Auto Propagation

Propagation Method

● Sum Normal

○ Max Normal

○ Sum Fast Retract

○ Max Fast Retract

Fast Retract is not available for compressed networks

Only Sum Normal is available for networks containing decision, utility, or continuous nodes

Monitor Updating

□ Auto Update Graphs in Continuous Nodes

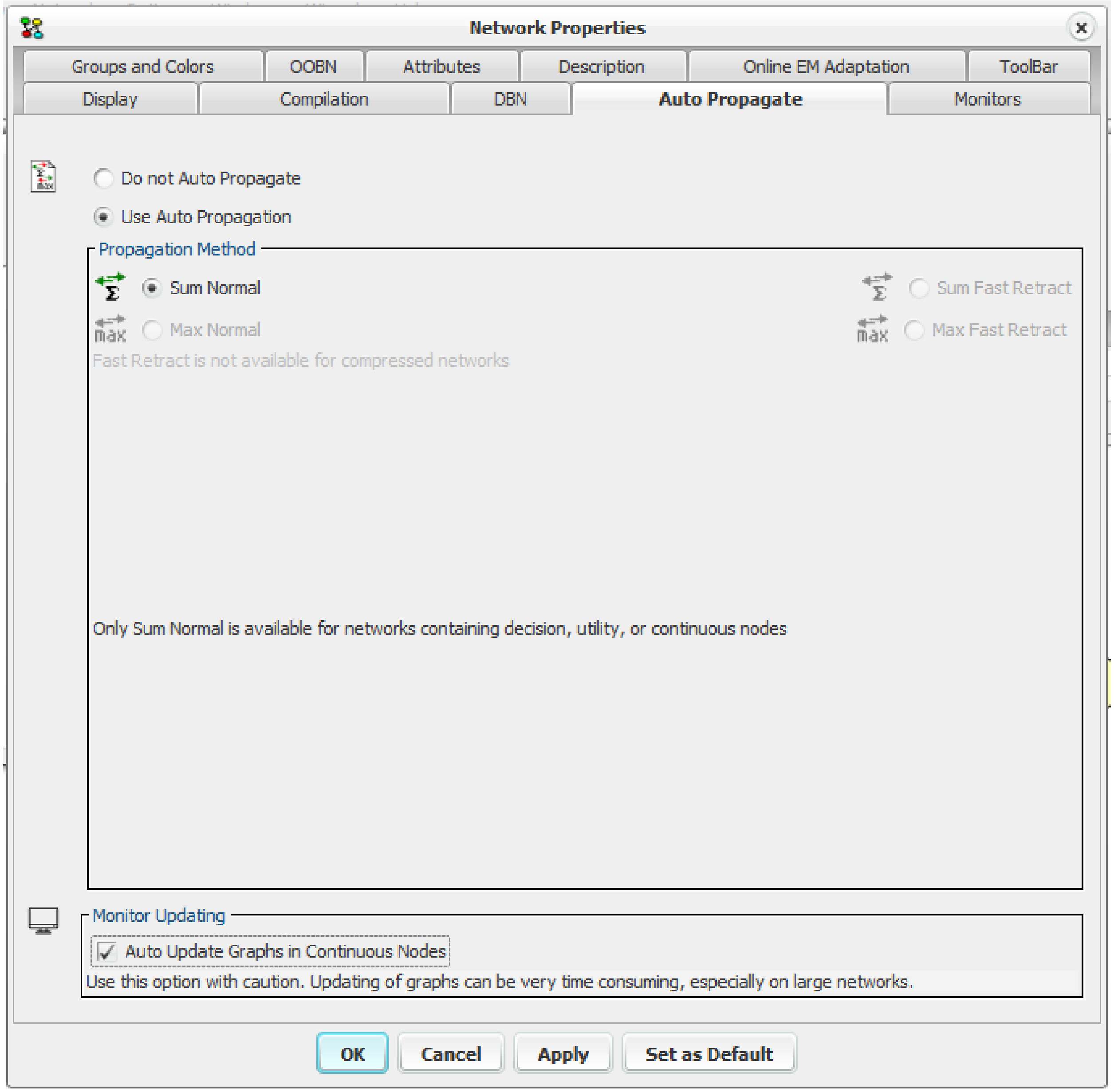
Use this option with caution. Updating of graphs can be very time consuming, especially on large networks.

Figure 1: The Network Properties dialog box showing the Auto Propagate tab.

In the Auto Propagate tab you can select one of two radio buttons to specify if you want to use auto propagation or not. If you choose auto propagation, you can also specify which propagation method you want to use. In most cases you would want to stick with *Sum Normal* (page 250).

4.1.14 Auto Update Graphs

HUGIN can be specified to auto update graphs for *continuous chance nodes* (page 224). This is done in the *Auto Propagate* (page 129) tab of the *Network Properties* (page 178) dialog box (see Figure 1). Simply check the check box in the Monitor Updating group box.



○ Do not Auto Propagate

● Use Auto Propagation

Propagation Method

● Sum Normal

○ Sum Fast Retract

○ Max Normal

○ Max Fast Retract

Fast Retract is not available for compressed networks

Only Sum Normal is available for networks containing decision, utility, or continuous nodes

Monitor Updating

☑ Auto Update Graphs in Continuous Nodes

Use this option with caution. Updating of graphs can be very time consuming, especially on large networks.

Figure 1: The Network Properties dialog box showing the Auto Propagate tab.

If auto update graph is enabled, the graphs in the *monitor windows* (page 194) of continuous chance nodes will automatically be updated when new evidence is entered. This can be very helpful on small networks but is not recommended larger networks because it can be rather time consuming.

4.1.15 Temporal Clone

When modeling a system that evolves over time (as a *DBN* (page 350)), random variables at a given time instant typically depend on the state of the system at past time instants.

In order to specify time dependencies, temporal clones can be constructed for the regular nodes.

A temporal clone of a variable X represents X at the previous time instant. Links can then be added from temporal clones to the regular nodes. Thus, the temporal clones represent the "interface" between two successive time slices.

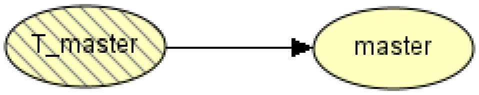


Figure 1: Temporal clone T_master and regular node master. T_master represents master from previous time slice.

Create temporal clone

To create a temporal clone, *select* (page 153) the master node that should be cloned, then right-click the master node and click Create temporal clone, finally click the *network panel* (page 177) at the position where the temporal clone should be inserted.

The Temporal Clone

The following list higligts the main charactaristica of temporal clones.

- Both discrete and continuous chance nodes can have temporal clones. In the case of a DBN with mixed nodes, there are restrictions on the compilation of the model.
- The links of the network must respect the natural flow of time. This means that temporal clones can only have other temporal clones as parents (not regular nodes).
- Discrete temporal clones can have tables and models. These tables are used to define the distribution over the temporal clones for the first time slice. For subsequent time slices, the temporal clones are replaced by the actual nodes in the previous time slice. Because links are allowed between temporal clones, any distribution can be specified.
- For discrete temporal clones, the category and attributes related to states cannot be changed directly. Changes must be performed through their temporal masters: Identical changes are then automatically performed for the clones.
- A temporal clone can be deleted. But if the temporal master of the clone is deleted, then the clone is also deleted.

4.1.16 Instance Node Replace Class

The class on an instance node of a Bayesian network or a LIMID model can be replaced by another (compatible) class using the "Instance Tool->Replace Class" under the right mouse button menu, or by selecting a new class in the *Node Properties Pane* (page 212) of the instance node.

4.1.17 Text labels

A text label is a dragable text component that can be used to add custom description to the current network. A text label can be added to the current network by activating the Text Label Tool item of the Edit Menu or Edit Mode Tool Bar, and clicking somewhere in the *Network Pane* (page 177).

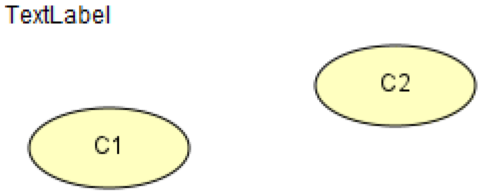


Figure 1: A transparent unselected text label component and two nodes.

A text label is made editable by double clicking on the given label, which also visually changes the background color of the text label to white and enables a cursor within the label (see Figure 2 below). In editable state, standard shortcuts on text such as: Ctrl + a (select all) and Ctrl + x (cut selected) works on the text.

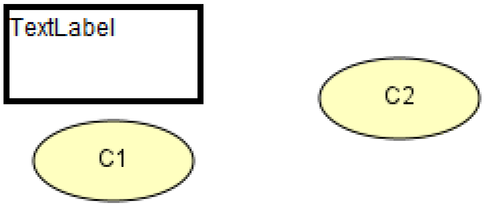


Figure 2: A selected editable text label

The text label is deselected by clicking anywhere outside the label. Also a text label cannot be moved while editable.

You can modify the font, the text color and the background color by right clicking the text label and selecting one of the options from the popup menu.

Text labels supports state-binding for *discrete chance node* (page 223). State binding means linking the belief of node state to a part of the text in the text label. When in *Run Mode* (page 123) the belief value is displayed in place of the binding and updated as the network is propagated. Right click a state label and press the menu item “Syntax Help” for more information.

4.1.18 Select State

The *select state* function is activated by double clicking a state of a node in the *Node List Pane* (page 243) or in a *Monitor Window* (page 194).

The *select state* function is a special case (the most common case) of *entering likelihood evidence* (page 141). What you actually do is entering the fact that one state now has probability 1 and all other states have probability 0.

4.1.19 Select All

The select all function is activated by selecting the Select All item of the *Edit Menu* (page 124).

The select all function selects all nodes in the *Network Pane* (page 177). This allows you to perform certain operations on all nodes in one go instead of performing them on one node at a time.

4.1.20 Belief Precision

The belief precision function is activated by selecting Belief Precision from the *View Menu* (page 188). The Belief Precision item has a series of sub items which are shown in Figure 1.

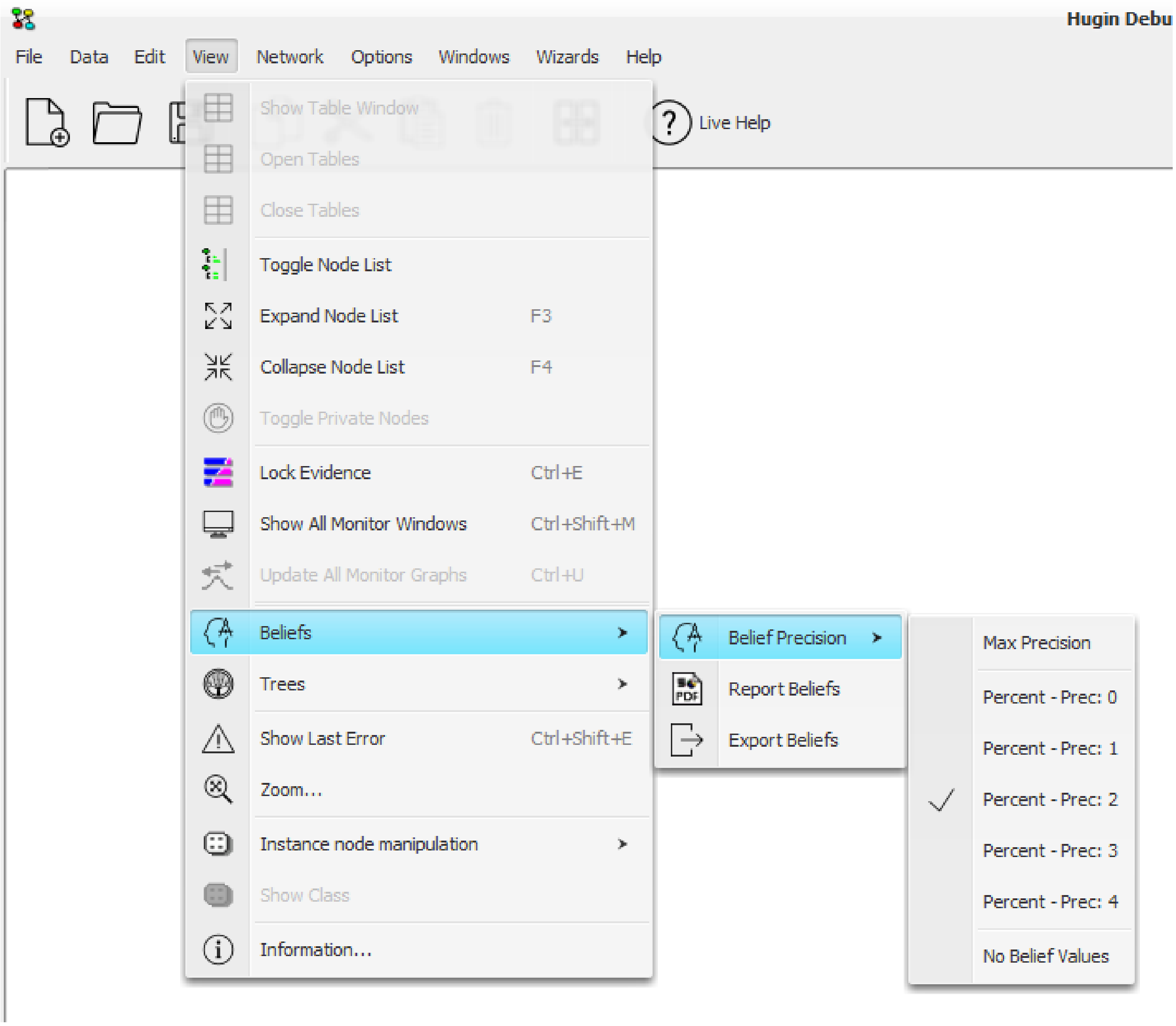


Figure 1: The sub items of Belief Precision in the View Menu.

The first sub item is Max Precision which when selected displays all beliefs as numbers between 0 and 1 with up to six significant digits (single precision values). In influence diagrams, expected utilities are displayed also with up to six significant digits.

The five “Percent” sub items enables display of beliefs as numbers between 0 and 100 (percent). Each of these sub items specifies a specific number of digits following the decimal point.

The last sub item is “No Belief Values”. This disables the display of belief values. In stead, only the graphical display of beliefs is used.

4.1.21 Constraints

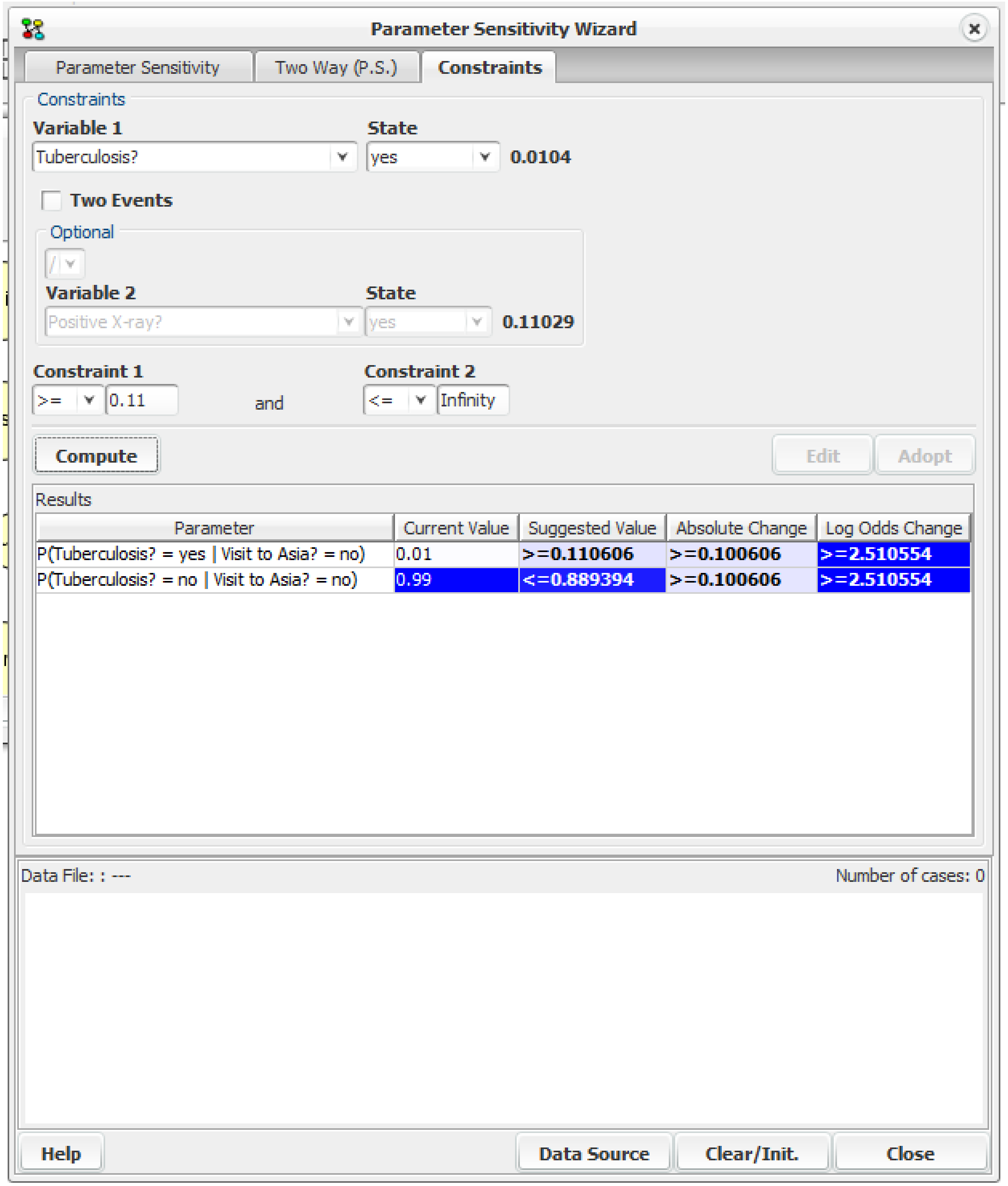
We consider three different types of query constraints; a single event constraint $P(y_1|\varepsilon)$, a two events difference constraint $P(y_1|\varepsilon)$) - $P(y_2|\varepsilon)$ and a two events ratio constraint $P(y1|\varepsilon)$ / $P(y_2|\varepsilon)$. For each type of constraint, we may consider =, <= and >= relations compared to a value x ∈ R. We assume the constants of the sensitivity functions involved have been identified such that: $f(t) = \alpha * t + \beta / \gamma * t + \delta$ and the evidence set ε to be constant.

To identify the required adjustment for the parameter t to satisfy a constraint, we solve three equations with respect to equality:

- Single event constraint : $x = P(y_1|\varepsilon)$.
- Two events difference constraint : $x = P(y_1|\varepsilon) - P(y_2|\varepsilon) = (\alpha_1 * t + \beta_1/\gamma * t + \delta) - (\alpha_2 * t + \beta_2/\gamma * t + \delta)$.
- Two events ratio constraint : $x = P(y_1|\varepsilon)/P(y_2|\varepsilon) = (\alpha_1 * t + \beta_1/\gamma * t + \delta)/(\alpha_2 * t + \beta_2/\gamma * t + \delta)$.

Constraints Panel

Part of the Parameter Sensitivity Wizard is the Query Constraints Panel. The purpose of this panel is to offer the user the ability to specify constraints on the posterior probability in the hypothesis and find the smallest change necessary to satisfy it. In Figure 1 the “asia_.net” network is loaded and a constraint is specified for the posterior probability of “Has tuberculosis”. The current belief for Has tuberculosis = yes is 0.1 and we would like it to be 0.11. After pressing the “Compute” button the tool returns a list of changes that could be made to satisfy the constraint. When selecting one of these suggestions, the buttons “Edit” and “Adopt” become enabled allowing the user to apply the changes.



Constraints

Variable 1: Tuberculosis? State: yes 0.0104

□ Two Events

Optional: /

Variable 2: Positive X-ray? State: yes 0.11029

Constraint 1: >= 0.11 and Constraint 2: <= Infinity

Results

Parameter	Current Value	Suggested Value	Absolute Change	Log Odds Change
P(Tuberculosis? = yes | Visit to Asia? = no)	0.01	>=0.110606	>=0.100606	>=2.510554
P(Tuberculosis? = no | Visit to Asia? = no)	0.99	<=0.889394	>=0.100606	>=2.510554

Data File: : --- Number of cases: 0

Figure 1: Constraints Panel

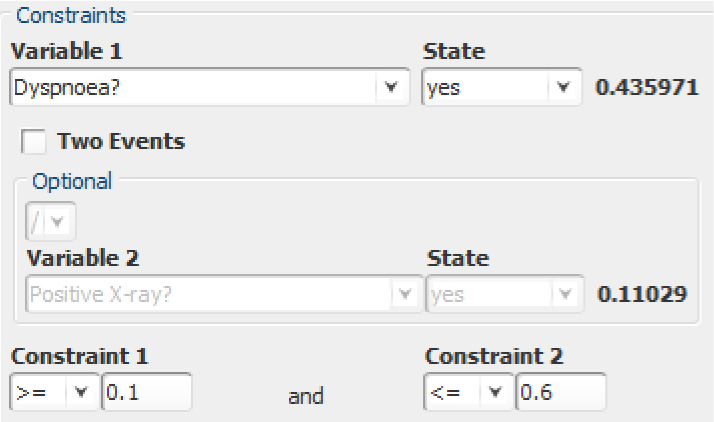
The constraints panel consists of the following elements:

- The *input panel* (page 136) is where the user can select the hypothesis nodes and their states and specify constraints to their posterior probabilities.

- The *results table* (page 137) is a list of suggested changes to values in CPT tables, that the tool has computed and can satisfy the constraints.
- The *cases panel* (page 335) is where the user can load a case file corresponding to the current domain, and select and insert a case in the domain.

Input Panel

There are three ways to specify constraints in the input panel; one event constraint, two event Ratio constraints and two event difference constraints. To specify a one event constraint, select a hypothesis node from the “Variable 1” combo box and then one of its states from the state combo box. The belief for the selected state is shown to the right.



Constraints

Variable 1: Dyspnoea?

State: yes

0.435971

□ Two Events

Optional

/

Variable 2: Positive X-ray?

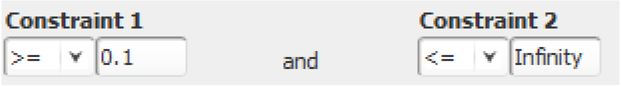
State: yes

0.11029

Constraint 1: >= 0.1 and Constraint 2: <= 0.6

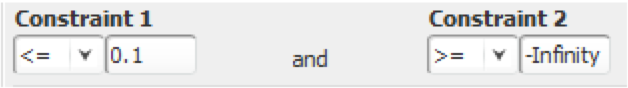
Figure 2: The input panel where one event constraint is specified.

To specify two event constraints the “Two Events” check box should be selected. The optional panel becomes enabled and allows the selection of the type of the two event constraint (ratio or difference) and the selection of the second Hypothesis node (Variable 2). The next step is to input the constraints in the constraint fields (Constraint 1, Constraint2). It is possible to specify one or two constraints (“<= con1 and >= con2”). When specifying one constraint the Constraint 2 field is automatically set to Infinity (Figure 3) or -Infinity (Figure 4) according to comparison symbol selected in the Constraint 1 combo box.



Constraint 1: >= 0.1 and Constraint 2: <= Infinity

Figure 3: One constraint, greater equal



Constraint 1: <= 0.1 and Constraint 2: >= -Infinity

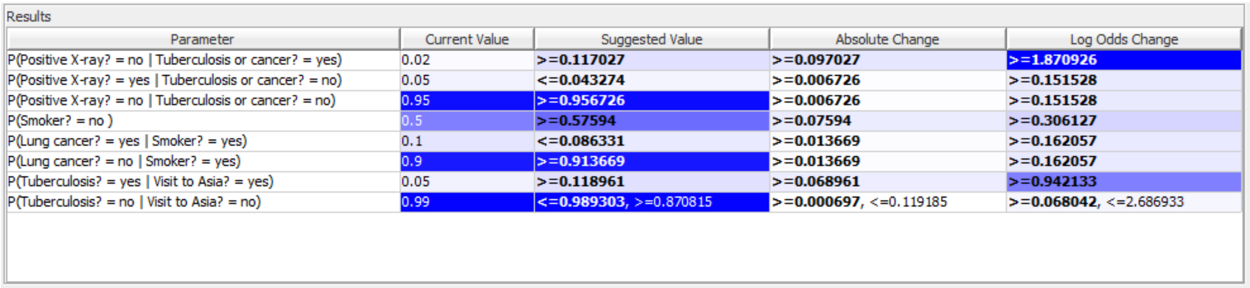
Figure 4: One Constraint, less equal

The computation of the suggested changes that could be made to satisfy the specified constraints, is triggered by pressing the “Compute” button or pressing Enter, while editing in one of the two constraint fields.

Results Table

The results table (Figure 5) displays suggested changes that can be made to certain cpt tables to satisfy the constraints. Each row describes one suggestion with the following columns:

- The Parameter column describes the posterior probability that needs to be changed.
- The current value is the value of the cpt index.
- The suggested value column shows a range of values that can replace the current value in order for the constraints to be satisfied. The value displayed in bold, indicates the smallest absolute change within the suggested values range.
- The absolute change is a range of values that represent the absolute difference between the suggested value and the current value. (Current - suggested). The value displayed in bold represents the absolute change for the suggested value displayed in bold.
- If q is the new value and p is the current value, the Log Odds Change is defined as follows: $log(q/(1-q)) - log(p/(1-p))$. The value displayed in bold, indicates the smallest log odds change within the suggested values range.



Results

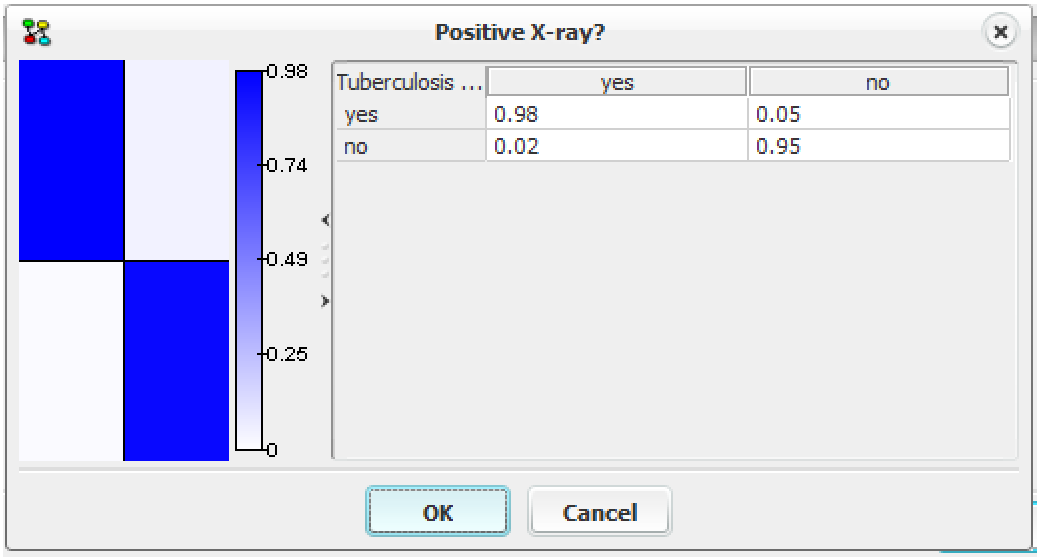
Parameter	Current Value	Suggested Value	Absolute Change	Log Odds Change
P(Positive X-ray? = no | Tuberculosis or cancer? = yes)	0.02	**>=0.117027**	**>=0.097027**	**>=1.870926**
P(Positive X-ray? = yes | Tuberculosis or cancer? = no)	0.05	**<=0.043274**	**>=0.006726**	**>=0.151528**
P(Positive X-ray? = no | Tuberculosis or cancer? = no)	0.95	**>=0.956726**	**>=0.006726**	**>=0.151528**
P(Smoker? = no)	0.5	**>=0.57594**	**>=0.07594**	**>=0.306127**
P(Lung cancer? = yes | Smoker? = yes)	0.1	**<=0.086331**	**>=0.013669**	**>=0.162057**
P(Lung cancer? = no | Smoker? = yes)	0.9	**>=0.913669**	**>=0.013669**	**>=0.162057**
P(Tuberculosis? = yes | Visit to Asia? = yes)	0.05	**>=0.118961**	**>=0.068961**	**>=0.942133**
P(Tuberculosis? = no | Visit to Asia? = no)	0.99	**<=0.989303,** >=0.870815	**>=0.000697,** <=0.119185	**>=0.068042,** <=2.686933

Figure 5: A table showing the suggested changes that could satisfy the constraints

Every column can be sorted by clicking on its header once (ascending order) or twice (descending order). Every cell in a column other than the Parameter column, is colored with a variation of the blue color that represents the actual value. In case of a range of values, the color represents the one displayed in bold. The color variation and the sorting function make it easier to select the suggested change with the smallest absolute or log odds change. When selecting a row, the “Edit” and “Adopt” buttons are enabled (Figure 6).

Figure 6: “Edit” and “Adopt” buttons

When pressing “Edit” a dialog with the selected cpt table is displayed allowing the user to manually apply the change selected in the results table. If the first row in the results table is selected the dialog in Figure 7 is displayed and the user can change the value 0.3 (smoker = no) to a value between 0.39907 and 0.708908.



Tuberculosis ...	yes	no
yes	0.98	0.05
no	0.02	0.95

Figure 7: CPT for “Has bronchitis”

When pressing “Adopt”, the selected change is automatically applied and the value displayed in bold is used as it represents the smallest change. After applying constraints in the domain the effects of the changes made can be seen in the *“Parameter Sensitivity Panel”* (page 335).

4.1.22 Prediction

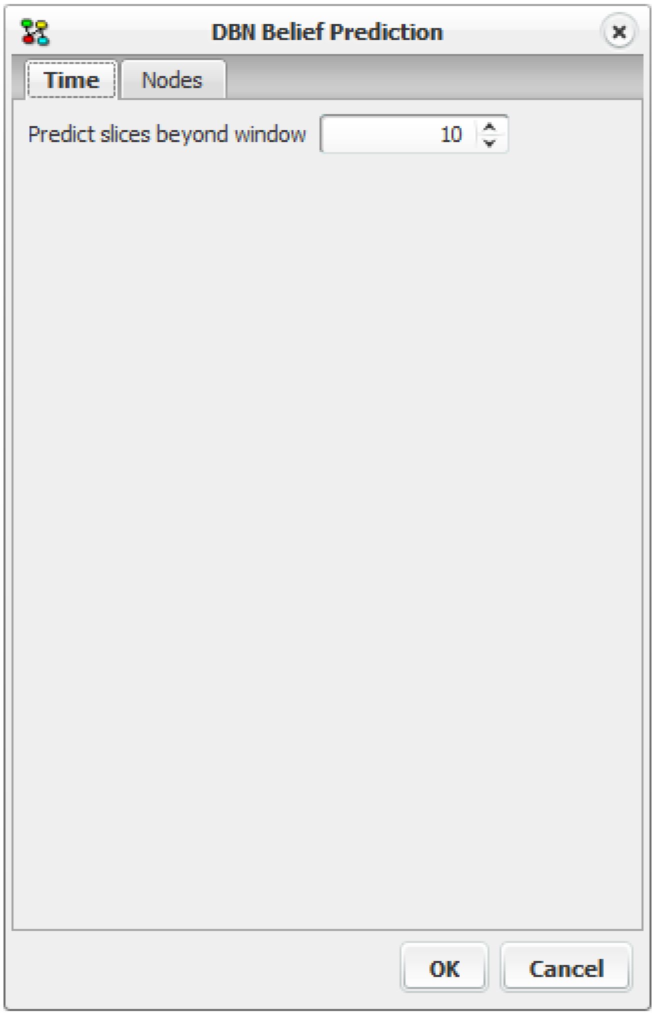
In a *DBN* (page 350) we can compute beliefs for nodes in time slices that lie beyond the time window. This operation is called prediction.

Note: only DBNs entirely consisting of discrete nodes supports prediction.

To perform prediction, click the prediction button and summon the prediction wizard.

Prediction Wizard

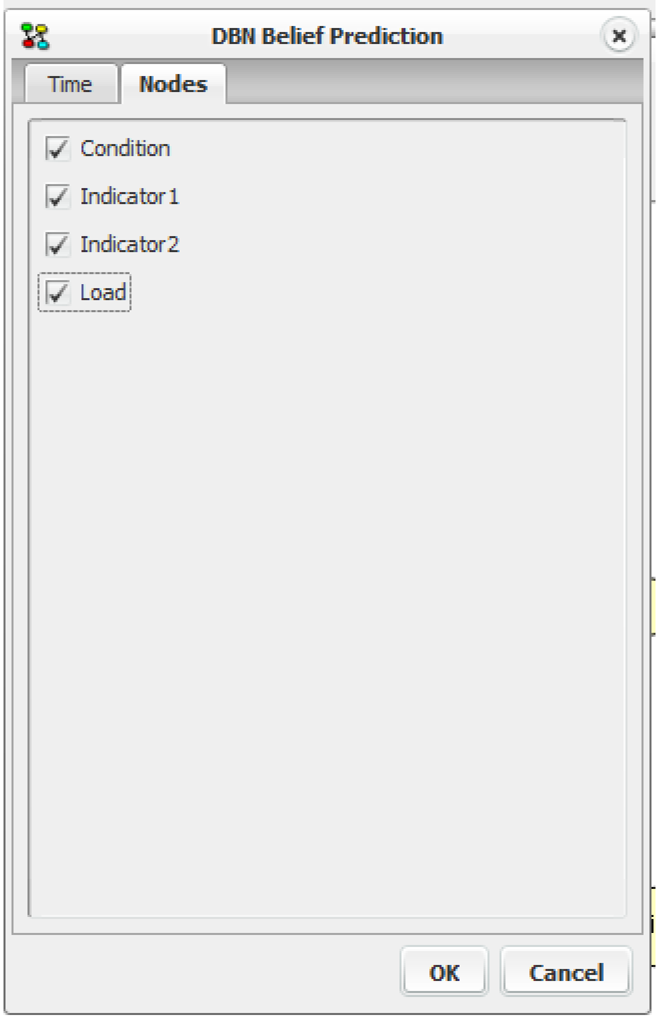
The belief prediction wizard has two tabs for configuring the computations. First configure the number of time slices beyond the current time window to predict.



Predict slices beyond window 10

Figure 1: Prediction, configure number of time slices beyond window to predict.

Next pick the nodes we wish to predict.



☑ Condition

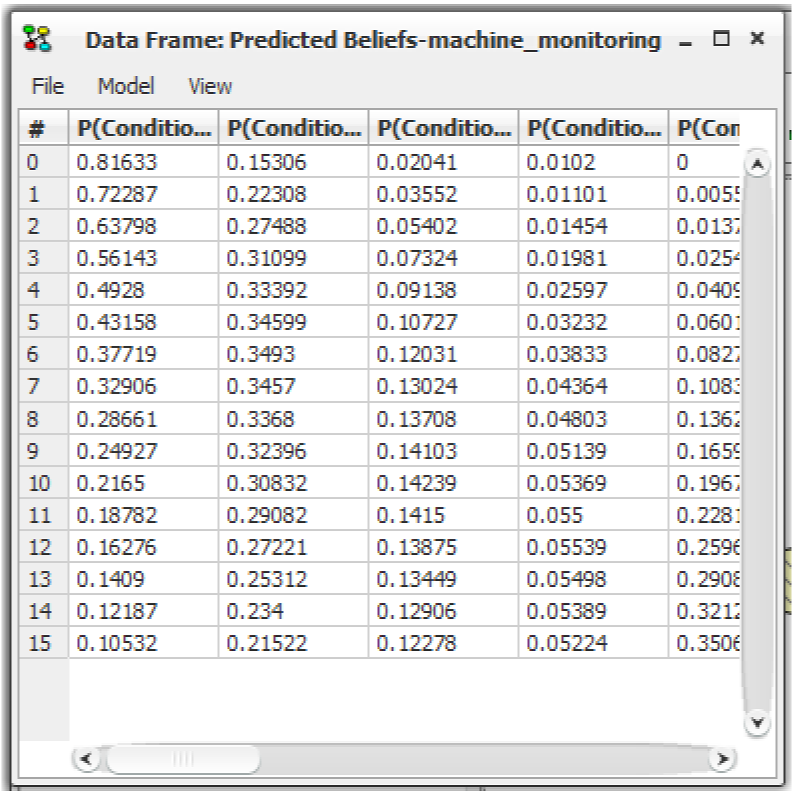
☑ Indicator1

☑ Indicator2

☑ Load

Figure 2: Prediction, pick which nodes to predict.

Finally when done configuring click the OK button to compute predictions. When computation is done a *data frame* (page 391) appears with the predicted beliefs.



#	P(Conditio...	P(Conditio...	P(Conditio...	P(Conditio...	P(Con
0	0.81633	0.15306	0.02041	0.0102	0
1	0.72287	0.22308	0.03552	0.01101	0.0055
2	0.63798	0.27488	0.05402	0.01454	0.0137
3	0.56143	0.31099	0.07324	0.01981	0.0254
4	0.4928	0.33392	0.09138	0.02597	0.0409
5	0.43158	0.34599	0.10727	0.03232	0.0601
6	0.37719	0.3493	0.12031	0.03833	0.0827
7	0.32906	0.3457	0.13024	0.04364	0.1083
8	0.28661	0.3368	0.13708	0.04803	0.1362
9	0.24927	0.32396	0.14103	0.05139	0.1659
10	0.2165	0.30832	0.14239	0.05369	0.1967
11	0.18782	0.29082	0.1415	0.055	0.2281
12	0.16276	0.27221	0.13875	0.05539	0.2596
13	0.1409	0.25312	0.13449	0.05498	0.2908
14	0.12187	0.234	0.12906	0.05389	0.3212
15	0.10532	0.21522	0.12278	0.05224	0.3506

Figure 3: A data frame displaying the predicted beliefs. Each row represents a time slice. The first 0 to (n - number of slices to predict) rows are the time slices from the current window and the remaining rows are the predicted time slices.

4.1.23 Enter Likelihood

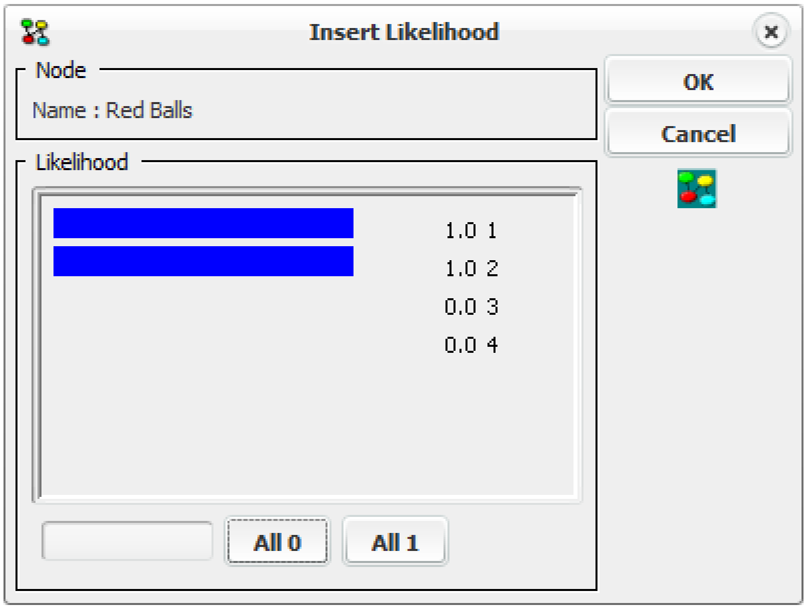
The enter likelihood function is available in *Run Mode* (page 123). You can activate the enter likelihood function either by selecting the Enter Likelihood item of the Network Menu or by pressing the enter likelihood tool button (see Figure 1).

Figure 1: The enter likelihood tool button from the Run Mode Tool Bar.

Entrance of likelihood can only be used on *discrete chance nodes* (page 223).

Entrance of likelihood is what you do when you learn something about the state of the world which can be entered into a node of your network. The simplest form is for example when you have a node “Ball Color” stating the color of a ball drawn from a basket of red and blue balls. This node has states “red” and “blue”. Then, when you see the color of the ball is actually blue, you can enter this fact by saying that “Ball Color” takes state “blue”. The easiest way to enter this is by using the *select state* (page 132) function. However, you can also do it by entering likelihood. That is, you want to enter the fact that the probability of “Ball Color” being in state “blue” is 1 while the probability of “Ball Color” being in state “red” is 0.

When you have activated the enter likelihood function, the dialog window shown in Figure 2 appears. Here, you can enter additional information that you might have come over.



Name : Red Balls

1.0 1
1.0 2
0.0 3
0.0 4

Figure 2: The likelihood dialog window.

In the case of Figure 2, we have a node called “# Red Balls” counting the number of red balls in a basket containing 3 balls of which some are blue and some are red. If you have taken two balls up from the basket and both of these were blue, you know that at most 1 ball in the basket is red. This information can be given to the network by entering likelihood into the “# Red Balls” node as we have done in Figure 2. We give “0 balls” and “1 ball” equal positive probability while “2 balls” and “3 balls” get probability 0.

In the likelihood dialog window, you have a Node area and a Likelihood area. In the Node area, the *label* (page 210) of the node is found. In the Likelihood area, there is a display showing what you have entered so far. Here, you can drag the bars representing a probability back and forward with the mouse. Below this display is a select field where you can choose between different standard probability values (0, 0.1, 0.33333, 0.5, etc.). Choosing one of these values will make the currently selected state in the display take this value as probability.

Below the display in the Likelihood area are also two buttons labeled “All 1” and “All 0”. Pressing one of these makes all states in the display take values 1 or 0 respectively.

You should not be concerned with the fact that the sum of all probabilities are not necessarily 1. HUGIN will normalize the values so that for example in the case shown in Figure 2, “0 Balls” and “1 Ball” each get probability 0.5.

4.1.24 Align

The align function is activated by selecting the Align item from the *Options Menu* (page 193). The Align item has five sub items which are shown in Figure 1.

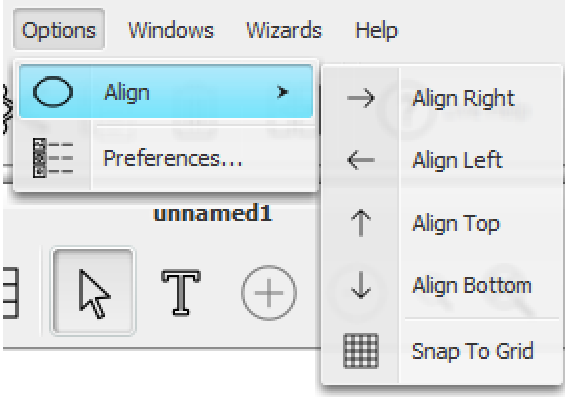


Figure 1: The sub items of Align in the Options Menu.

The four “Align” sub items are used for aligning a series of nodes to the same horizontal or vertical line. The line is specified by “Left”, “Right”, “Top”, or “Bottom”. The “Align” sub items can only be activated if two or more nodes are selected. When an “Align” sub item is activated, all the selected nodes are placed on the specified line. If, for example, the “Align Right” sub item is activated, the specified line is the vertical line through the right most node in the series of selected nodes (“Align Left” specifies a vertical line through the left most selected node, while “Align Top” and “Align Bottom” specifies horizontal lines through the top most and bottom most selected nodes, respectively).

If the specified line is a vertical line the selected nodes keep their horizontal coordinate and if it is a horizontal line they keep their vertical coordinate.

The “Snap to Grid” sub item of the “Align” item moves each node to the grid point closest to its current position. A shortcut to this function is available through Shift + , (comma).

4.1.25 Tile Windows Horizontally

The *tile windows horizontally* function can be activated from the Tile Horizontal item in the *Window Menu* (page 189) or from the tile windows horizontally button in the *Main Window* (page 186) Tool Bar (the button shown in Figure 1).

Figure 1: The tile windows horizontally tool button from the Main Window Tool Bar.

The tile windows horizontally function arranges the network windows in the Main Window below each other.

4.1.26 Tile Windows Vertically

The tile windows vertically function can be activated from the Tile Vertical item in the *Window Menu* (page 189) or from the tile windows vertically button in the *Main Window* (page 186) Tool Bar (the button shown in Figure 1).

Figure 1: The tile windows vertically tool button from the Main Window Tool Bar.

The tile windows vertically function arranges the network windows in the Main Window side by side.

4.1.27 Cascade Windows

The cascade windows function can be activated from the Cascade item in the *Window Menu* (page 189) or from the cascade windows button in the *Main Window* (page 186) Tool Bar (the button shown in Figure 1).

Figure 1: The cascade windows tool button from the Main Window Tool Bar.

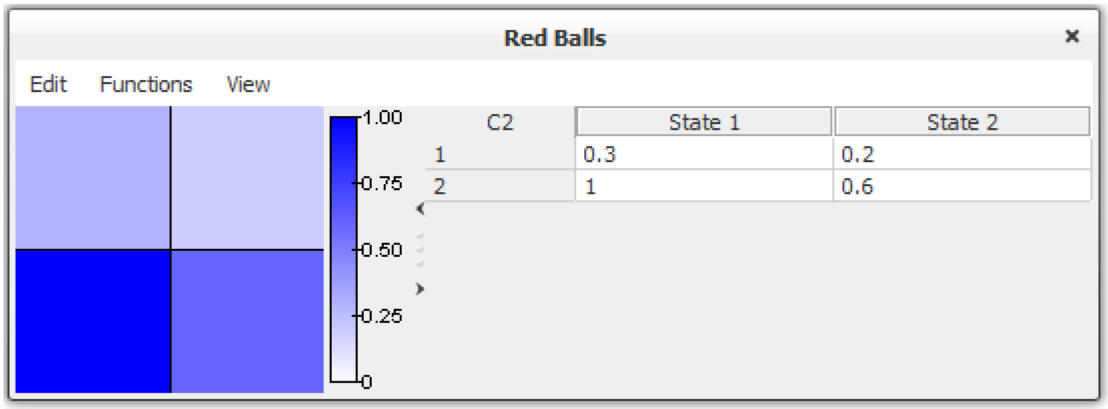
The cascade windows function arranges the network windows in the Main Window in a cascade (window upon window upon window…).

4.1.28 Color Chart

The purpose of the color chart is to graphically display the contents of an entire table at once. This will easily reviel to the user where large values are concentrated. Color charts are used various places in HUGIN for displaying different kinds of information:

- In conjunction with the *table pane* (page 239) for displaying the CPT. (See figure 1.)
- In the *Paramater Sensitivity Analysis wizard* (page 335).
- In the *Correlation analysis tool* (page 329).

Figure 1 shows the Tables Pane for a discrete chance node in conjunction with a color chart.



C2	State 1	State 2
1	0.3	0.2
2	1	0.6

Figure 1: Color chart from the Tables Pane.

4.1.29 Component Tree

The component tree tool gives an overview of any classes instantiated in an *OOBN* (page 346) from a selected network class. Each class is represented by a rectangle in the tree. The component tree tool can be activated from the *View menu* (page 188).

Figure 1 shows a screenshot of the component tree window for a random network. Functionality for printing the tree and zooming is provided by the toolbar in the top of the window. Clicking on a class in the component tree will bring the propper *network pane* (page 177) in focus.

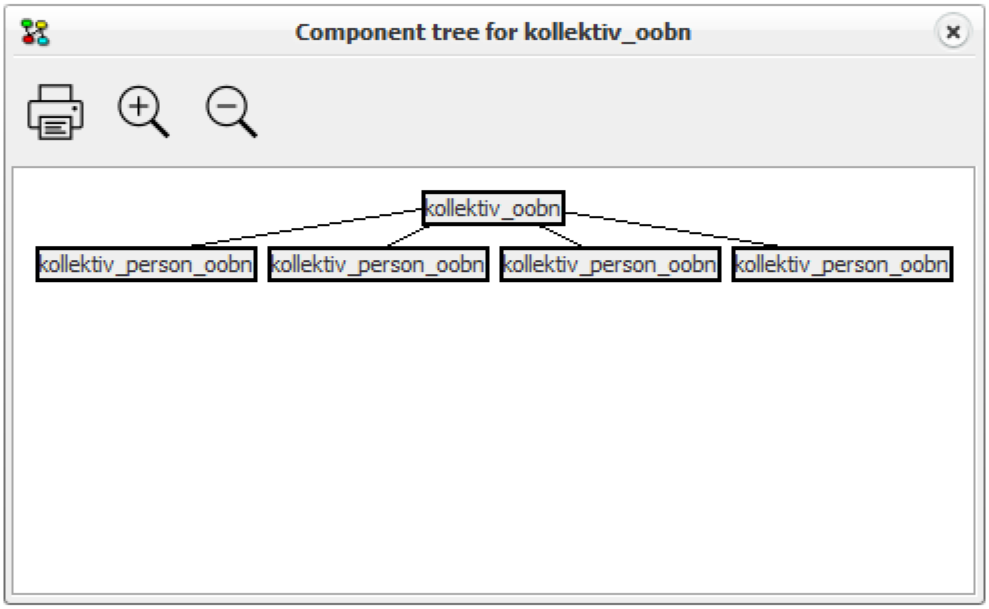


Figure 1: Component tree tool.

4.1.30 Resize Node

It is possible to assign a size to each individual node. This can either be done by setting the size in the *Node Properties* (page 212), or by dragging the "size handles" on the node (displayed in Figure 1). The "size handles" appear when only a single node is selected.

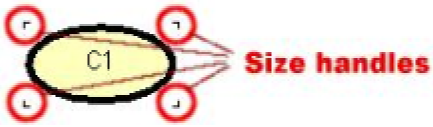


Figure 1: The size handles are placed at the four "corners" of the node

When dragging one of the handles, the node will resize in all four directions. That is, the center of the node remains fixed. Note, that the size of the node cannot be below 10 or above 300 in any direction.

4.1.31 Layout Network Nodes

The *layout network nodes* tool will try to organize the nodes of the network using the algorithm preferred by the user. Layout network nodes can be accessed by selecting the corresponding item of the *Network Menu* (page 185).

The internal algorithm of the HUGIN Graphical User Interface assumes the network as a certain type of structure. This implies that if a network does not have the assumed structure, then the algorithm may produce an organization of the nodes, which seems suboptimal.

If the user has install Dot (Graphviz - open source graph visualization software) on her computer, then she may configure the HUGIN Graphical User Interface to use Dot instead of the internal algorithm. The automatic layout algorithm is configured in Network *Preferences* (page 200).

4.1.32 Lock Display of Evidence

The *lock display of evidence* function can be activated either from the Lock Evidence item of the *View Menu* (page 188) or from the lock evidence button in the *Run Mode* (page 123) Tool Bar (the button shown in Figure 1).

Figure 1: The lock evidence button from the Run Mode Tool Bar.

The lock display of evidence function is meant to allow display of the likelihood entered. Normally (when the lock display of evidence is disabled), the *monitors* (page 194) and the *Node List Pane* (page 243) will for all nodes show the belief of a node being in each of its states. When the lock display of evidence function is activated, the monitors and Node List Pane will show the entered likelihood for all nodes with *likelihood evidence entered* (page 141).

4.1.33 Retract Evidence

The *retract evidence* function can be activated either by selecting the Retract Evidence item of the *Network Menu* (page 185) or by double clicking in the *Node List Pane* (page 243) at the currently selected state of the node for which you want evidence retracted.

You can retract evidence from a node when you want to eliminate the effect of the evidence entered into this node.

4.1.34 Show Class

The Show Class function can be activated by selecting the Show Class item of the *View Menu* (page 188). This function selects the *Network Window* (page 182) containing the network (class) of which the selected *instance node* (page 224) represents an instance (the function is available only when an instance node is the only node selected).

As a shortcut, the class of an instance node can be selected by clicking the left mouse button inside the instance node while pressing the Alt key.

4.1.35 Link

A *link* in a *Bayesian network* (page 345) defines a *causal link* (page 148) from one chance node (*discrete* (page 223) or *continuous* (page 224)) to another.

In an *LIMID* (page 345), a link can mean one of three things:

1. It can be a causal link as in a pure Bayesian network. This is the case, when the link goes from a chance node or *decision node* (page 224) to a chance node.
2. It can define the utility function to depend on a chance node or a decision node. This is the case if the link goes from a chance node or decision node to a *utility node* (page 226).
3. It can specify the information available to the decision maker. This is the case if the link goes from a chance node or decision node to a decision node. These links are referred to as *informational links*.

In an object-oriented Bayesian network (or LIMID), a link is defined as above, except if the link points to an *input node* (page 227) of an *instance node* (page 224). In that case the link is interpreted as a *binding link* (page 149) (i.e., the input node is bound to the node at the other end of the link).

A link can be established in *Edit Mode* (page 124) by activating the *Link Tool* (page 155) and dragging a line from one *node* (page 209) to another.

There are three different display modes for a link:

1. *Normal*: A straight line from one node to another.
2. *Spline*: A bezier line (i.e., a curve that passes over every 3 points of a series).
3. *Lines*: A series of straight lines.

The display mode for a link is selected by first selecting the link (i.e., left-click at or near the link so it gets highlighted), then right-clicking at the background of the *Network Pane* (page 177) to activate the pop-up menu for the pane, then selecting the Link Mode item, and finally selecting the desired mode from the Link Mode sub-menu (see Figure 1).

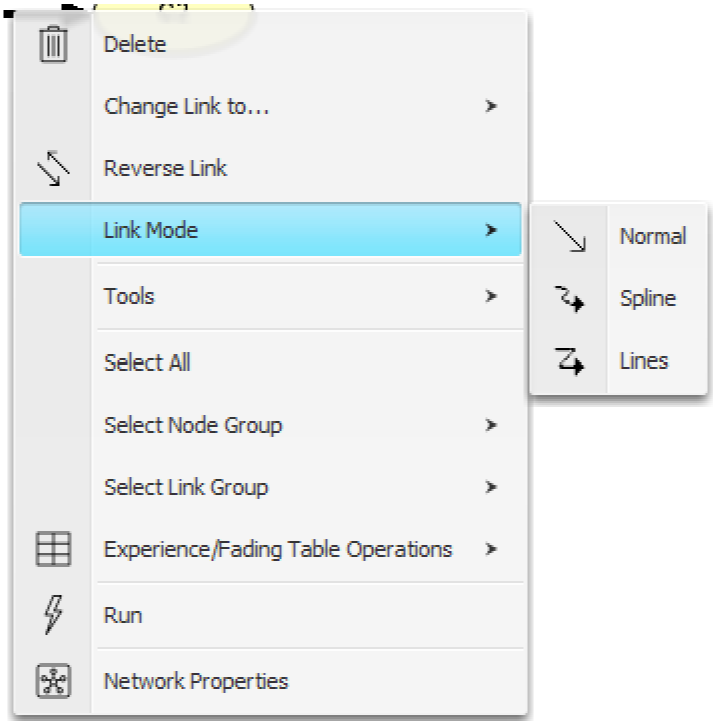


Figure 1: There are three different display modes for a link.

New bezier points a created simply by click-and-drag operations. Figure 2 shows a link with two bezier points, displayed in Spline mode and Lines mode, respectively. The bezier points can be moved using click-and-drag operations. Please note that if a bezier point is moved to a location lying on the line between its neighboring points, it will be removed.

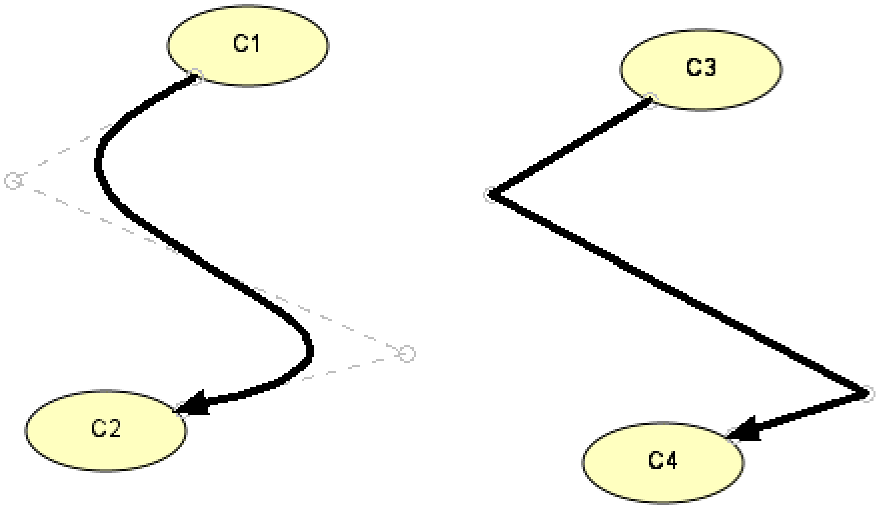


Figure 2: There are three different display modes for a link.

4.1.36 Causal Link

A causal link is a *link* (page 147) from a chance node (*discrete* (page 223) or *continuous* (page 224)) or a decision node to a chance node.

It defines a causal dependence from the parent node to the child node. That is, the state of the parent has an impact on the state of the child.

It can sometimes be difficult to determine what is a causal dependence and what is not. For ecample, imagine you have two nodes: One for “Red Spots” and one for “Measles”. Some might say that when you observe red spots on a child it is very likely that the child has measles, so you should add a link from “Red Spots” to “Measles”. This, however, is terribly WRONG! The relation between red spots and measles is that measles causes red spots. The link should go from “Measles” to “Red Spots” (because it is a causal link).

Always make sure that the causal links you add are causal.

4.1.37 Binding Link

A binding link indicates that the node at which the link starts is bound to the input node to which the link points. For further details, see the description of *input nodes* (page 227).

4.1.38 Link (or Arc) Reversal

A link in a Bayesian network or a LIMID can be reversed using the *Reverse Link Tool* (page 156).

Please note that, to respect the conditional independence and dependence statements encoded in the network, link reversal may cause new links to be added. For example, assume we wish to reverse the link B → A in B → A ← C, which states that B and C are dependent given A. Then, to preserve this dependence statement, a link must be introduced between B and C. More precisely, the rule for link (or arc) reversal is the following:

1. Let X → Y be a link in a Bayesian network, where X has parents $\{XP_1,\ldots,XP_n\}$ and Y has parents $\{X,YP_2,\ldots,YP_m\}$.
2. Reversing the link X → Y, makes $\{XP_1,\ldots,XP_n\} \cup \{Y,YP_2,\ldots,YP_m\}$ be the new parents of X and $\{YP_2,\ldots,YPm\} \cup \{XP_1,\ldots,XP_n\}$ the new parents of Y.

This rule can easily be derived from Bayes' rule.

As an example, consider the network in Figure 1. When reversing the link between X and Y in this network, X get additional parents {C,D} (i.e., those parents of Y that are not already parents of X) and Y get additional parents {A} (i.e., those parents of X that are not already parents of Y). The resulting network is shown in Figure 2.

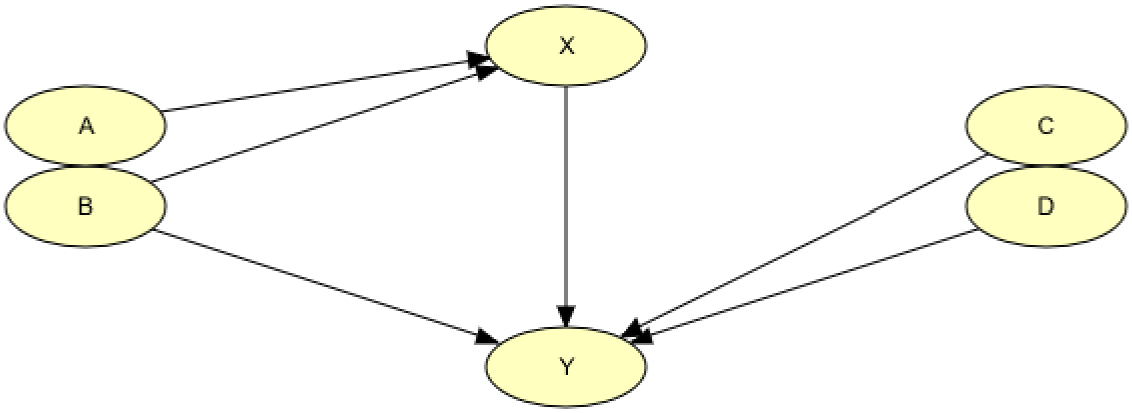


Figure 1: A sample network, with link X → Y selected.

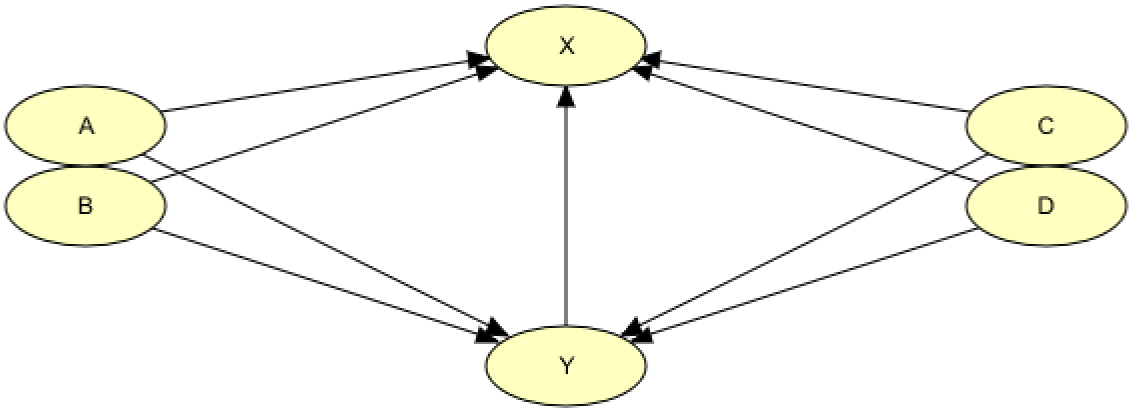


Figure 2: The resulting network after reversal of link X → Y.

Please note that only links between discrete chance nodes can be reversed.

4.1.39 Toggle Node List

The *Toggle Node List* function can be activated by selecting the “Toggle Node List” item of the *View Menu* (page 188). It toggles the display of the *Node List* (page 243) when the network is in *Run Mode* (page 123).

The preferred state of the Node List (hidden or shown) can also be set in the *Preferences* (page 200).

4.1.40 Expressions

Expressions is the key building block of *Models* (page 242) and can be used to simplify the specification of conditional probability tables for discrete chance nodes, utility tables for utility nodes, and initial policies for decision nodes. This is particularly useful when the conditional probability distribution for a variable follows (at least approximately) certain functional or distributional forms. In such cases it is cumbersome to specify the conditional probability table manually.

An expression is built using standard statistical distributions (e.g., Normal, Binomial, Beta, Gamma, etc.), arithmetic operators, standard mathematical functions (e.g., logarithmic, exponential, trigonometric, and hyperbolic functions), logical operators (e.g., and, or, if-then-else), and relations (e.g., less-than, equals).

Expressions can be constructed manually or by the assistance of the *Expression Builder* (page 274), which guides the user through the construction, using series of dialog boxes. See the *Table Generator Tutorial* (page 73) for more information about expressions.

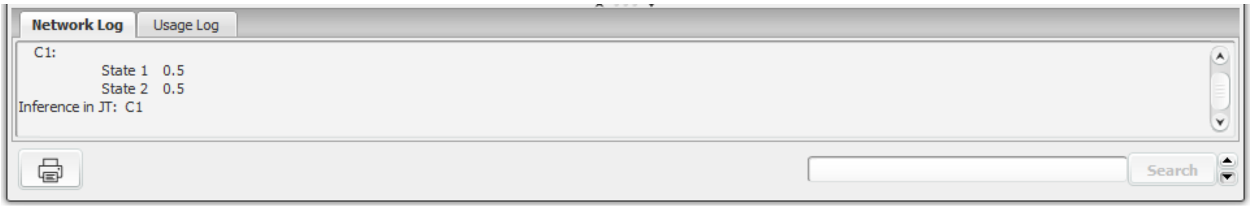
4.1.41 Show / Hide Node Locator

The Show / Hide Node Locator function is available in *Run Mode* (page 123). It toggles the appearance of the *Node Locator* (page 214) in the *Network Window* (page 185) toolbar.

Figure 1: Show/hide node locator button as seen in the run mode toolbar.

4.1.42 Information

The *information* function is activated by selecting the Information item of the *View Menu* (page 188). When the information function is activated, a pane displaying the *Network log* and Usage log appears at the bottom of the screen. The tab Network log shows output comments generated during compilation. The tab *Usage log* shows all actions performed since the network was loaded. Figure 1 shows this pane.



```
  C1:
            State 1   0.5
            State 2   0.5
Inference in JT:  C1
```

Figure 1: The Network Log tab of the Information pane.

4.1.43 Initialize

The initialize function is activated either by selecting the Initialize item of the *Network Menu* (page 185) or by pressing the Initialize Tool button in the *Run Mode* (page 123) Tool Bar (see Figure 1).

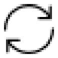


Figure 1: The Initialize Tool button in the Run Mode Tool Bar.

The initialize function retracts all evidence entered into the current network.

4.1.44 Update global node and link groups

The Update global node and link groups function is activated by selecting the Update global node and link groups item of the *Network Menu* (page 185), when a network is in *Edit Mode* (page 124).

The Update global node and link groups function works by updating the user created node- and link groups globally for all classes in the network, such that each class will consistently contain the same groups. This has the side-effect that when the network is later saved in the .net format, the node- and link groups of the top class will be consistent with the groups of all other classes.

4.1.45 Update Monitor Graphs

One can update monitor graphs by selecting the Update all Monitor Graphs item of the *View Menu* (page 188), by clicking the Update Monitor Graph Button in the *Run Mode* (page 123) Tool Bar, or by clicking the update button in the lower right corner of each *Monitor Window* (page 194) showing a graph (*continuous chance nodes* (page 224) only). An update graph button is shown in figure 1.

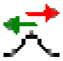


Figure 1: An update monitor graph button.

One may wish to update the monitor graphs when evidence has been propagated in a network containing continuous chance nodes. One can specify HUGIN to *automatically update the graphs* (page 130), but in general (at least for large networks) this feature should be disabled because it can be very time consuming.

4.1.46 Toolbar

The tool bar can be customized to include a button for the CDVT function (see Figure 1).

Figure 1: The Toolbar tab.

Back

4.1.47 Function Tool

The Function Tool can be activated either from the Function Tool item of the *Edit Menu* (page 192) or from the function tool button in the *Edit Mode* (page 124) Tool Bar (the button shown in Figure 1).

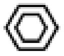


Figure 1: The Function Tool button from the Edit Mode Tool Bar.

When the Function Tool is activated (when the function tool button is pressed) you can place a *function node* (page 225) in the *Network Pane* (page 177): Simply click the mouse somewhere within the Network Pane. When you have placed the function node the *select tool* (page 153) is normally activated again. If you want keep the Function Tool activated (to create a series of nodes) hold down the SHIFT key while placing a node.

4.1.48 Instance Tool

The *Instance Tool* can be activated either from the Instance Tool item of the *Edit Menu* (page 192) or from the Instance Tool Button in the *Edit Mode* (page 124) Tool Bar (the button shown in Figure 1).

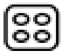


Figure 1: The Instance Tool Button from the *Edit Mode* (page 124) Tool Bar.

When the Instance Tool is activated (i.e., when the Instance Tool Button is pressed), a menu appears with the list of networks that can be instantiated in the currently selected network. (Note that only networks loaded into the HUGIN Graphical User Interface tool can appear in the list.) However, if there is only one network that can be instantiated, no menu will appear. If you select one of the networks in the menu, you can now place an *Instance node* (page 224) in the *Network Pane* (page 177): Simply click the left mouse button somewhere within the Network Pane. When you have placed the instance node, the *select tool* (page 153) is normally activated again. However, if you want to keep the Instance Tool activated (to create a series of nodes) hold down the SHIFT key while placing a node.

4.1.49 Select Tool

The select tool is activated either by selecting the Select Tool item of the *Edit Menu* (page 192) or by clicking the select tool button in the Tool Bar of the *Edit Mode* (page 124) *Network Window* (page 182). In *Run Mode* (page 123), the select tool is always active. Figure 1 shows the select tool button.

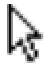


Figure 1: The select tool button from the Edit Mode Tool Bar.

The select tool allows you to select one or more nodes in the *Network Pane* (page 177). You select a single node by clicking it or dragging a rectangle around it when the select tool is active. You can select a series of nodes by dragging a rectangle around the nodes you want selected. You can also add more nodes to the currently selected series of nodes by holding down the SHIFT key while selecting more nodes.

4.1.50 Continuous Chance Tool

The Continuous Chance Tool can be activated either from the Continuous Chance Tool item of the *Edit Menu* (page 192) or from the continuous chance tool button in the *Edit Mode* (page 124) Tool Bar (the button shown in Figure 1).

Figure 1: The Continuous Chance Tool button from the *Edit Mode* (page 124) Tool Bar.

When the Continuous Chance Tool is activated (when the continuous chance tool button is pressed) you can place a *continuous chance node* (page 224) in the *Network Pane* (page 177): Simply click the mouse somewhere within the Network Pane. When you have placed the continuous chance node the *select tool* (page 153) is normally activated again. If you want to keep the Continuous Chance Tool activated (to create a series of nodes) hold down the SHIFT key while placing a node.

4.1.51 Discrete Chance Tool

The *Discrete Chance Tool* can be activated either from the Discrete Chance Tool item of the *Edit Menu* (page 192) or from the discrete chance tool button in the *Edit Mode* (page 124) Tool Bar (the button shown in Figure 1).

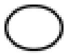


Figure 1: The Discrete Chance Tool button from the Edit Mode Tool Bar.

When the Discrete Chance Tool is activated (when the discrete chance tool button is pressed) you can place a *discrete chance node* (page 223) in the *Network Pane* (page 177): Simply click the mouse somewhere within the Network Pane. When you have placed the discrete chance node the *select tool* (page 153) is normally activated again. If you want keep the Discrete Chance Tool activated (to create a series of nodes) hold down the SHIFT key while placing a node.

4.1.52 Decision Tool

The Decision Tool can be activated either from the Decision Tool item of the *Edit Menu* (page 192) or from the Decision Tool button in the *Edit Mode* (page 124) Tool Bar (the button shown in Figure 1).

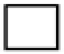


Figure 1: The Decision Tool button from the Edit Mode Tool Bar.

When the Decision Tool is activated (when the Decision Tool button is pressed) you can place a *decision node* (page 224) in the *Network Pane* (page 177): Simply click the mouse somewhere within the Network Pane. When you have placed the decision node, the *select tool* (page 153) is normally activated again. If you want to keep the Decision Tool activated (to create a series of nodes) hold down the SHIFT key while placing a node.

4.1.53 Discrete Function Tool

The *Discrete Function Tool* can be activated either from the Discrete Function Tool item of the *Edit Menu* (page 192) or from the discrete function tool button in the *Edit Mode* (page 124) Tool Bar (the button shown in Figure 1).

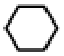


Figure 1: The Discrete Function Tool button from the Edit Mode Tool Bar.

When the Discrete Function Tool is activated (when the discrete function tool button is pressed) you can place a *discrete function node* (page 229) in the *Network Pane* (page 177): Simply click the mouse somewhere within the Network Pane. When you have placed the discrete function node the *select tool* (page 153) is normally activated again. If you want keep the Discrete Function Tool activated (to create a series of nodes) hold down the SHIFT key while placing a node.

4.1.54 Utility Tool

The *Utility Tool* can be activated either from the Utility Tool item of the *Edit Menu* (page 192) or from the utility tool button in the *Edit Mode* (page 124)

Tool Bar (the button shown in Figure 1).

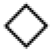


Figure 1: The Utility Tool button from the Edit Mode Tool Bar.

When the Utility Tool is activated (when the Utility Tool button is pressed) you can place a *utility node* (page 226) in the *Network Pane* (page 177): Simply click the mouse somewhere within the Network Pane. When you have placed the utility node, the *select tool* (page 153) is normally activated again. If you want keep the Utility Tool activated (to create a series of nodes) hold down the SHIFT key while placing a node.

4.1.55 Change Link Tool

The Change Link Tool can be used to replace a parent *node* (page 209) in a *link* (page 147) with another node. The old and new parent nodes must be compatible. That is, they must be of the same class, and have an identical state set. During this operation the *CPT* (page 241) of the child node remains unchanged.

The Change Link Tool is activated by first selecting the *link* (page 147) (i.e., left-click at or near the link so it gets highlighted), then right-clicking at the background of the *Network Pane* (page 177) to activate the pop-up menu for the pane, then selecting the "Change Link to…" item. The parent node can be switched to any of the compatible nodes on the list (see Figure 1).

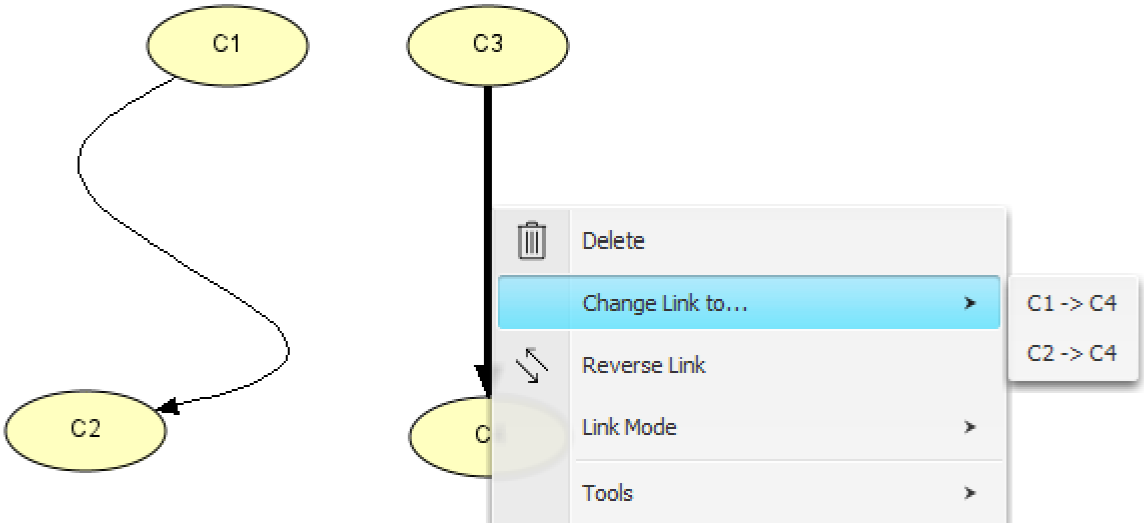


Figure 1: Right click on a link and select the item "Change Link to…" will yield a list of compatible nodes that may replace the current parent node in the link.

4.1.56 Link Tool

The *Link Tool* can be activated either from the Link Tool item of the *Edit Menu* (page 192) or from the Link Tool button in the *Edit Mode* (page 124) Tool Bar (the button shown in Figure 1).

Figure 1: The Link Tool button from the Edit Mode Tool Bar.

When the Link Tool is activated (when the Link Tool button is pressed) you can drag a *link* (page 147) from one node to another in the *Network Pane* (page 177): Press the left mouse button somewhere within one node and drag the cursor into another node. Then, release the mouse button again. When you have created the link, the *select tool* (page 153) is normally activated again. If you want to keep the Link Tool activated (to create a series of links) hold down the SHIFT key while creating the link.

4.1.57 Reverse Link Tool

The Reverse Link Tool can be activated either from the Reverse Link item of the *Edit Menu* (page 192) or from the Reverse Link Tool button in the *Edit Mode* (page 124) Tool Bar (the button shown in Figure 1).

Figure 1: The Reverse Link button from the Edit Mode Tool Bar.

The Reverse Link functionality is activated only when a single link has been selected (click the left mouse button at or near a link).

See *Link Reversal* (page 149) for more information.

4.1.58 Tables Pane Button

The Tables Pane can be activated from the Tables Pane Button in the Edit Mode Tool Bar (the button shown in Figure 1).

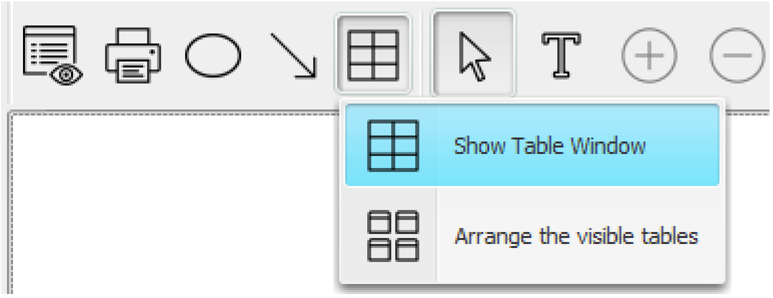


Figure 1: The Tables Pane Button from the Edit Mode Tool Bar.

4.1.59 Adaptation Button

Adaptation (Sequential Learning) can be activated through the Adaptation item of the Network menu (shown in Figure 1).

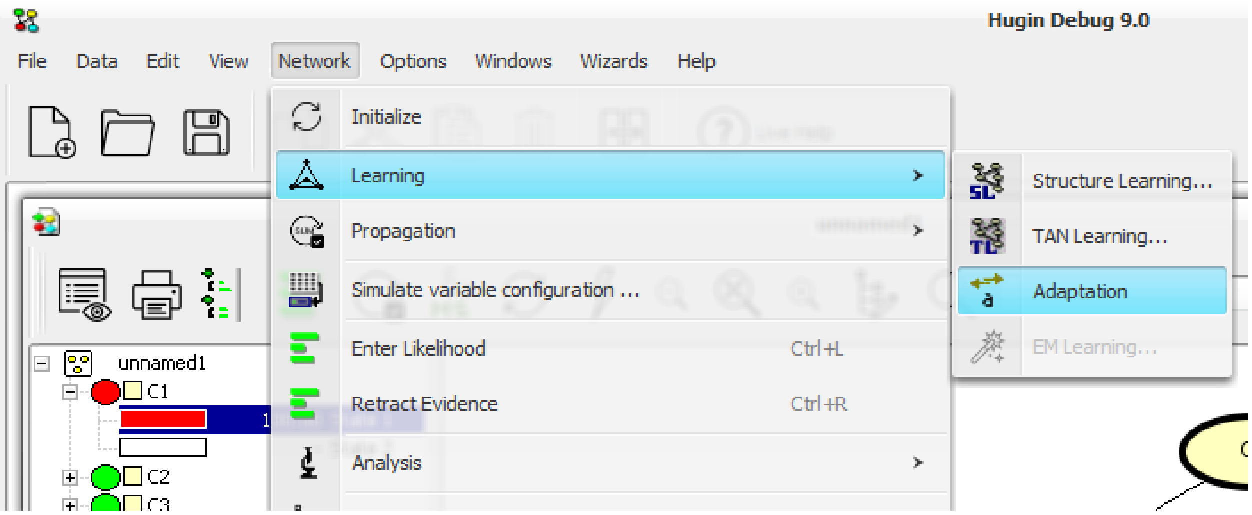


Figure 1: The Adaptation Menu Item.

4.1.60 Case Generator Button

The *Case Generator* (page 391) can be activated through the Simulate Cases item of the *File Menu* (page 191).

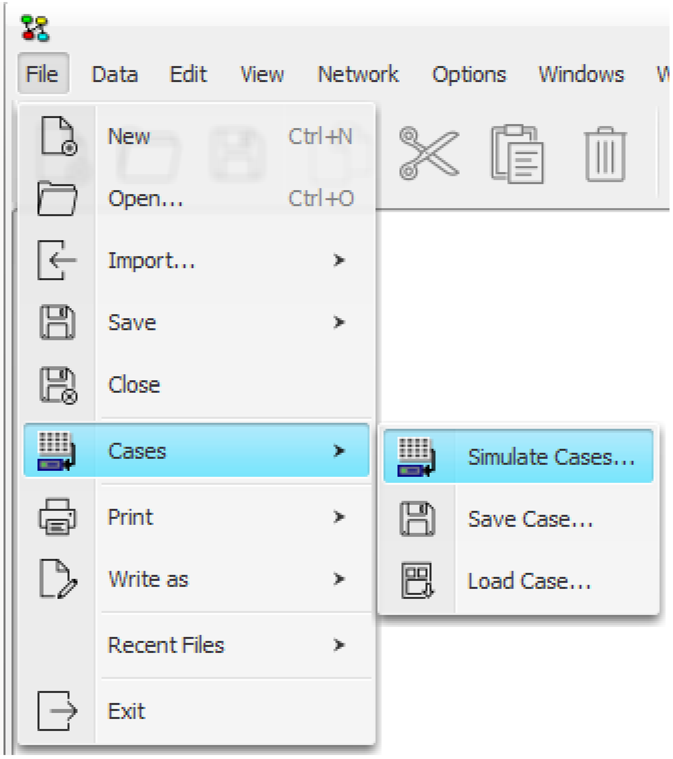


Figure 1: The Case Generator Menu Item.

4.1.61 Save As

The save as function is activated by selecting the Save As item of the *File Menu* (page 191).

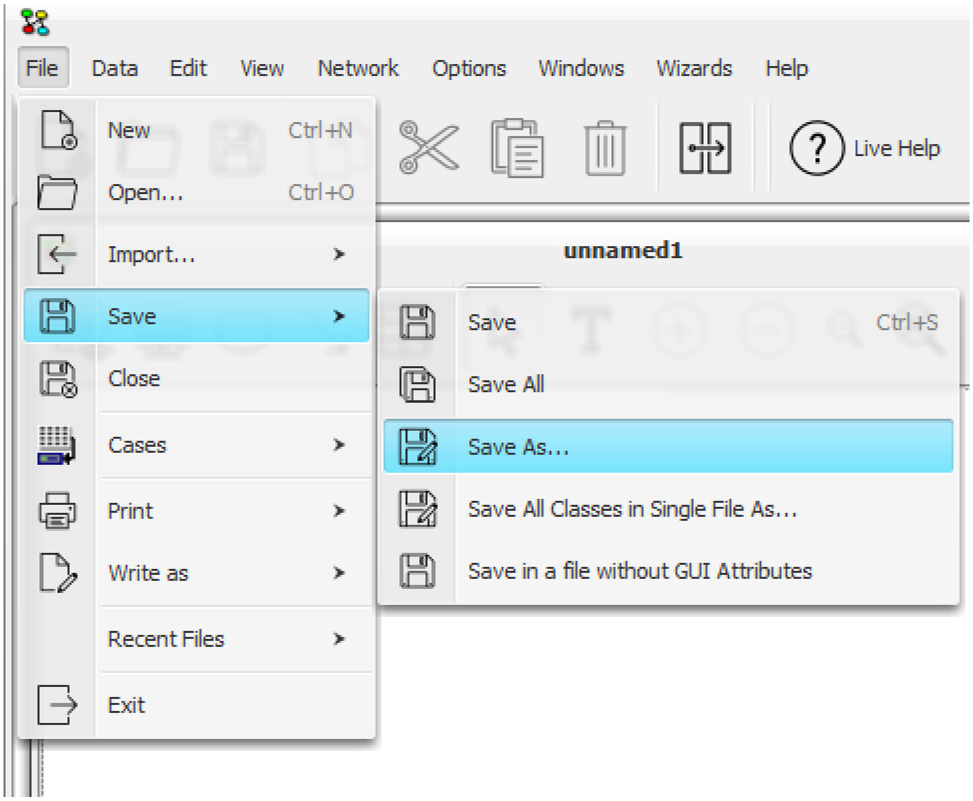


Figure 1: The Save As Menu Item.

The save as function is similar to the *Save* (page 159) function except that it always asks for a name of a file in which to save the network (the save function just saves the network in the file from which the network was loaded or previously saved). Furthermore, the save as function allows for changing the format of the saved file. Possible formats are:

- **.net**: Save the network in 6.0 net-format (OOBN)
- **.net(5.7)**: Save the network in 5.7 net-format
- **.hkb(5.7)**: Save the network as a Knowledge Base
- **password protected hkb**: Save the network as a Knowledge Base, and protect the file with a password.

4.1.62 Save

The save function is activated either by selecting the Save item of the *File Menu* (page 191) or by clicking the save button in the *Main Window* (page 186) Tool Bar. Figure 1 shows the save button.

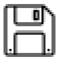


Figure 1: The save button from the Main Window Tool Bar.

When the save function is activated the user is able to save the currently selected network in the HUGIN network format through a standard save dialog.

If you want to save your network in a new file you should use the *Save As* (page 158) function.

4.1.63 Save All

The save all function is activated by selecting the Save All item of the *File Menu* (page 191).

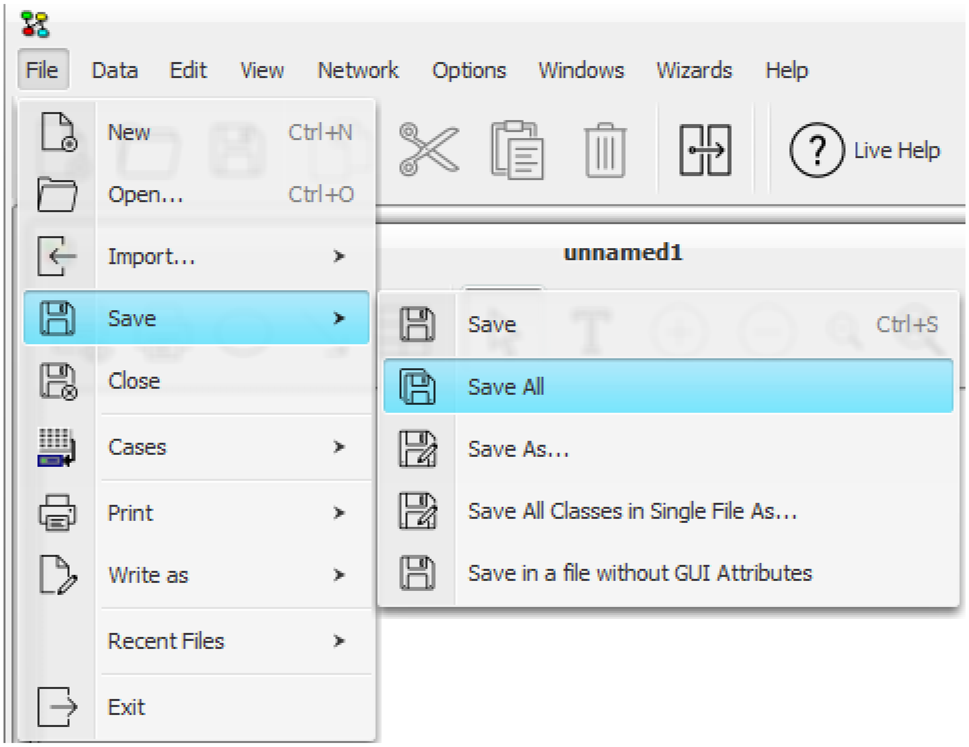


Figure 1: The Save All Menu Item.

The save all function works by invoking the *Save* (page 159) function for all current networks. This function is particularly useful to ensure that an OOBN is consistent.

4.1.64 Save Elimination Order

The *Save Elimination Order* (save triangulation) function can be activated by selecting the “Save Elimination Order” item of the *Network Menu* (page 185) when the network is in run mode. It will save an elimination order corresponding to the junction tree in the chosen file.

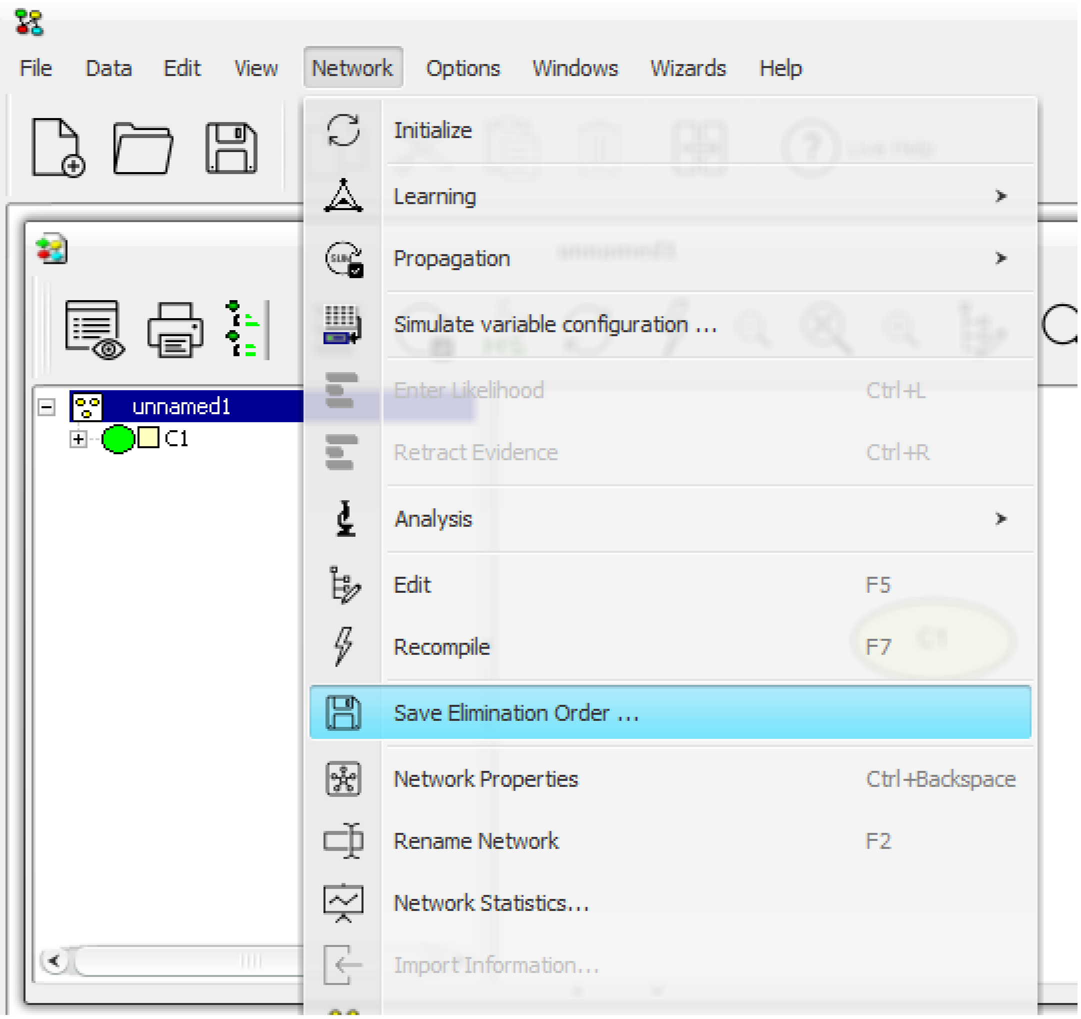


Figure 1: The Save Elimination Order Menu Item.

4.1.65 Save All Classes in Single File As…

The *Save All Classes in Single File As…* function is activated by selecting the item of the *File Menu* (page 191).

The single file will contain all classes loaded in HUGIN at the time. This enables the user to store multiple class definitions in a single file. (Please note that any flat networks will not be included in the file).

4.1.66 Copy

The copy function is activated either from the Copy item of the *Edit Menu* (page 192) or from pressing the copy button which is found in both the *Edit Mode* (page 124) Tool Bar and the *Run Mode* (page 123) Tool Bar (see Figure 1).

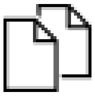


Figure 1: The copy tool button from the *Main Window* (page 186) Tool Bar.

When you have activated the copy function, you have the selected sub-network copied onto the clipboard of the Windows system. Then you can use the *paste function* (page 162) in the HUGIN Graphical User Interface to make a copy of it in the same or another Network Pane (which must be in Edit Mode).

When you cut a selected sub-network, all internal links in the sub-network are copied to the clipboard. Also links from external nodes to nodes within the selection are copied and can be restored when pasting. Links from nodes within the selection to external nodes, however, are not copied.

When pasting a sub-network from the clipboard, a link from an external node to a node within the sub-network is restored if the external node "exists" in the new network. Here, we say that a node "exists" if a node appears with the same *node name* (page 210) as the external node in the original HUGIN network. The external node in the new network needs not have the same *node label* (page 210) as the external node in the original network. If the external node does not "exist" in the new network, the link is not restored. This has impact on the *conditional probability table* (page 241) of the node which the link pointed to and one should reconstruct it.

4.1.67 Paste

You can activate paste either by selecting the Paste item from the *Edit Mode* (page 124) or by pushing the paste button from the *Main Window* (page 186) Tool Bar. Figure 1 shows the paste button.

Figure 1: The paste button from the Main Window Tool Bar.

The paste function lets you insert what is currently stored on the clip board. In *Edit Mode* (page 124) you can *cut* (page 162) or *copy* (page 161) a sub-network in the *Network Pane* (page 177) and afterwards paste it in the same Network Pane or in the Network Pane of another HUGIN network (*Bayesian network* (page 345), *LIMID* (page 345), or *object-oriented network* (page 346)).

4.1.68 Cut

The cut function is activated either by selecting the Cut item of the *Edit Menu* (page 192) or by pressing the cut tool button (see Figure 1) of the *Main Window* (page 186) Tool Bar.

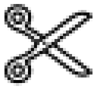


Figure 1: The cut tool button from the Main Window Tool Bar.

The cut function can only be activated in edit mode and when one or more nodes or causal links are selected in the Network Pane.

The selected sub-network will be deleted from the Network Pane and copied to the clipboard when the cut function is activated. Then you can *paste* (page 162) this sub-network to other open HUGIN networks or you can paste it in other applications having access to the clipboard.

When you cut a selected sub-network, all internal links in the sub-network are copied to the clipboard. Also links from external nodes to nodes within the selection are copied and can be restored when pasting. Links from nodes within the selection to external nodes, however, are not copied.

When pasting a sub-network from the clipboard, a link from an external node to a node within the sub-network is restored if the external node "exists" in the new network. Here, we say that a node "exists" if a node appears with the same *node name* (page 210) as the external node in the original HUGIN network. The external node in the new network needs not have the same *node label* (page 210) as the external node in the original network. If the external node does not “exist” in the new network, the link is not restored. This has impact on the *conditional probability table* (page 241) of the node which the link pointed to and one should reconstruct it.

4.1.69 Delete

The delete function is activated either by selecting the Delete item of the *Edit Menu* (page 192) or by pressing the delete tool button (see Figure 1) of the *Main Window* (page 186) Tool Bar.

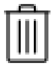


Figure 1: The delete tool button from the Main Window Tool Bar.

The delete function can only be activated in *edit mode* (page 124) and when one or more nodes or links are selected in the *Network Pane* (page 177).

The selected objects will be deleted from the network. It will *not* be copied to the clipboard. The *cut* (page 162) function is very similar to the delete function, but it copies the deleted objects to the clipboard.

4.1.70 Delete State

The delete state function can be activated from the Delete State item in the *Edit Menu* (page 192) or from the delete state button in the *Edit Mode* (page 124) Tool Bar (the button shown in Figure 1).

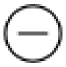


Figure 1: The delete state tool button from the Edit Mode Tool Bar.

The delete state function deletes a state/decision of the (discrete) node currently selected. The function only works for *discrete chance nodes* (page 223) and *decision nodes* (page 224).

The delete state function can only be activated when the currently selected node has two or more states.

4.1.71 Close

The close function is activated by selecting the Close item in the *File Menu* (page 191).

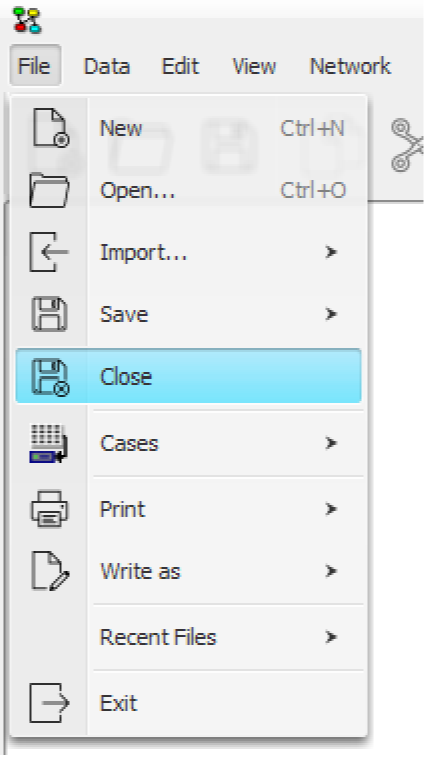


Figure 1: The Save As Menu Item.

The close function closes the currently selected network window. If there has been changes to the network of this network window, the user is asked if he/she wants to save these changes before the network window is closed.

4.1.72 Close All

The close all function is activated by selecting the Close All item in the *Window Menu* (page 189).

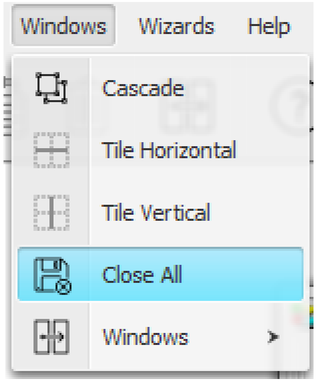


Figure 1: The Save As Menu Item.

The close all function closes all network windows currently open in the *Main Window* (page 186). If there has been changes to the network of a network window, the user is asked if he/she wants to save these changes before this network window is closed.

4.1.73 Open

The open function is activated either by selecting the Open item of the *File Menu* (page 191) or by pressing the open button from the *Main Window* (page 186). Figure 1 shows the open tool button.

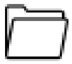


Figure 1: The open button from the Main Window Tool Bar.

The open function in HUGIN is like the open function in most other applications: It opens a saved HUGIN network. It can open files of both HKB format (compiled HUGIN network format) and NET format (standard network description format).

If the file is a password protected HKB file, a password dialog opens, in which the user can specify the password. If the password is incorrect, opening the file will fail.

4.1.74 New

The new function can be activated by selecting the New item of the *File Menu* (page 191) or by pressing the New tool button (see Figure 1) in the *Main Window* (page 186) Tool Bar.

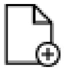


Figure 1: The new button from the Main Window Tool Bar.

The new function opens a new clean *network window* (page 182) where you start constructing a new HUGIN network (*Bayesian network* (page 345), *LIMID* (page 345), or *object-oriented network* (page 346))

4.1.75 Exit

Exit is activated by selecting the Exit item of the *File Menu* (page 191). You can also activate exit by simply closing the *Main Window* (page 186).

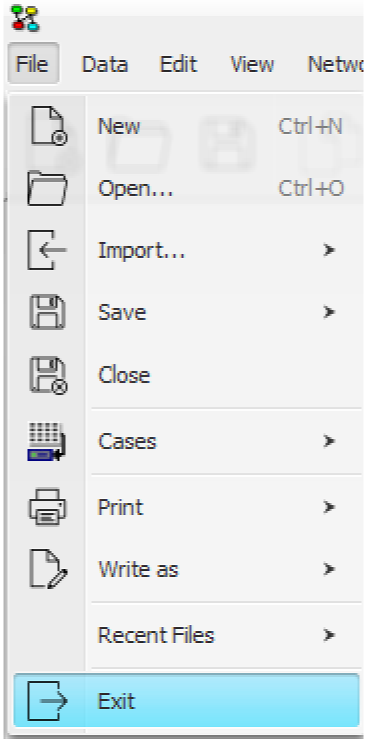


Figure 1: The Save As Menu Item.

The exit function closes all *Network Windows* (page 185) and exits from the HUGIN Graphical User Interface. Each time a network window is closed, the HUGIN Graphical User Interface asks you to save the network of this window if it contains unsaved changes.

4.1.76 Zooming

You can zoom in or out on the currently selected network using the buttons on the run and edit toolbars.

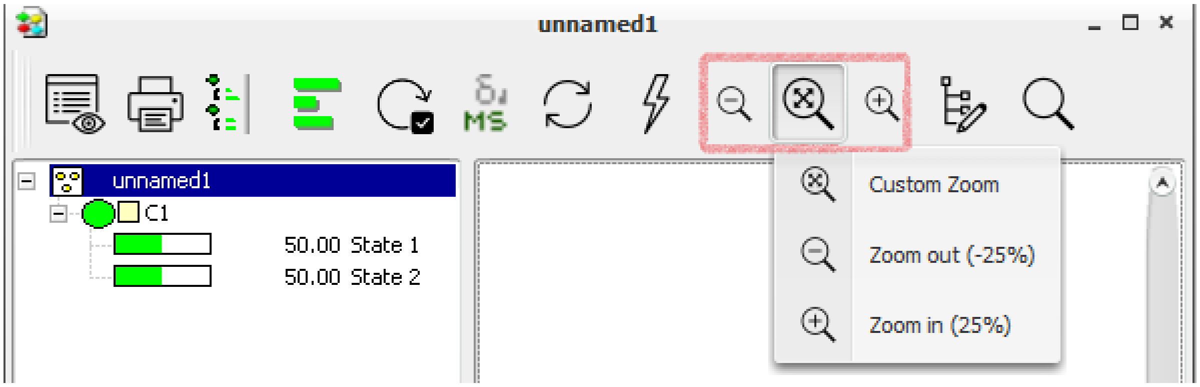
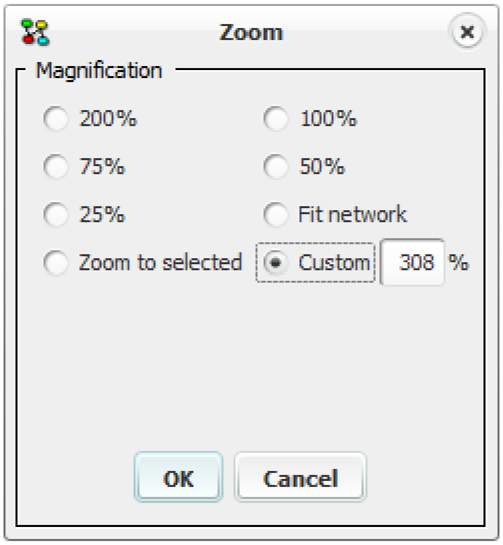


Figure 1: The Zoom buttons as seen in the run toolbar.

If you want to specify the magnification of the zoom, you can open a zoom pane by clicking the ‘Custom Zoom’ button among the zoom buttons. This function is useful on large networks to get a better view of the whole network. The Zoom Pane is shown in Figure 2.



Zoom

Magnification

○ 200% ○ 100%

○ 75% ○ 50%

○ 25% ○ Fit network

○ Zoom to selected ● Custom 308 %

Figure 2: The Zoom Pane.

You can also perform zooming operations using the following keyboard shortcuts:

- **Ctrl + ArrowUp:** Zoom in on the network (and monitor windows for discrete nodes, if in Run Mode). Current magnification is multiplied by 1/0.9.
- **Ctrl + ArrowDown:** Zoom out on the network (and monitor windows for discrete nodes, if in Run Mode). Current magnification is multiplied by 0.9.
- **Ctrl + Home:** Set magification to 100% for the currently selected network (and its monitor windows for discrete nodes, if in Run Mode).
- **Shift + Home:** Set magnification for the currently selected network (and its monitor windows for discrete nodes, if in Run Mode) such that the network fits the available area of the network window.
- **Ctrl + Shift + ArrowUp:** Similar to Ctrl + ArrowUp, except that only monitor windows are scaled.
- **Ctrl + Shift + ArrowDown:** Similar to Ctrl + ArrowDown, except that only monitor windows are scaled.
- **Shift + End:** Zoom to selected nodes, making the selected nodes fill out the entire view.

Scaling the monitor windows can be useful if they overlap.

4.1.77 Print

HUGIN has print facilities to print out a network in three different forms: You can print the *network graph* (page 168), the *node tables* (page 170) of the selected nodes, and the *probability distributions* (page 170) for selected nodes in *Run Mode* (page 123). The print buttons shown in Figure 1.

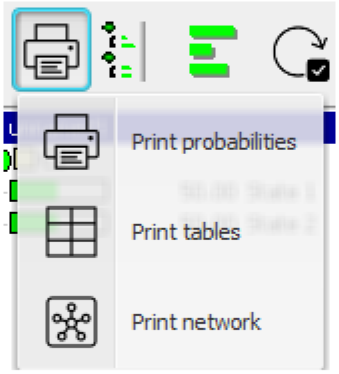


Figure 1: The print buttons in Run Mode.

4.1.78 Print Network

The print network function can be activated either by selecting the Print Network item from the *File Menu* (page 191) or by pressing the print network button (see Figure 1) in the *Main Window* (page 186) Tool Bar.

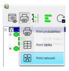
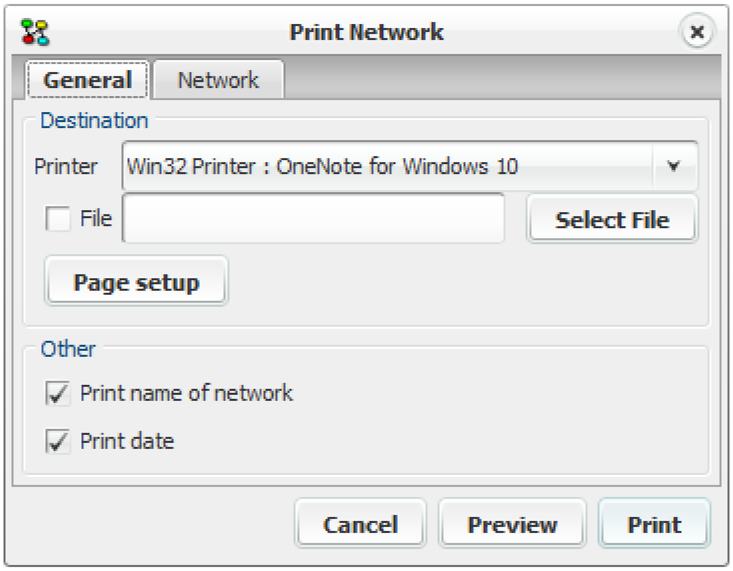


Figure 1: The print network button from the Main Window Tool Bar.

The *print* (page 167) network function prints the network graph of the currently selected *network window* (page 182) using the print dialog shown in Figure 2.



Print Network

General | Network

Destination

Printer: Win32 Printer : OneNote for Windows 10

□ File

Select File

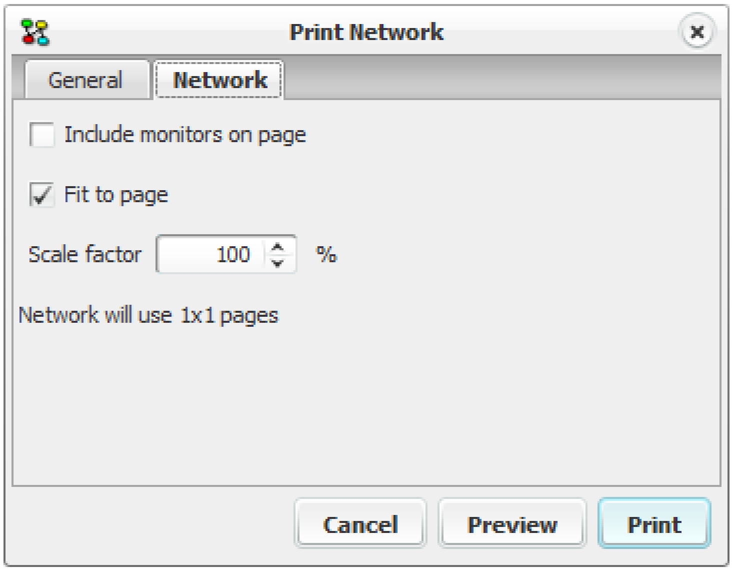
Page setup

Other

☑ Print name of network

☑ Print date

Cancel | Preview | Print



□ Include monitors on page

☑ Fit to page

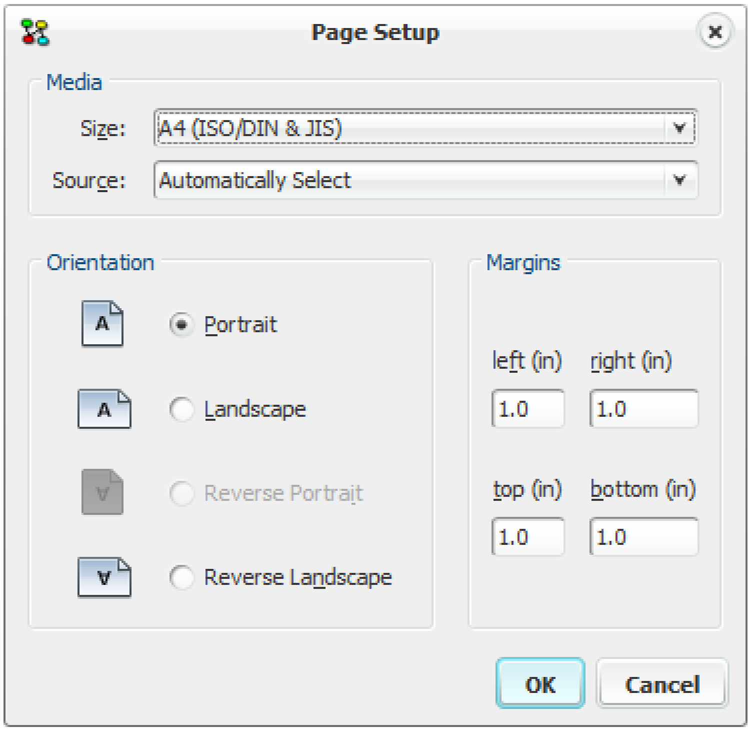
Scale factor 100 %

Network will use 1x1 pages

Figure 2: The General and Network tabs of the print network dialog.

As per default, the network will be printed on a single sheet, making large networks be scaled down. If the scale factor is increased, the network might be too large to fit within the size of a single sheet of the selected media type, and hence will be printed on several sheets (indicated in rows-by-columns format).

Checking the option "include monitors on page" will result in all visible *monitors* (page 194) in the network being printed as well.



Size: A4 (ISO/DIN & JIS)

Source: Automatically Select

● Portrait

○ Landscape

○ Reverse Portrait

○ Reverse Landscape

left (in)	right (in)
1.0	1.0
top (in)	bottom (in)
1.0	1.0

Figure 3: The Page Setup dialog of the print network dialog.

Selecting the “Write as BMP” item of the *File Menu* (page 191), the network graph can also be printed to a BMP file.

4.1.79 Print Node Table

The print node table function can be activated either by selecting the Print Node Table item from the *File Menu* (page 191) (only in Edit Mode) or by pressing the print node table button (see Figure 1) in the Edit Mode Tool Bar.

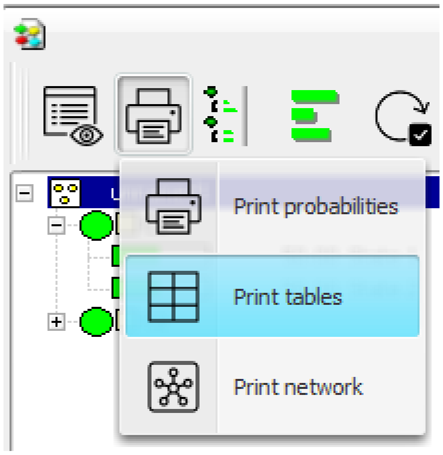


Figure 1: The print node table button from the Edit Mode Tool Bar.

The print node table function *prints* (page 167) the tables (*CPTs* (page 241), experience tables, fading tables, or *utility tables* (page 233)) of the currently selected nodes using a standard print dialog. The print node table function will also print the expression if the table containes one and the option is not unchecked.

Selecting the “Write as BMP” item of the *File Menu* (page 191), the node tables can also be written to a BMP file.

4.1.80 Print Probabilities

The print probabilities function can be activated either by selecting the Print Probabilities item from the *File Menu* (page 191) (only in Run Mode) or by pressing the print probabilities button (see Figure 1) in the *Run Mode* (page 123) Tool Bar.

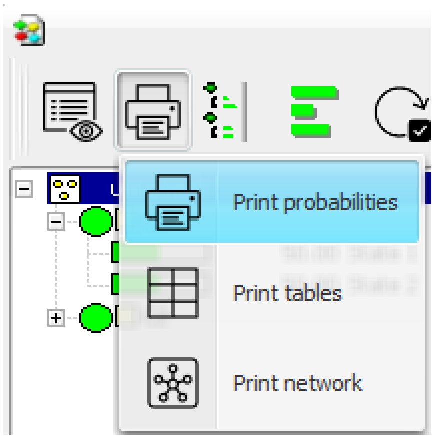


Figure 1: The print probabilities button from the Run Mode Tool Bar.

The print probabilities function *prints* (page 167) the posterior probability distributions for the selected nodes. When you activate the print probabilities function, a standard print dialog appears.

Selecting the "Write as BMP" item of the *File Menu* (page 191), the probability distributions can also be written to a BMP file.

4.1.81 Generate Network HTML Documentation

Using the *generate documentation* tool it is possible to automatically generate a documentation of the selected network. The generate documentation tool will generate a HTML documentation of the selected network describing the user defined properties of the network including network and node descriptions, attribute descriptions, and node state descriptions.

Notice that the *generate documentation* tool may use a significant amount of time to document a large and complex model. The tool can be accessed through the menu item shown in Figure 1.

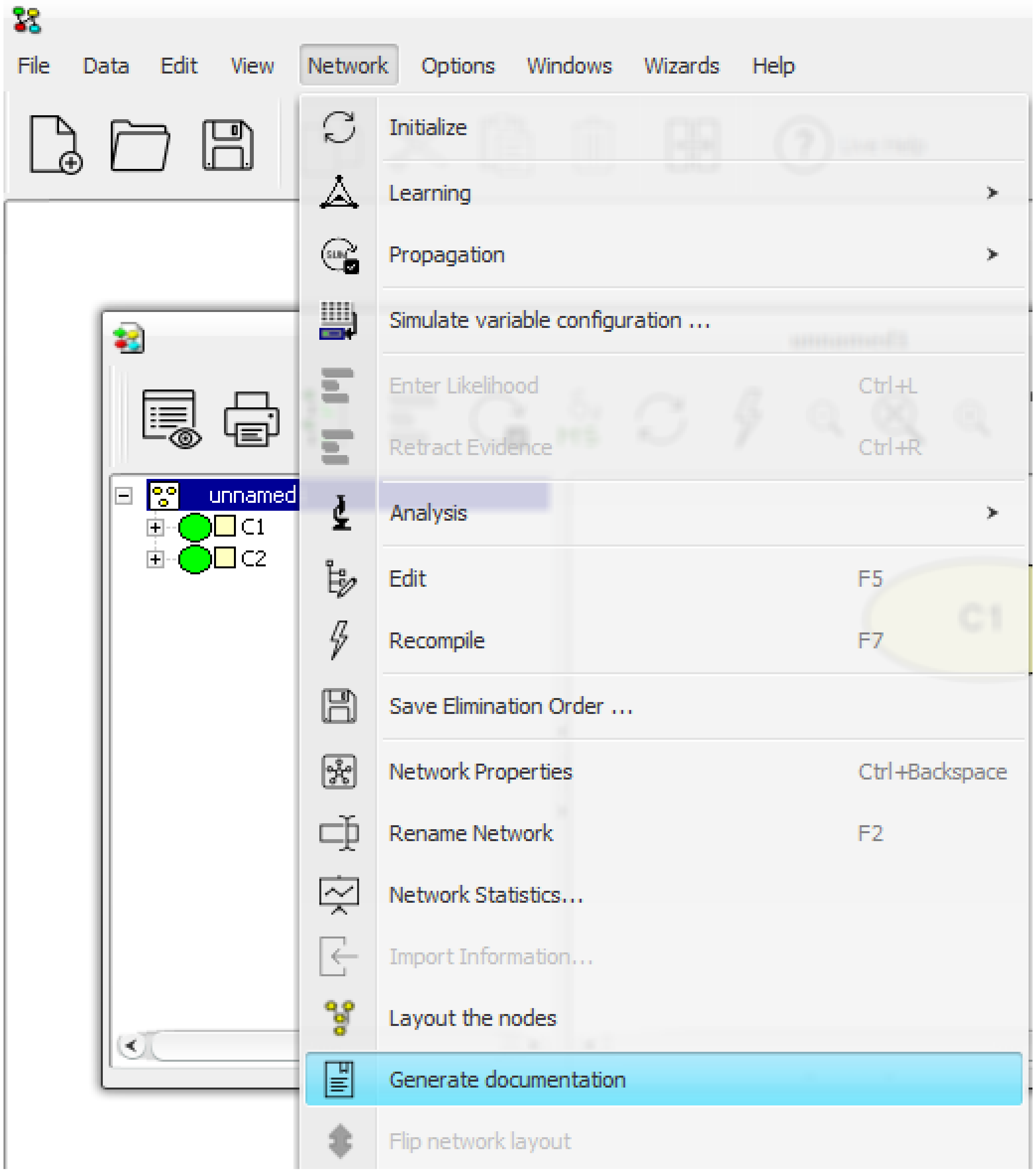
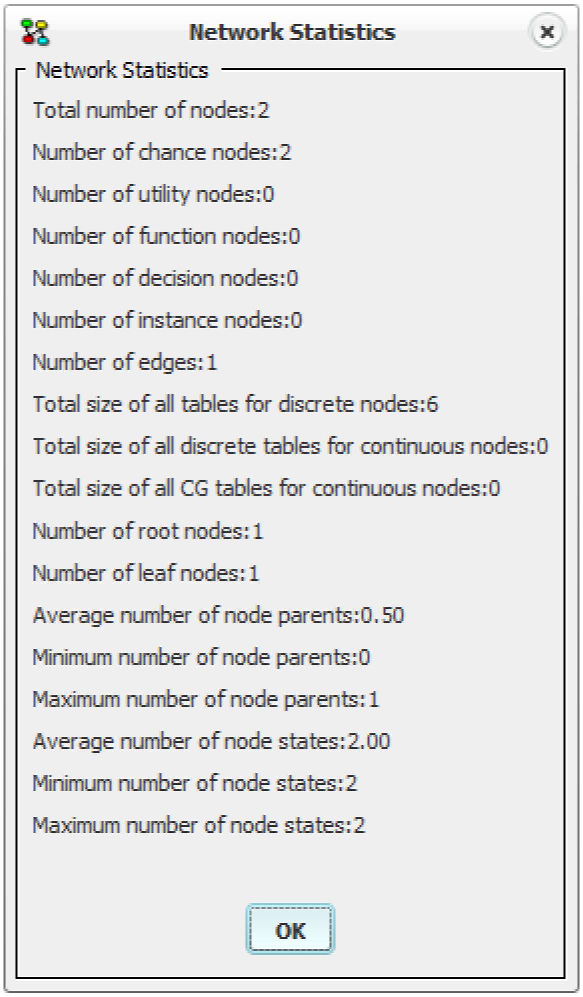


Figure 1: The Generate Documentation menu item.

4.1.82 Statistics

The statistics function can be activated by selecting the “Statistics” item of the Network Menu. This brings up the “Statistics” panel showing various statistics about the network and it’s nodes as depicted in Figure 1.



Total number of nodes:2
Number of chance nodes:2
Number of utility nodes:0
Number of function nodes:0
Number of decision nodes:0
Number of instance nodes:0
Number of edges:1
Total size of all tables for discrete nodes:6
Total size of all discrete tables for continuous nodes:0
Total size of all CG tables for continuous nodes:0
Number of root nodes:1
Number of leaf nodes:1
Average number of node parents:0.50
Minimum number of node parents:0
Maximum number of node parents:1
Average number of node states:2.00
Minimum number of node states:2
Maximum number of node states:2

Figure 1: The “Statistics” pane in an empty network.

4.1.83 Recall Policies

The Recall Policies feature is activated from *Run Mode* (page 123) by pressing the *Recall Policies* button (see Figure 1).

The function recalls the policies stored using the *Store Policies* (page 174) function. This is a valuable feature when performing analysis and experimenting with the decisions of the LIMID - as one can revert to the original policies when done.

Note: Modifying the LIMID automatically discards any stored policies.

Figure 1: The ‘Recall Policies’ button from the Run Mode Tool Bar.

4.1.84 Store Policies

The Store Policies feature is activated from *Run Mode* (page 123) by pressing the *Store Policies* button (see Figure 1).

The function stores a temporary backup in memory containing all policies. The stored policies can be recalled by pressing the *Recall Policies* (page 173) button. This is a valuable feature when performing analysis and experimenting with the decisions of the LIMID - as one can revert to the original policies when done.

Note: Modifying the LIMID automatically discards any stored policies.

Figure 1: The ‘Store Policies’ button from the Run Mode Tool Bar.

4.1.85 Reset uninstantiated policies

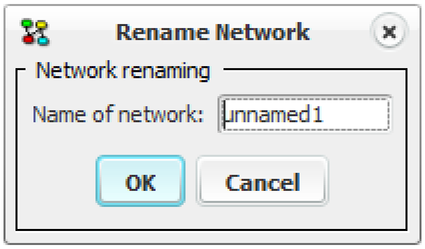
The reset uninstantiated policies feature is activated from *Run Mode* (page 123) by pressing the reset uninstantiated policies button (see Figure 1).

The function resets any uninstantiated policies to the uniform distribution.

Figure 1: The ‘Reset uninstantiated policies’ button from the Run Mode Tool Bar.

4.1.86 Rename Network

The rename network function can be activated by selecting the “Rename Network” item of the *Network Menu* (page 185) or by right-clicking the title bar of the network. This brings up the “Rename Network” panel depicted in Figure 1.


Rename Network

Network renaming

Name of network: unnamed1

Figure 1: The “Rename Network” pane.

Note, when working with OOBNs the name of the network must be unique within the OOBN. Furthermore, changing the name of a network instantiated in another network will change the name of the instance nodes derived from the network. **Warning**: if the name of such a network is changed without the network containing the derived instances being open, the OOBN will become corrupt. HUGIN will give a warning in that case, but if the warning is ignored, the network with the derived instances can no longer be opened!

The name of a network can also be changed by selecting the *Save as* (page 158) item in the *File Menu* (page 191).

4.1.87 Show Last Error

The *show last error* function is activated by selecting the Show Last Error item of the *View Menu* (page 188).

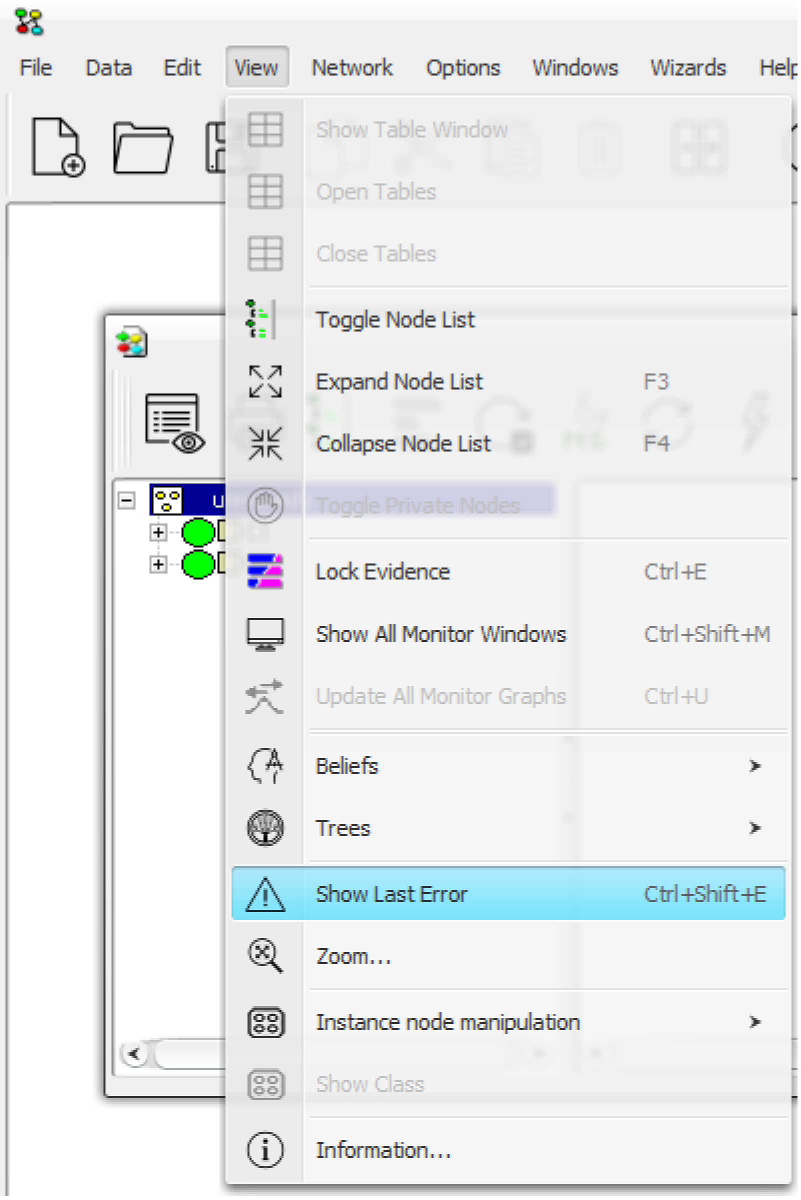


Figure 1: The Show Last Error menu item from the View Menu.

If you get an error in HUGIN, you can use this function to view the error message and a description of the error in a small dialog box.

4.1.88 Bug Report

The Report a Bug function can be activated by selecting the Report a Bug item of the *Help Menu* (page 191) (Figure 1). Should you encounter a bug, an undesired behavior, or if you wish to suggest improvements of the HUGIN Graphical User Interface, you can use this convenient on-line facility. We very much appreciate any feedback.

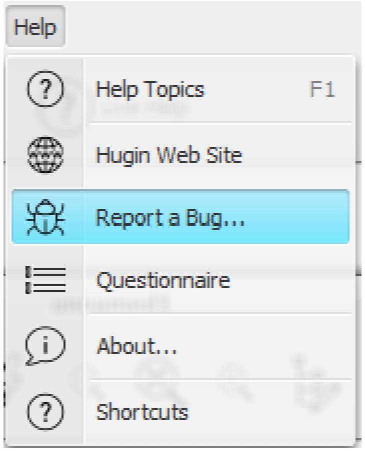


Figure 1: The Bug Report menu item in the Help Menu.

4.1.89 Report Beliefs

The report beliefs function can be activated either by selecting the Report Beliefs item from the *View Menu* (page 188) (Figure 1 - only in Run Mode) or by pressing the right mouse button on a node in the node list.

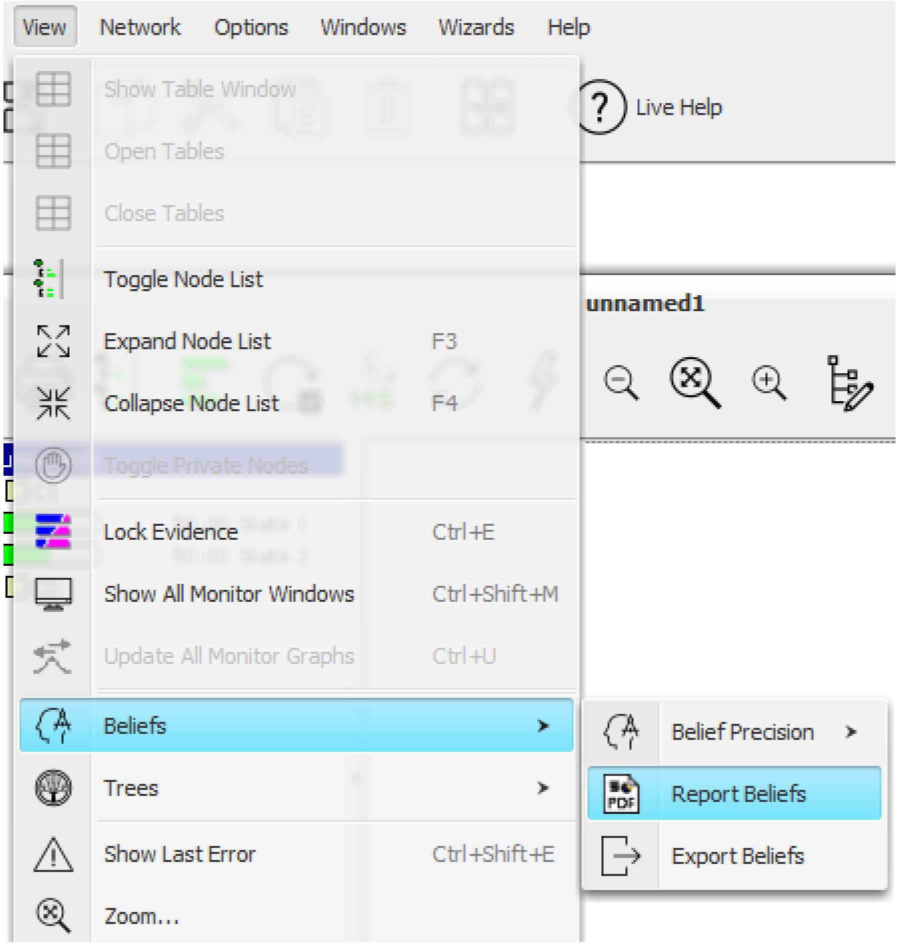


Figure 1: The Report Belief menu item in the View Menu.

The report beliefs function prints the posterior probability distributions for the selected nodes to the network log.

4.2 Network Windows and Menus

4.2.1 Network Pane

The *Network Pane* is the part of a *network window* (page 182) which contains the graph of the current network. It displays the relationship between the *nodes* (page 209) represented by *links* (page 147). The Network Pane is visible in both *Edit Mode* (page 124) and *Run Mode* (page 123).

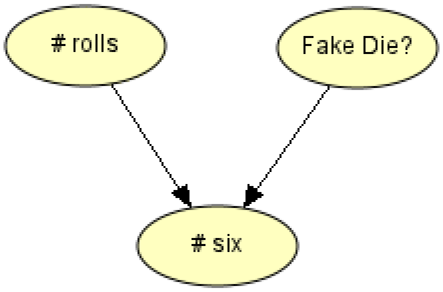


Figure 1 shows the Network Pane containing a *network* (page 345).

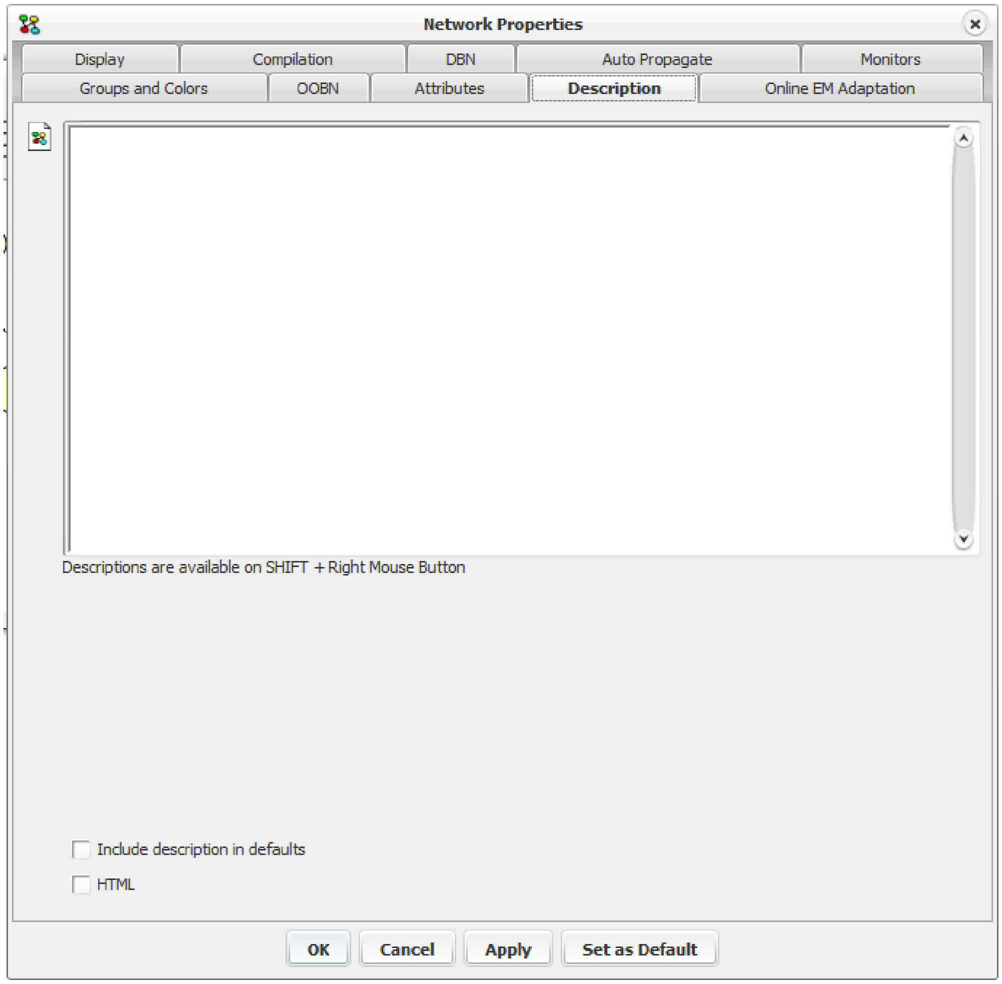
4.2.2 Network Properties

Network properties can be viewed by selecting the Network Properties item of the *Network Menu* (page 185). It allows you to set up the HUGIN Graphical User Interface tool as suits you best.

In the network properties dialog box are seven tabs covering different parts of the tool:

- *Attributes* (page 183)
- *Auto Propagation* (page 129)
- *Compilation* (page 126)
- Description
- *DBN* (page 183)
- *Display* (page 205)
- *Groups and Colors* (page 207)
- *Monitors* (page 194)
- *Object-Orientation* (page 346)
- *Online EM Adaptation* (page 245)
- *Toolbar* (page 152)

All these tabs except for the Description tab are covered elsewhere in the manual (under their name). The Description tab contains a simple text editor pane for describing the network for documentation purposes. The user has the option of allowing the support of hyperlinks or not by selecting th “HTML” check box. The network properties dialog box and the Description tab are shown in Figure 1.



Descriptions are available on SHIFT + Right Mouse Button

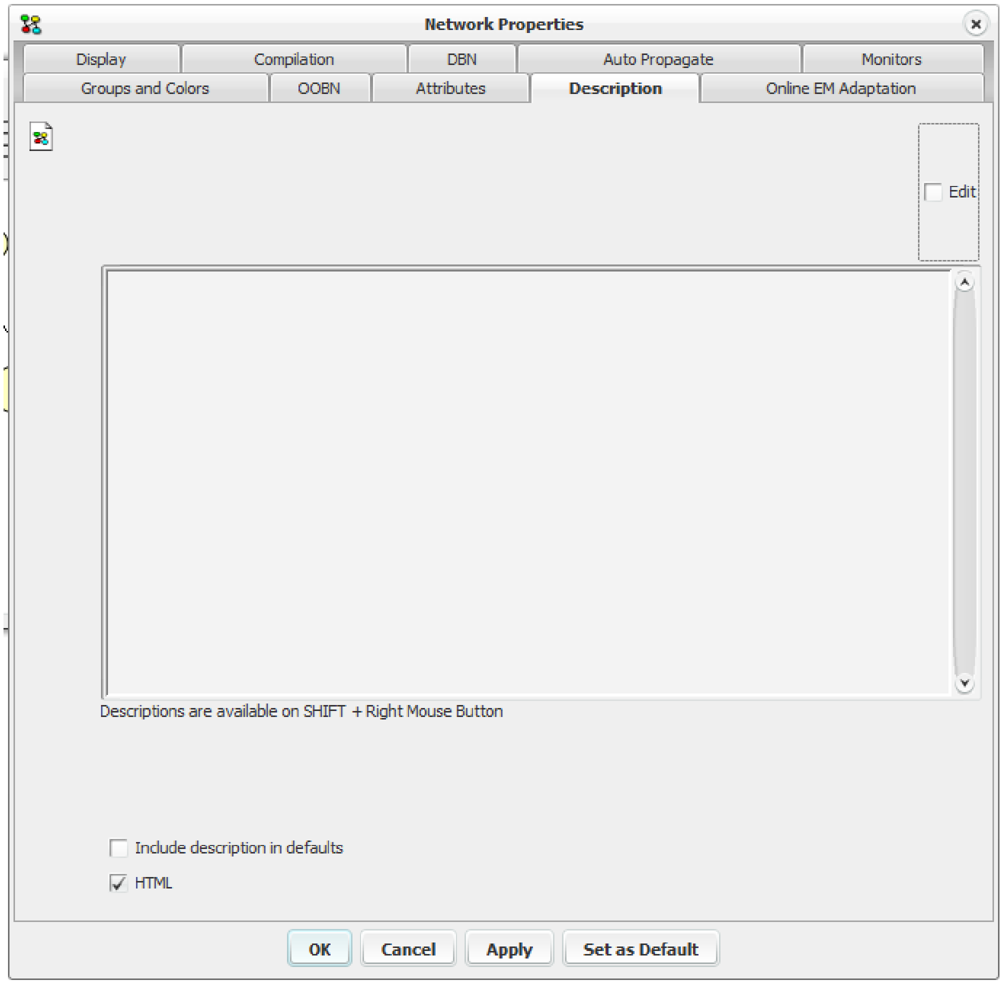
□ Include description in defaults

□ HTML

Figure 1: The network properties dialog box with the Description tab selected.

The "HTML" check box is located at the bottom of the panel. This check box allows the user to specify if hyperlinks should be allowed or not in the Network and the *Node description* (page 212). If the check box is selected the description text is saved as html and hyperlinks are allowed. (see Figure 2).

The "Edit" check box is located at the top right corner when selecting "HTML".(see Figure. 2). Selecting it allows the editing of the "description text" and of course the creation of hyperlinks. If it is not selected and hyperlinks are present in the text, they can be activated by clicking on them or their hyperlink paths can be visible on the status bar by pointing at them with the mouse.



□ Edit

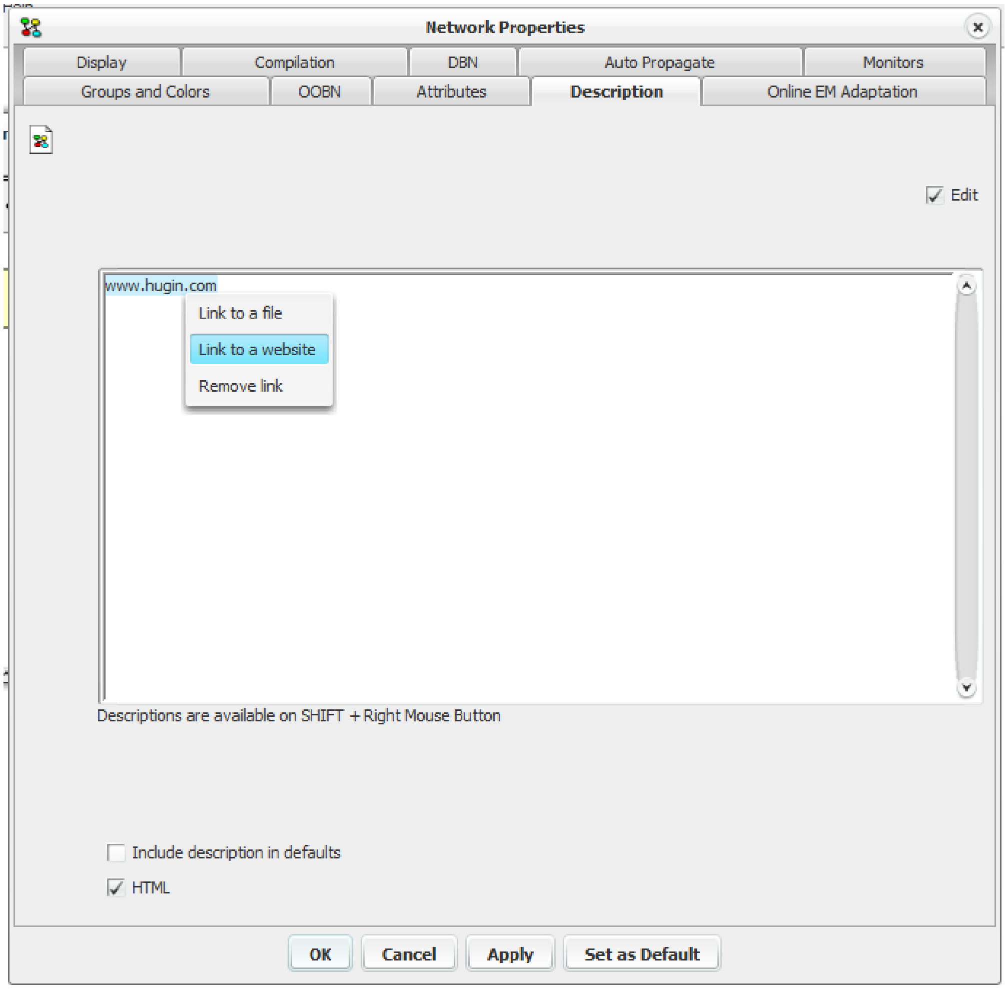
Descriptions are available on SHIFT + Right Mouse Button

□ Include description in defaults

☑ HTML

Figure 2: The network properties dialog box with the Description tab selected and html check box selected.

Create a hyperlink: To create a hyperlink select the text to turn into a link and right click on the mouse. A pop-up displays (see Figure 3.) which gives the user three options:



☑ Edit

www.hugin.com

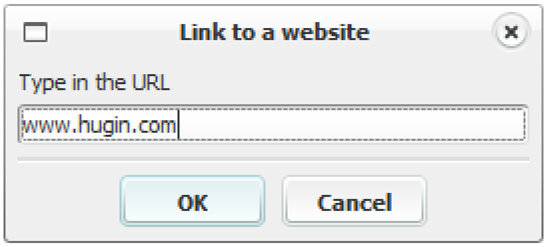
Descriptions are available on SHIFT + Right Mouse Button

☐ Include description in defaults

☑ HTML

Figure 3: The network properties dialog box in edit mode. Creating a hyperlink.

- **Link to a file** initiates a file chooser panel which allows the user to select a file to link to. If the selected text had already a file link associated with it, the path to that file is selected in the file chooser. Selecting a file and pressing "Select" associates the selected text with the file.
- **Link to a web site** initiates a panel that prompts the user to type the address of a website (Figure 4). If the selected text was already associated with a link to a web site, the link is displayed in the field. Pressing OK after typing the url will associate the selected text with the url.
- **Remove link** as it suggests, removes the link from the selected text.(if there is one).



Type in the URL

www.hugin.com

Figure 4: The dialog that prompts the user to type in a web site address.

The creation of hyperlinks in the *"Node Properties" panel* (page 212) is done in the same way as above if the html check box is selected in the "Network Properties" panel.

The activation of the hyperlinks is possible in the *network panel* (page 177) by pressing SHIFT + Right mouse button and in the Network Properties panel and Node Properties panel if they are not in edit mode.

The documentation generated by the *Generate documentation* (page 171) option(*Menu-Network-Generate documentation* (page 185)) includes the hyperlinks created in Network and Node description. The hyperlinks can also be activated from the generated document.

Pointing on a hyperlink with the mouse, displays the link's path in the status bar at the bottom of the network panel.

When changing from html mode to plain text mode, (by unselecting the html check box) a panel with a warning message prompts the user to confirm the action because all hyperlinks will be lost if the descriptions change to plain text. The hyperlinks in the Network description are lost immediately. The node descriptions are still available when changing back to html mode if they haven't been edited while in plain text mode.

If you are editing or running a network (*Bayesian network* (page 345), *influence diagram* (page 345), or *object-oriented network* (page 346)), you can view the network description by pressing SHIFT + right mouse button in the *Network Pane* (page 177).

4.2.3 Network Window

Within the *Main Window* (page 186) you can open a series of *network windows*. Each network window allows you to edit or run a HUGIN network (*Bayesian network* (page 345), *LIMID* (page 345), or *object-oriented network* (page 346))

The network windows can appear in two different ways depending on the mode you are currently working in. If you are constructing (editing) a network, the network window is in *Edit Mode* (page 124) (see Figure 1). If you are using the network for computations, the network window is in *Run Mode* (page 123) (see Figure 2).

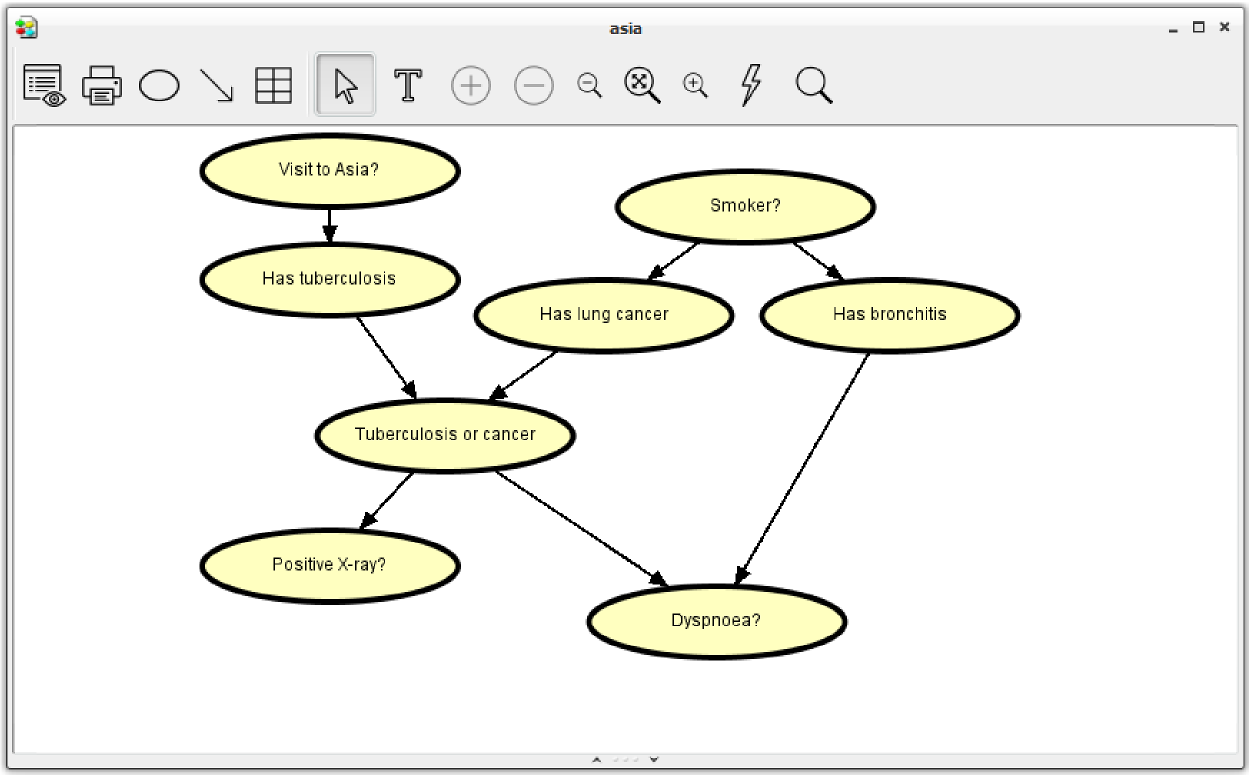


Figure 1: A network window in Edit Mode.

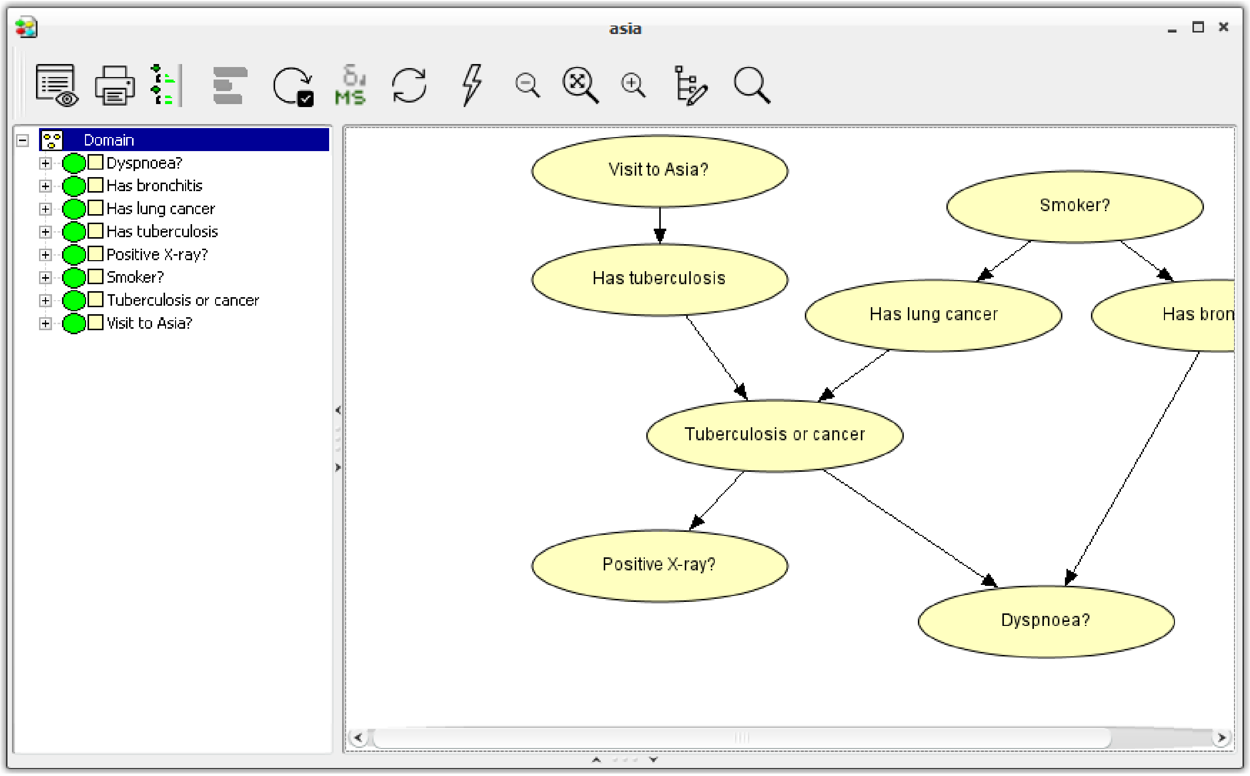


Figure 2: A network window in Run Mode.

4.2.4 Network Attributes

A set of Network Attributes for the model can be defined in the Network Properties Attributes tab (see Figure 1). An Attribute is a key-value pair defining a variable's name and its value. The Network Attributes defined in this panel are completely ignored by the HUGIN GUI1. The Attributes can however be accessed using the HUGIN API, which enables the GUI User to pass parameters to a user program from within the model. This is useful for wiring a user program with a model when for instance a node's state must be mapped to a traffic light's red, amber or green lights. To add a new Attribute, write a name in the name bar (shown inside the "List of Attributes" frame) and click 'Add'. Attribute descriptions can be associated with an attribute name only after creation.

Figure 1: The Network Properties dialog box showing the Network Attributes tab.

Attributes can also be defined on nodes. See Node Attributes.

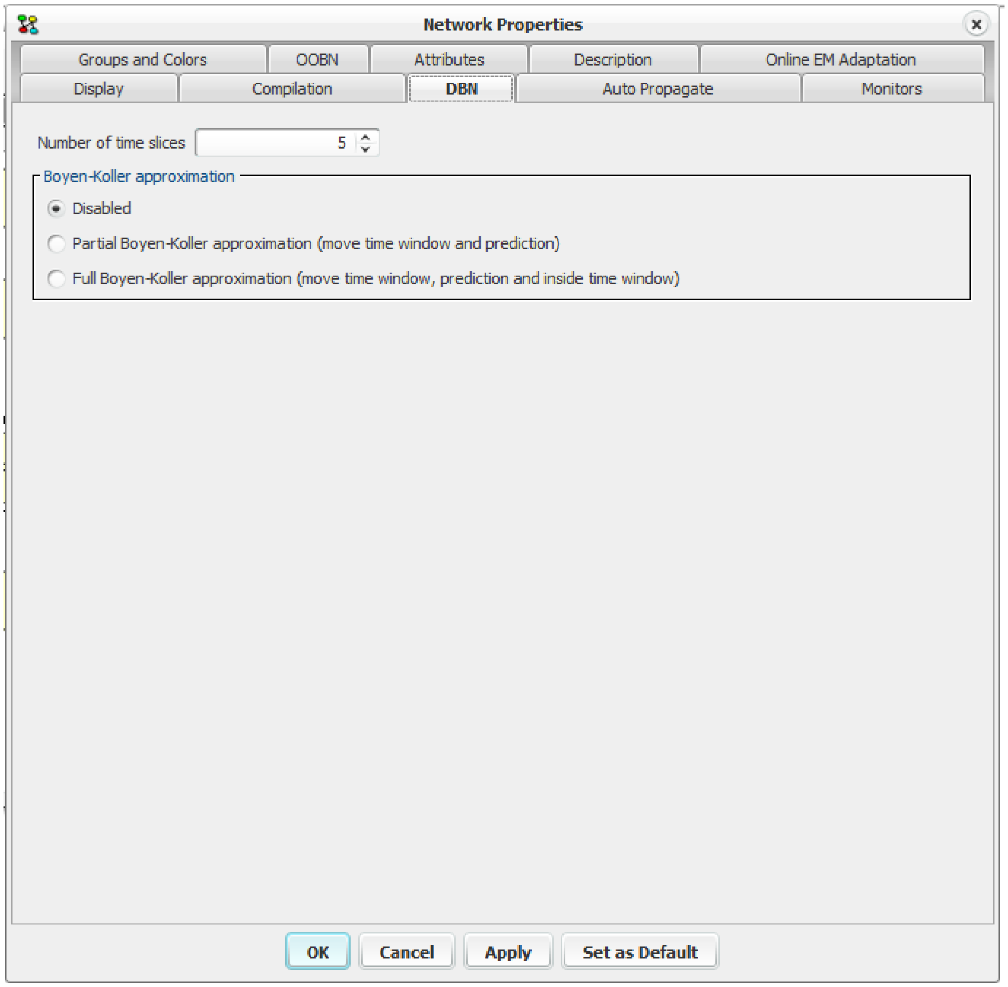
4.2.5 Network Properties DBN Pane

Various Dynamic Bayesian Networks (DBN) parameters can be modified in the Network Properties DBN tab, shown in Figure 1. The parameters that can be modified are:

- The Number of Time Slices: Specifies the number of time slices that is generated when the model is compiled.
- **Boyen-Koller approximation: Specifies the level of Boyen-Koller approximation to be used. The options are:**
  - Disabled: No approximation is used.
  - Partial Boyen-Koller approximation (move time window and prediction)

- Full Boyen-Koller approximation (move time window, prediction and inside time window)

More details can be read about HUGIN's implementation of DBN models, time slices and Boyen-Koller approximation in HUGIN C API Help Document distributed along with the software.



Number of time slices 5

Boyen-Koller approximation

● Disabled

○ Partial Boyen-Koller approximation (move time window and prediction)

○ Full Boyen-Koller approximation (move time window, prediction and inside time window)

Figure 1: The DBN dialog in the network properties.

Figure 1: The network properties dialog box with the DBN tab selected.

4.2.6 Network Menu

The Network Menu (shown in Figure 1) is found in the menu bar of the *Main Window* (page 186).

The Network Menu contains items for different actions that can be performed on a HUGIN model. Please note that some of the actions are available only in Run Mode.

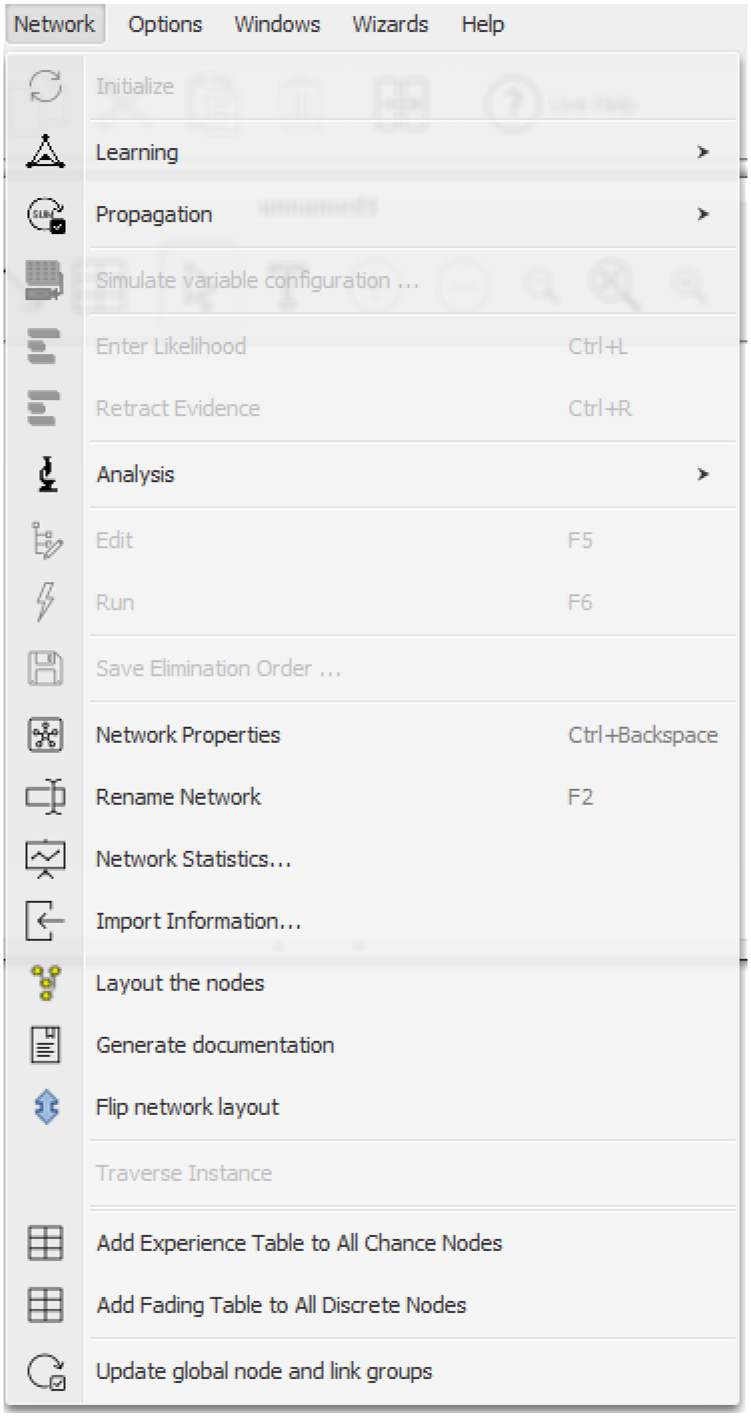


Figure 1: The contents of the Network Menu.

4.2.7 Main Window

The Main Window is built up to contain a series of *network windows* (page 182) each showing an open HUGIN network (*Bayesian network* (page 345), *LIMID model* (page 345), or *object-oriented network* (page 346)). The Main Window has a menu bar, a tool bar (called the Main Window Tool Bar), and a pane which can contain a number of network windows. Below this pane is the status bar which shows different information about the currently selected network window.

Please note that each item of the menu bar and the tool bar is clickable at the image in Figure 1. This provides you with a quick means of getting to the relevant information.

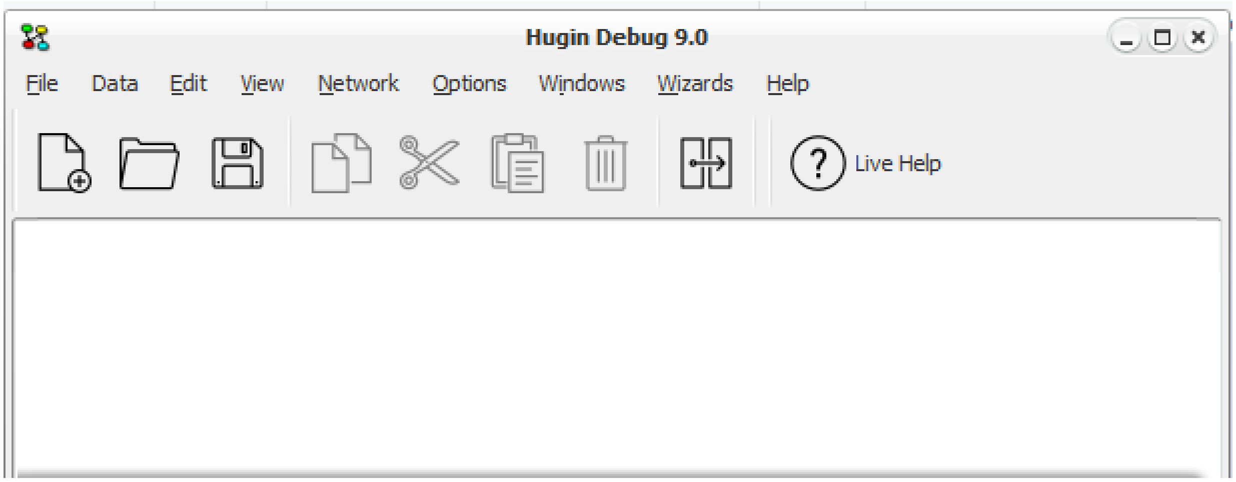
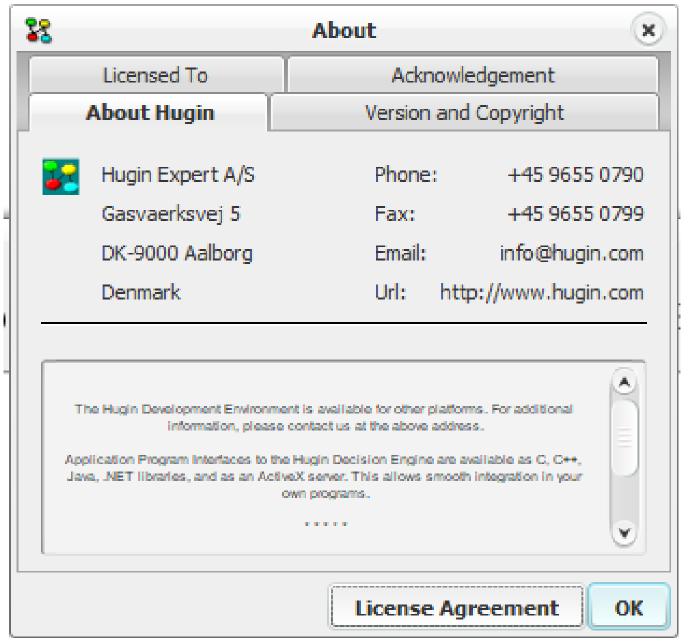


Figure 1: The Main Window

4.2.8 About

The About item of the *Help Menu* (page 191) Help Menu gives you information about version, license, copyrights, contact information, etc. Also, you get information about the HUGIN Development Environment, and application program interfaces available if you wish to develop your own applications based on Bayesian-network technology.



Hugin Expert A/S
Gasvaerksvej 5
DK-9000 Aalborg
Denmark

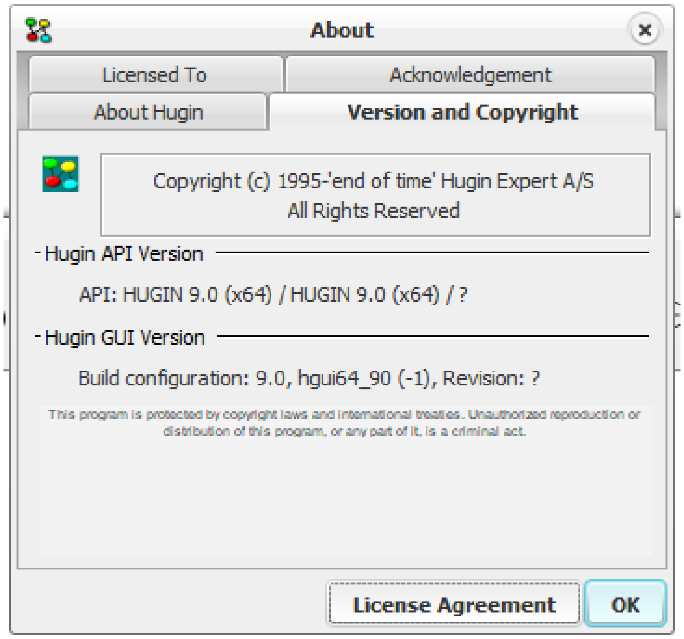
Phone: +45 9655 0790
Fax: +45 9655 0799
Email: info@hugin.com
Url: http://www.hugin.com

The Hugin Development Environment is available for other platforms. For additional information, please contact us at the above address.

Application Program Interfaces to the Hugin Decision Engine are available as C, C++, Java, .NET libraries, and as an ActiveX server. This allows smooth integration in your own programs.

.....

Figure 1: The about dialog box showing contact information, information about the HUGIN Development System, etc.



Copyright (c) 1995-'end of time' Hugin Expert A/S
All Rights Reserved

Hugin API Version

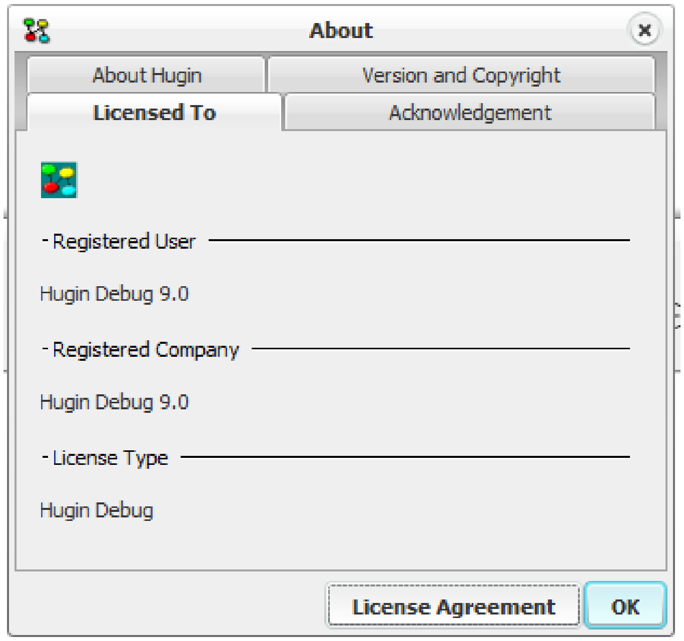
API: HUGIN 9.0 (x64) / HUGIN 9.0 (x64) / ?

Hugin GUI Version

Build configuration: 9.0, hgui64_90 (-1), Revision: ?

This program is protected by copyright laws and international treaties. Unauthorized reproduction or distribution of this program, or any part of it, is a criminal act.

Figure 2: The about dialog showing version and copyright information.



- Registered User

Hugin Debug 9.0

- Registered Company

Hugin Debug 9.0

- License Type

Hugin Debug

Figure 3: The about dialog box showing license information.

4.2.9 View Menu

The View Menu is found in the menu bar of the *Main Window* (page 186). It contains items which allow the user to change the view of the currently selected network. The View Menu is shown in Figure 1.

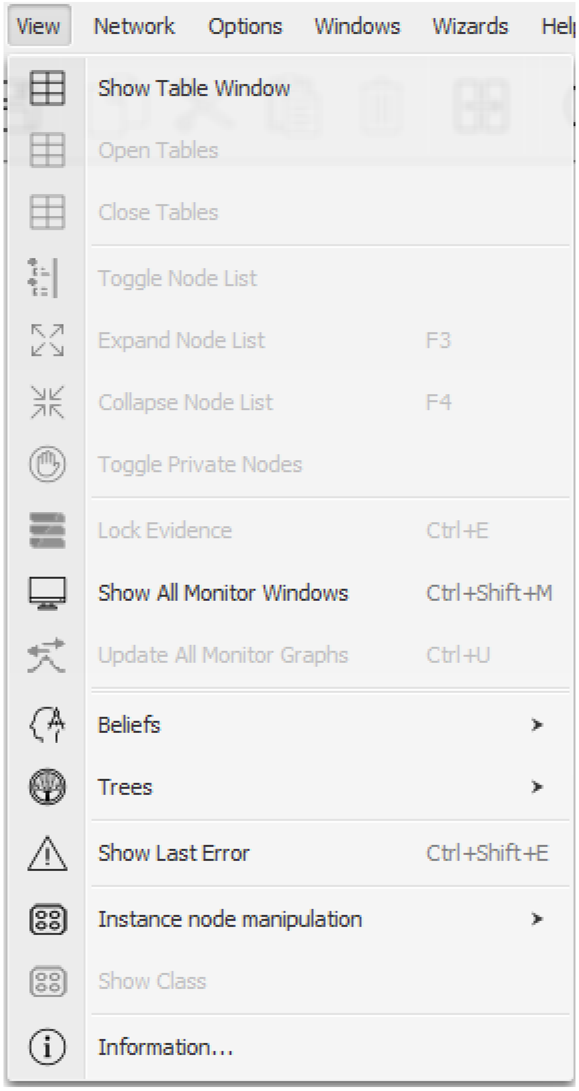


Figure 1: The View Menu of the Main Window.

4.2.10 Window Menu

The Window Menu is found in the menu bar of the *Main Window* (page 186). It contains items which allow the user to arrange the currently selected *network windows* (page 182) within the Main Window. The Window Menu is shown in Figure 1.

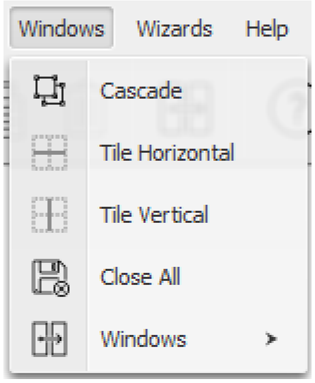


Figure 1: The Window Menu of the Main Window.

4.2.11 Wizards Menu

The Wizards Menu is found in the menu bar of the *Main Window* (page 186). It contains a number of items: The general *Learning Wizard* (page 267), the *EM Learning Wizard* (page 281), the *Analysis Wizard* (page 259), the *Hidden Node Analyzer* (page 281), the *Feature Selection Analyzer* (page 284), the *Adaptation Wizard* (page 267), the *Parameter Sensitivity Wizard* (page 288) and the *Code Wizard* (page 280). The Wizards Menu is shown in Figure 1.

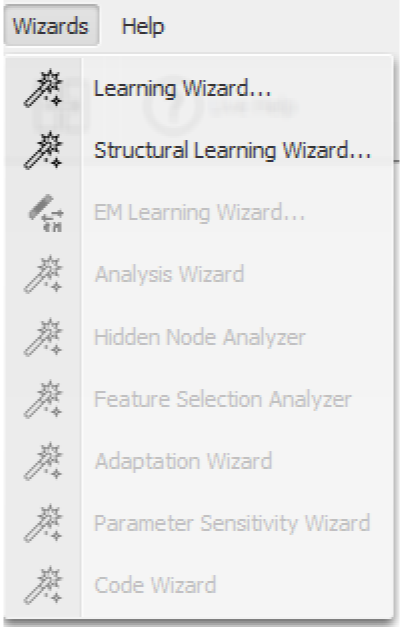


Figure 1: The Wizards Menu of the Main Window.

4.2.12 File Menu

The File Menu (shown in Figure 1) is found in the menu bar of the *Main Window* (page 186).

The File Menu contains items covering functions for manipulating network files (open, close, save, etc.).

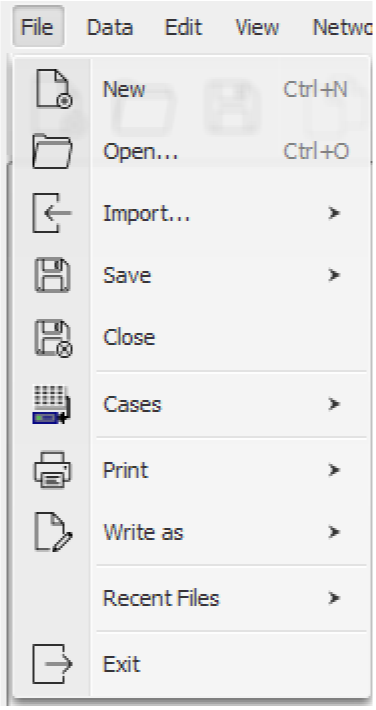


Figure 1: The File Menu.

4.2.13 Help Menu

The Help Menu is found in the menu bar of the *Main Window* (page 186).

The Help Menu contains items for different kinds information about the HUGIN Graphical User Interface. Figure 1 shows the Help Menu.

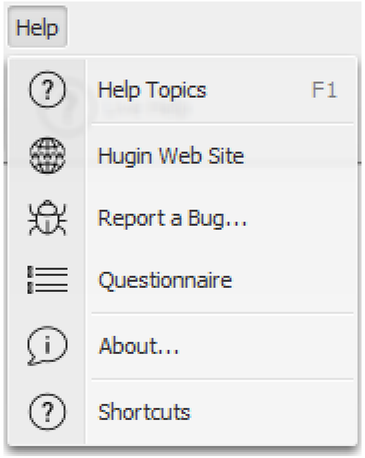


Figure 1: The Help Menu.

4.2.14 Edit Menu

The Edit Menu (shown in Figure 1) is found in the menu bar of the *Main Window* (page 186). The Edit Menu contains items covering functions mainly for constructing a HUGIN network. This means that most of them are only active in *Edit Mode* (page 124). Some of them, however, can also be used in *Run Mode* (page 123).

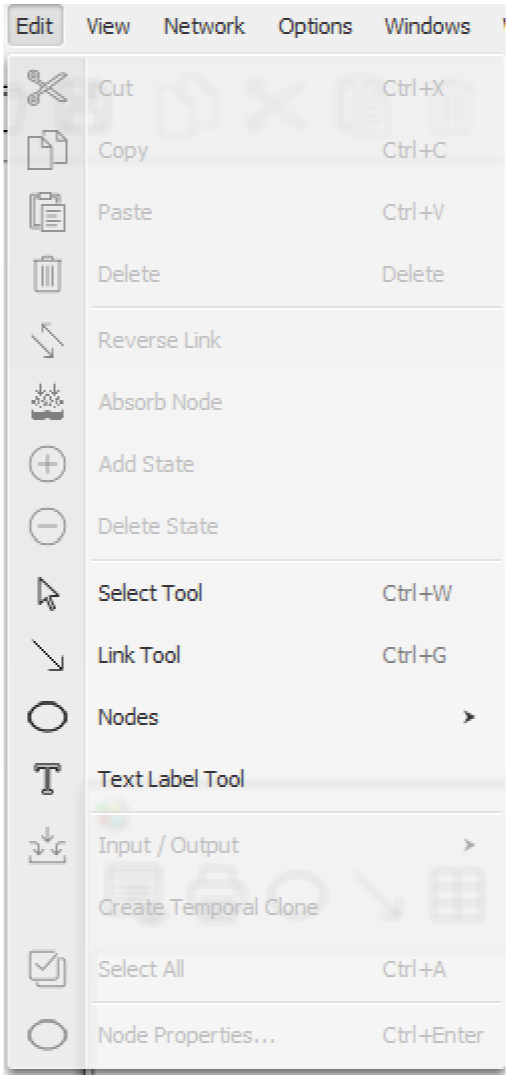


Figure 1: The Edit Menu.

4.2.15 Options Menu

The Options Menu is found in the menu bar of the *Main Window* (page 186). It contains an item for making different kinds of alignment operations on the currently selected nodes and an item that provides you with a dialog box for customizing the HUGIN Graphical User Interface in different ways.

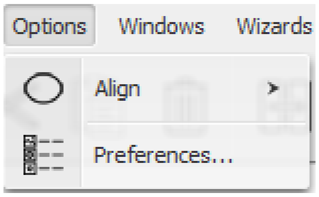
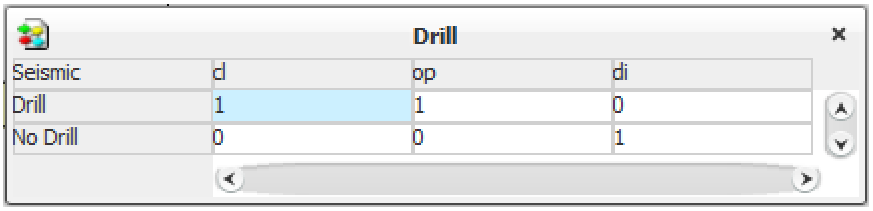


Figure 1: The Options Menu from the Main Window.

4.2.16 Policy Window

A policy window that displays the policies for a given *decision node* (page 224). If the node is instantiated then the policy is ignored and the policy window is disabled. Only policies with parent configurations of a non-zero probability are displayed. This is indicated by collapsing columns corresponding to parent configurations of zero probability, which happens e.g. when a parent node has evidence entered (see Figures 1 and 2).

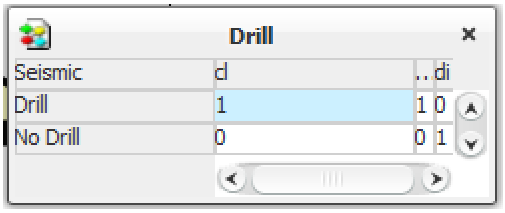
See also *Single Policy Update* (page 258).



Drill

Seismic	cl	op	di
Drill	1	1	0
No Drill	0	0	1

Figure 1: Example of a policy window for the node ‘Drill’ of the *oil wildcatter example* (page 517)



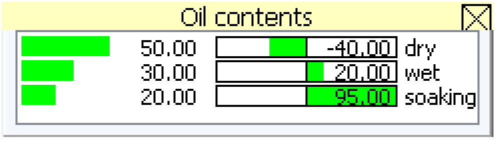
Drill

Seismic	cl	..	di
Drill	1	1	0
No Drill	0	0	1

Figure 2: Example of a policy window for the node ‘Drill’ with zero-probability configurations collapsed. *(Parent node is in state ‘cl’)*

4.2.17 Monitor Windows

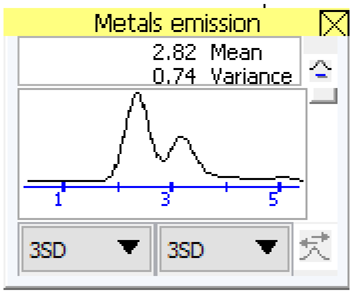
In *Run Mode* (page 123) you can open Monitor Windows for all or some of the nodes. You can choose the Toggle Monitors item from the Network Pane Menu which is activated when you press the right mouse button somewhere in the *network pane* (page 177) while working in Run Mode. You can also select the nodes and press the monitor icon on the top left of the *network pane* (page 177). There are three types of monitor windows (see Figure 1, Figure 2, and Figure 3) - one for each of the node types: *discrete chance node* (page 223), *continuous chance node* (page 224), and *decision node* (page 224). You cannot open a monitor window for a *utility node* (page 226).



Oil contents

50.00	-40.00	dry
30.00	20.00	wet
20.00	95.00	soaking

Figure 1: A monitor window for a discrete chance node.

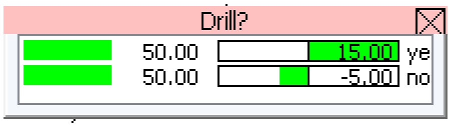


Metals emission

2.82 Mean
0.74 Variance

3SD
3SD

Figure 2: A monitor window for a continuous chance node.



Drill?

🟩	50.00	15.00	ye
🟩	50.00	-5.00	no

Figure 3: A monitor window for a decision node.

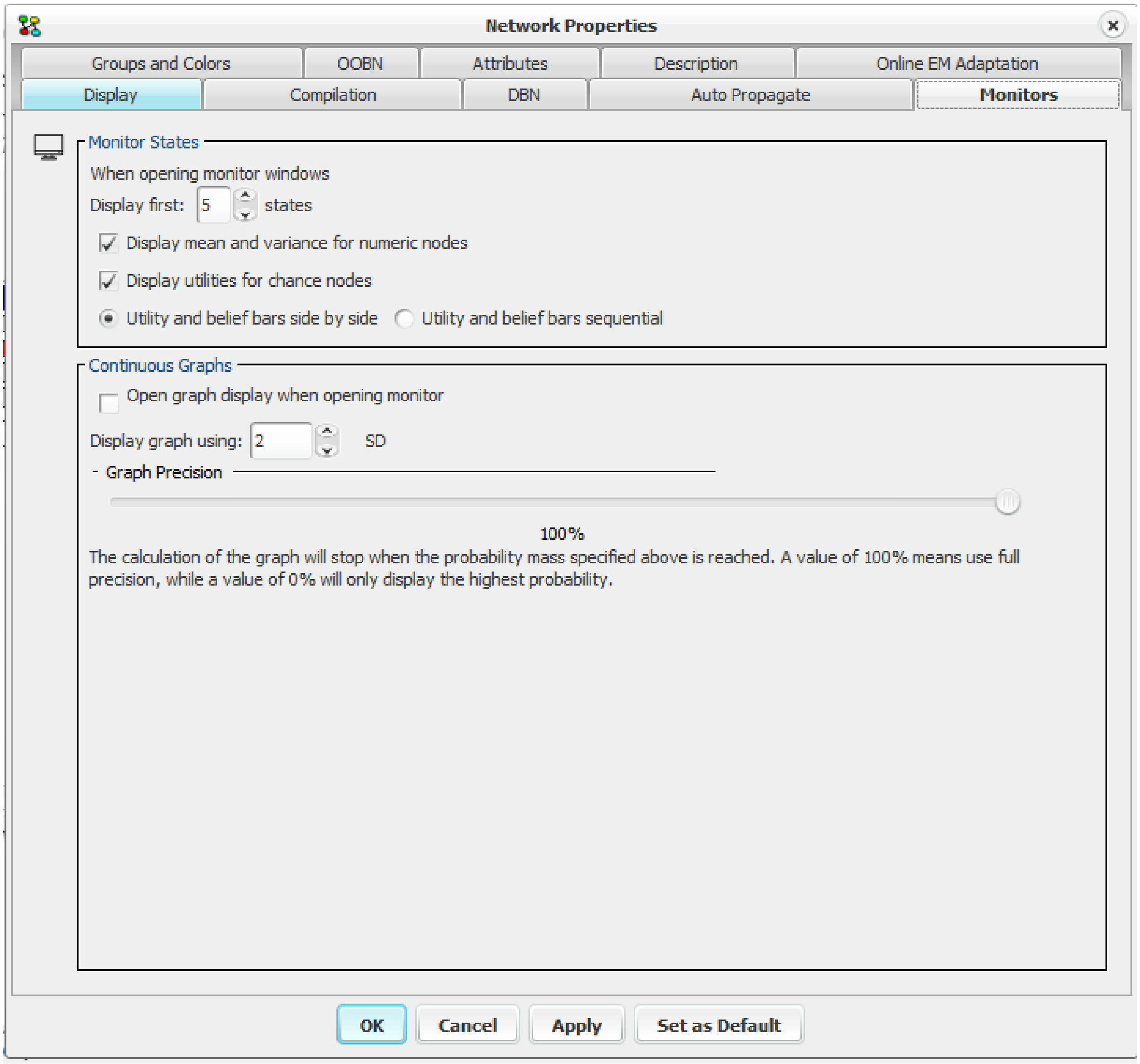
A monitor window shows the computed results (i.e., posterior marginal distribution and expected utility function) for a node:

- **Discrete chance nodes**: The probability of each state. In a LIMID both the probability and the expected utility of each state under the current strategy are shown.
- **Continuous chance nodes**: The mean, variance (SD), and the PDF (probability density function) graph.
- **Decision nodes**: The probability and expected utility of each action (decision option) under the current strategy.

Monitor windows for continuous chance nodes can open with or without showing the graph display below the pane showing mean and variance (depending on the settings in Network Properties as described below). You can open and close it by clicking the small button on the right of the mean and variance. You can chose different end points as the mean plus/minus an integral multiple of the variance from the two drop down lists below the pane containing the graph.

On the right side of the graph there is a slider which can be used to zoom in on the x-axis to spot local peaks on the graph. Just below this slider is a button which should be pressed after each propagation to update the graph if *auto update of graphs* (page 130) is not enabled. HUGIN does not automatically update the graphs because this could take a very long time for large networks.

You can customize the monitor windows to your needs in the Monitors tab of the *Network Properties* (page 178) dialog box. Figure 4 shows the Monitors tab.



Network Properties

Monitor States

When opening monitor windows

Display first: 5 states

☑ Display mean and variance for numeric nodes

☑ Display utilities for chance nodes

● Utility and belief bars side by side ○ Utility and belief bars sequential

Continuous Graphs

□ Open graph display when opening monitor

Display graph using: 2 SD

Graph Precision

100%

The calculation of the graph will stop when the probability mass specified above is reached. A value of 100% means use full precision, while a value of 0% will only display the highest probability.

Figure 4: The Network Properties dialog box showing the Monitors tab.

You can configure the monitors to show at most a certain number of states in the Monitor States group box. For continuous chance nodes, you can specify if you always want the monitor to open with the graph pane and you can specify the end points (in multiples of the variance) of the graph.

It is also possible to configure monitors to display means and variance for numeric nodes (See Figure 5), whether utilities should be displayed for chance nodes (LIMIDs) and whether Utilities and Belief bars should be arranged side by side or in sequence.

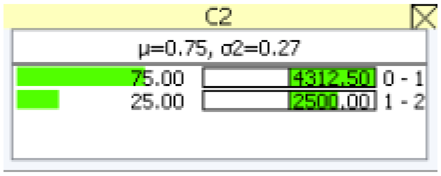


Figure 5: A Numeric node shown with Mean and Variance.

You can also specify the graph precision. Normally, you would want the graph as precise as possible but if you have a large network with many continuous chance nodes and poor performance, you might save some time if you decrease graph precision. Different display modes are available for the monitor windows. For discrete chance nodes there are four different modes:

Normal Mode. This is the default mode, showing the current marginal distribution for the node. Relative Difference Mode. This mode shows the relative change of each probability with respect to what it was before new evidence was inserted and propagated. The change is displayed in percent. Absolute Difference Mode. This mode shows the absolute change of each probability with respect to what it was before new evidence was inserted and propagated. Mixed Mode. This mode is similar to Normal Mode, except that below each bar is shown another (slimmer) bar indicating the probability before new evidence was inserted and propagated. Display mode is changed by right-clicking on the monitor, and by selecting “Display Mode” (Figure 6).

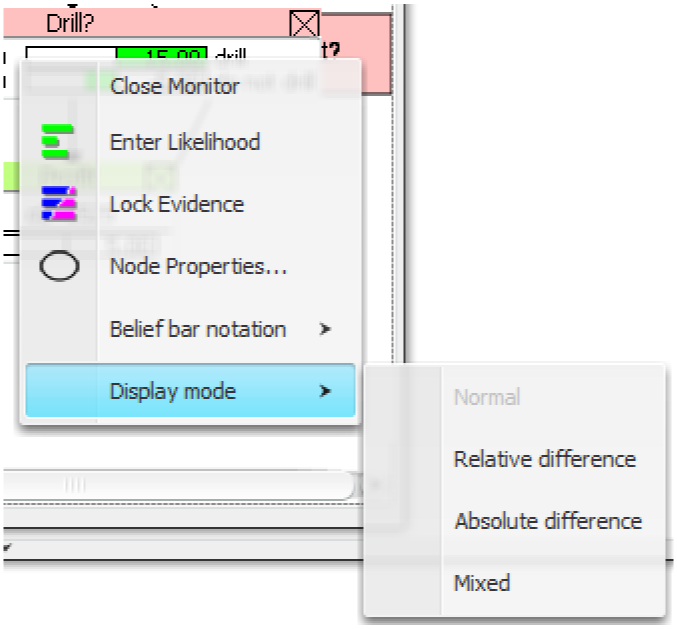


Figure 6: The Normal Mode before evidence has been entered and propagated.

Figure 7 shows a monitor in Normal Mode for a discrete chance node before evidence is entered and propagated.

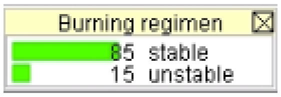
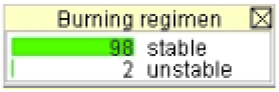


Figure 6: The Normal Mode before evidence has been entered and propagated.

Figure 8 shows the same monitor in Normal Mode after evidence has been entered and propagated.

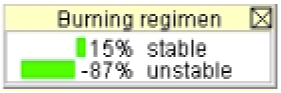


Burning regimen

98 stable
2 unstable

Figure 7: The Normal Mode after new evidence.

Figure 9 shows the monitor in Relative Difference Mode after the evidence has been entered and propagated.

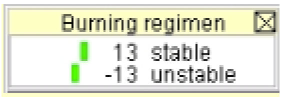


Burning regimen

15% stable
-87% unstable

Figure 9: The Relative Difference Mode after new evidence.

Figure 10 shows the monitor in Absolute Difference Mode after the evidence has been entered and propagated.

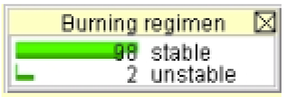


Burning regimen

13 stable
-13 unstable

Figure 10: The Absolute Difference Mode after new evidence.

Figure 11 shows the monitor in Mixed Mode after the evidence has been entered and propagated.



Burning regimen

98 stable
2 unstable

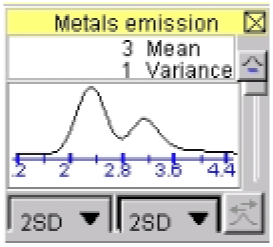
Figure 11: The Mixed Mode after new evidence.

For continuous chance nodes there are two different modes:

1. *Normal Mode*. This is the default mode, showing the current marginal density function for the node.
2. *Mixed Mode*. In this mode, both the new an the old marginal density functions are shown.

Please note that selection of display mode for continuous monitors has effect only when the monitor is expanded, showing the marginal density function (i.e., the display mode has no effect on the mean and variance values shown in the upper part of the monitor).

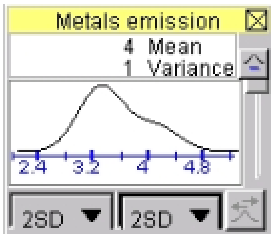
Figure 12 shows a monitor in Normal Mode for a continuous chance node before evidence is entered and propagated.



Metals emission

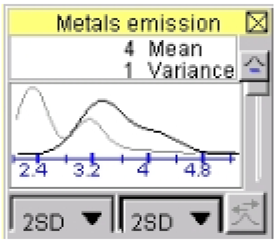
3 Mean
1 Variance

Figure 12: The Normal Mode before evidence has been entered and propagated.


Metals emission

4 Mean
1 Variance

Figure 13: The Normal Mode after new evidence.

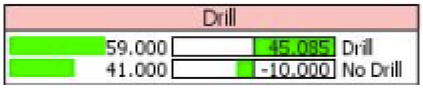

Metals emission

4 Mean
1 Variance

Figure 14: The Mixed Mode after new evidence.

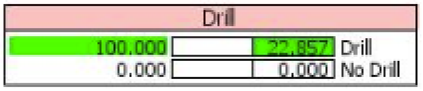
For decision nodes there are two different modes:

1. *Normal Mode*. This is the default mode, showing the probability and expected utility for the each decision option for the node.
2. *Absolute Difference Mode*. This mode shows the absolute change in the probability and the expected utility for each decision option with respect to what it was before new evidence was inserted and propagated.


Drill

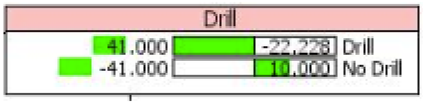
59.000 45.085 Drill
41.000 -10.000 No Drill

Figure 15: The Normal Mode before evidence has been entered and propagated.


Drill

100.000 22.857 Drill
0.000 0.000 No Drill

Figure 15: The Normal Mode after new evidence.


Drill

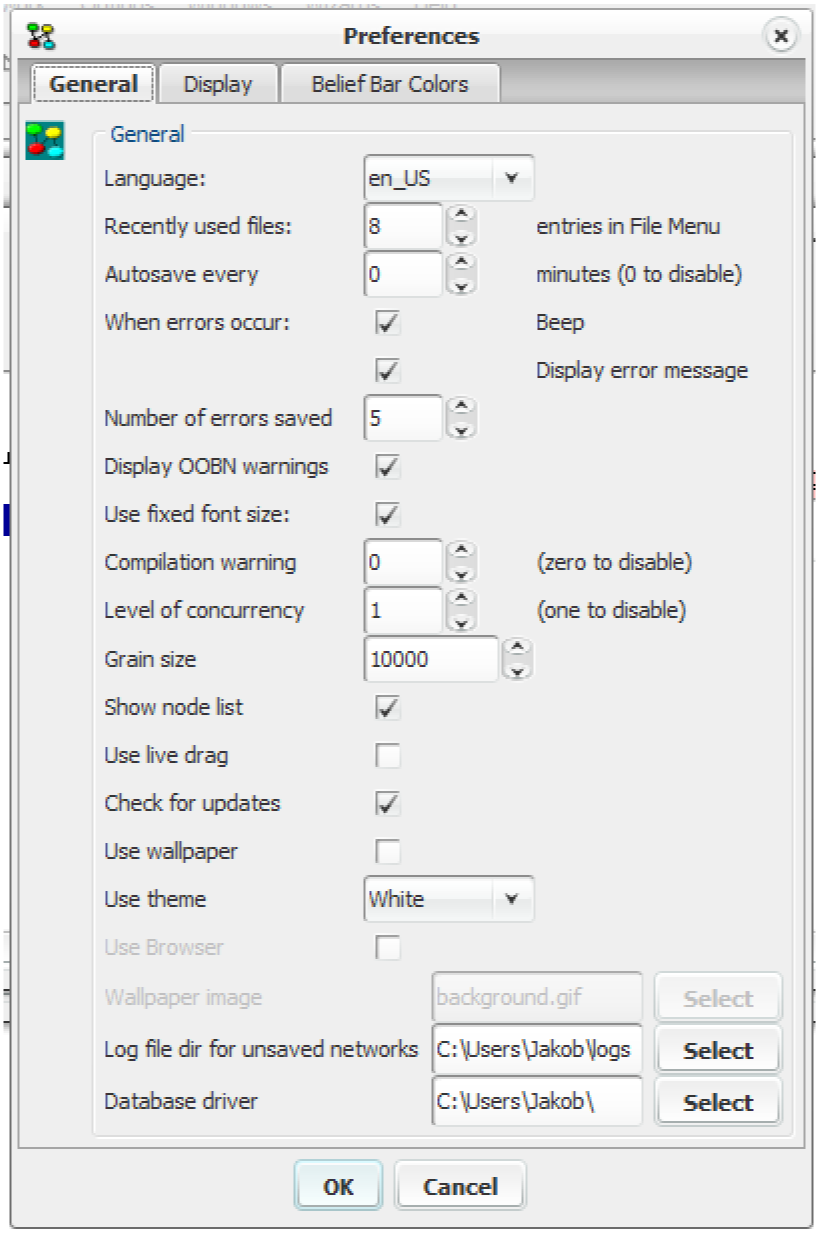
41.000 -22.228 Drill
-41.000 10.000 No Drill

Figure 16: The Absolute Difference Mode after new evidence.

Display modes are selected through the popup menu that appears when right-clicking a monitor window.

4.2.18 Preferences

The preferences dialog box is opened from the *Options Menu* (page 193). It allows the user to customize the HUGIN user interface. Figure 1 and Figure 2 shows the preferences dialog box with each of the two tabs chosen.



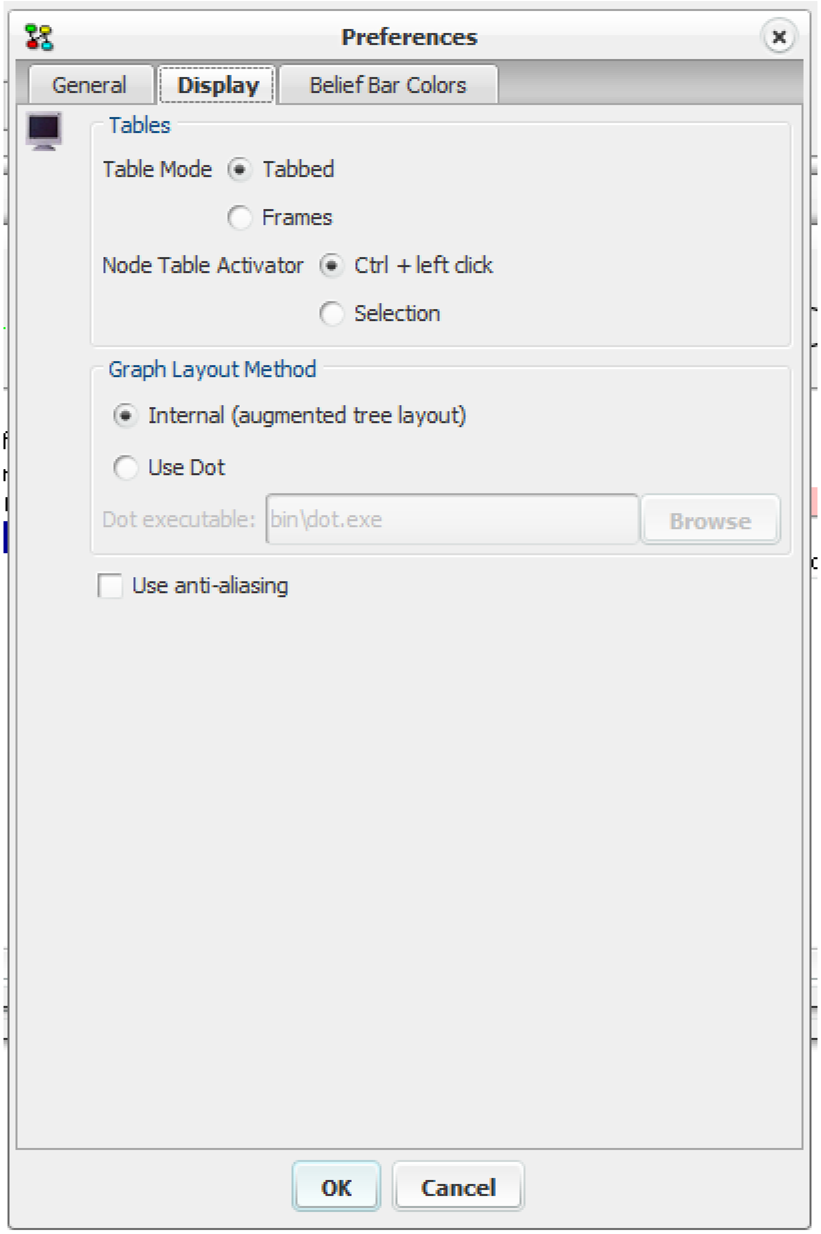
Language: en_US
Recently used files: 8 entries in File Menu
Autosave every 0 minutes (0 to disable)
When errors occur: ☑ Beep
☑ Display error message
Number of errors saved 5
Display OOBN warnings ☑
Use fixed font size: ☑
Compilation warning 0 (zero to disable)
Level of concurrency 1 (one to disable)
Grain size 10000
Show node list ☑
Use live drag ☐
Check for updates ☑
Use wallpaper ☐
Use theme White
Use Browser ☐
Wallpaper image background.gif
Log file dir for unsaved networks C:\Users\Jakob\logs
Database driver C:\Users\Jakob\

Figure 1: The preferences dialog box with the General Options tab chosen.

In the General Options tab (Figure 1) you can specify

- Choose a language. Currently supporting English and Japanese.
- how many files you want to be shown in the *recently used file* (page 445) list of the *File Menu* (page 191),
- if and how often you want to auto-save (auto-save file is named <netname>_AUTOSAVED.net),
- if you want a sound indication when an error occurs,

- if you want error messages to appear in popup dialog boxes or just in the status bar of the *Main Window* (page 186)
- the number of most recent error messages to be kept in a history list of errors (accessible through the Show Last Error item of the *View Menu* (page 188))
- if you want warnings to appear when performing operations on OOBNs that may lead to a situation of inconsistency (making it impossible to load the OOBN later on) - such operations can be performed when only sub-networks of an *OOBN* (page 51) have been loaded (useful when several people work on the same overall project), and could, for example, be operations like renaming a class, an *input node* (page 227), or an *output node* (page 228), changing the *node category* (page 209) or kind of an input node, changing the number of states of an output node, or delete an input node or an output node,
- if you want HUGIN to use fixed font size (it may be a good idea to disable this property, if you are running HUGIN on a small screen with a high resolution).
- if you want HUGIN to give a compilation warning when the compilation process may require a large amount of memory or involve a large number of computations when generating tables from expressions,
- if you want the *Node List Pane* (page 243) to be displayed in *Run Mode* (page 123),
- if you want *network windows* (page 182) to be repainted while dragging for resizing,
- if you want HUGIN to check for updates on the internet each time it starts up,
- if you want to use an image file as wallpaper on the background of the *Main Window* (page 186),
- if you want to connect to a database using a JDBC driver (in *Analysis Wizard* (page 259), Learning Wizard etc.)select the folder that includes the database JDBC drivers.
- the theme you want to use - currently there are six themes, ‘dark’, ‘white’, ‘classy’, ‘aluOxide’, ‘green’, and ‘blue’.
- if you want a your default browser or an in-GUI view when browsing help from the GUI



Tables

Table Mode ● Tabbed

○ Frames

Node Table Activator ● Ctrl + left click

○ Selection

Graph Layout Method

● Internal (augmented tree layout)

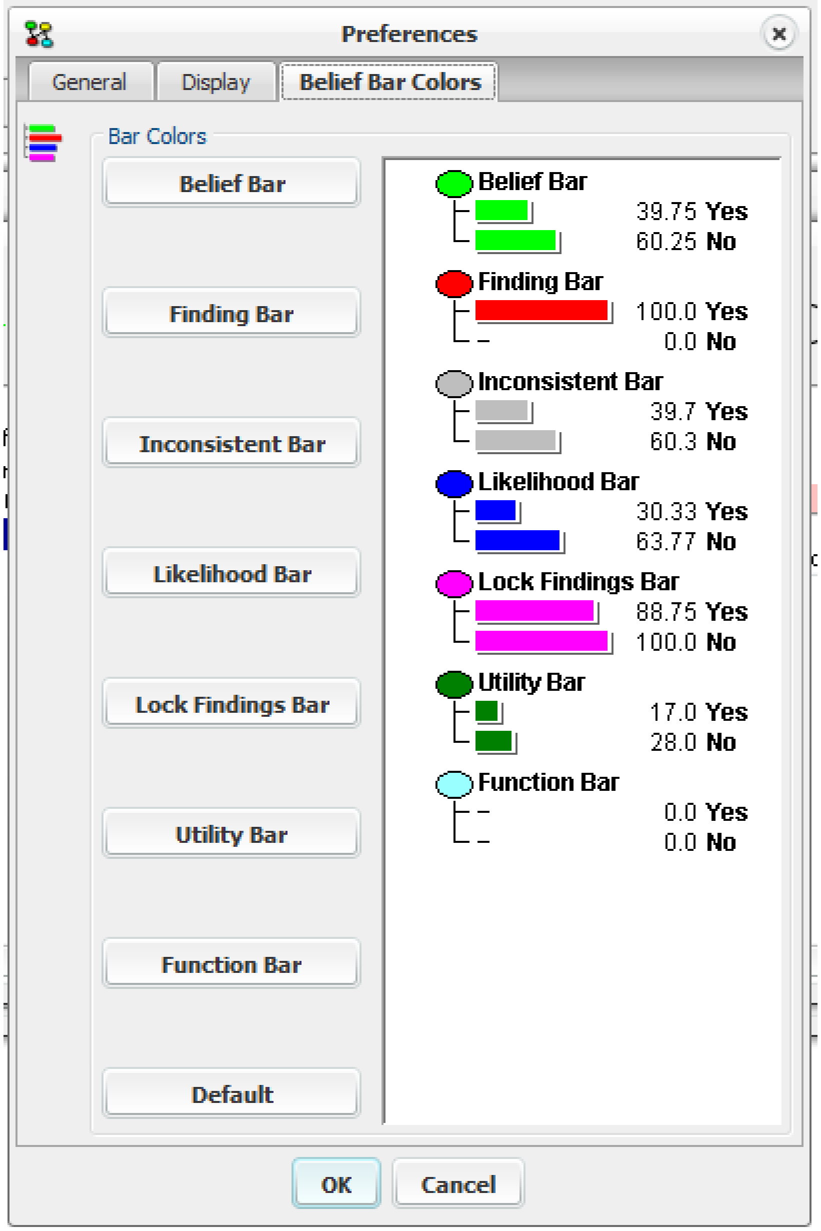
○ Use Dot

Dot executable: bin\dot.exe

□ Use anti-aliasing

Figure 2: The preferences dialog box with the Display Options tab chosen.

In the Display Options tab (Figure 1) you can specify: * the way in which *tables* (page 241) are displayed * how to get a nodes table shown in the table pane * which *graph layout algorithm* (page 146) to use



```
Belief Bar
   39.75 Yes
   60.25 No
Finding Bar
   100.0 Yes
     0.0 No
Inconsistent Bar
    39.7 Yes
    60.3 No
Likelihood Bar
   30.33 Yes
   63.77 No
Lock Findings Bar
   88.75 Yes
   100.0 No
Utility Bar
    17.0 Yes
    28.0 No
Function Bar
     0.0 Yes
     0.0 No
```

Figure 3: The preferences dialog box with the Belief Bar Colors tab chosen

In the Belief Bar Colors tab (Figure 3) you can change the default colors for the bars showing beliefs, findings, inconsistencies, likelihoods, and locked findings used in the *Node List Pane* (page 243) and the *monitor windows* (page 194).

Figure 4 shows the color select dialog window which appears when you click any of the buttons in the Belief Bar Colors tab.

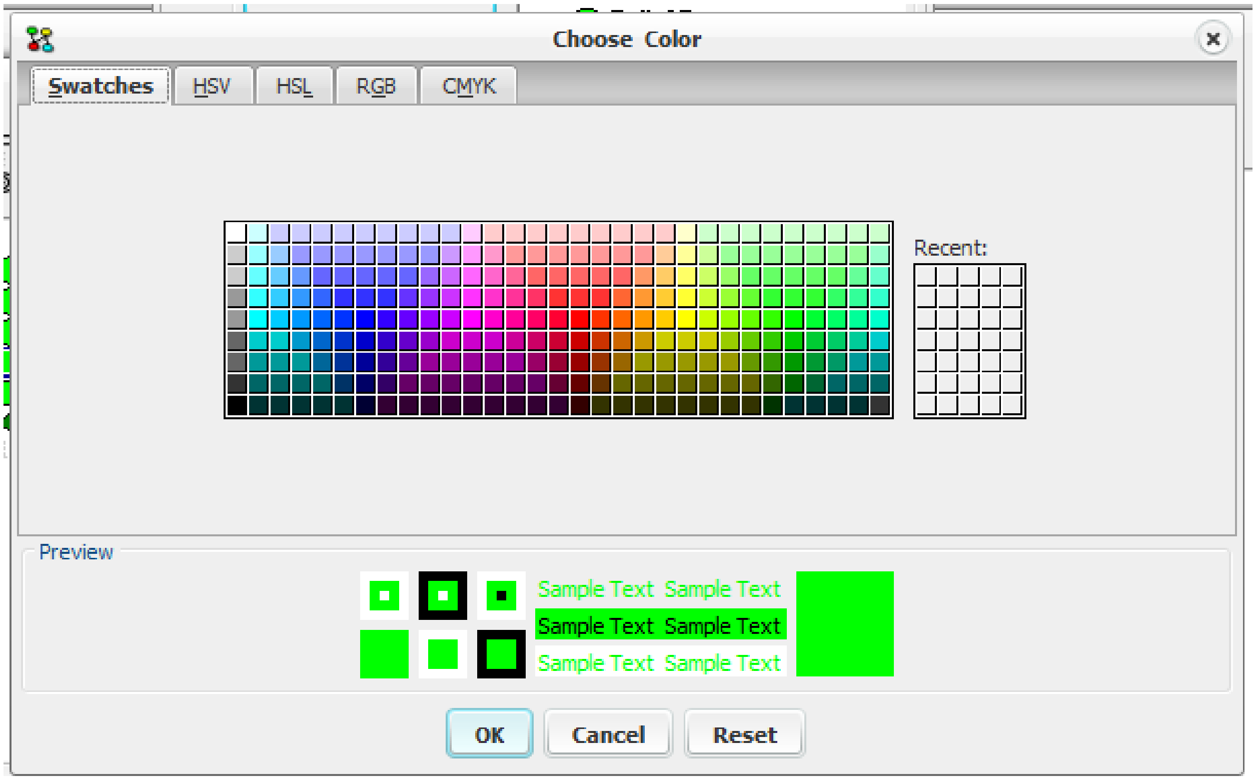


Figure 4: The color selection dialog box

4.2.19 Window List

At the bottom of the *Window Menu* (page 189) you can find a list of the currently open *network windows* (page 182). If you select one of these windows, it will be the currently selected window and appear on top of all other network windows.

4.2.20 HUGIN Web Site

From the *Help Menu* (page 191) you can select the *HUGIN Web Site item* (Figure 1). This opens your registered browser and takes you to the HUGIN WWW pages (http://www.hugin.com).

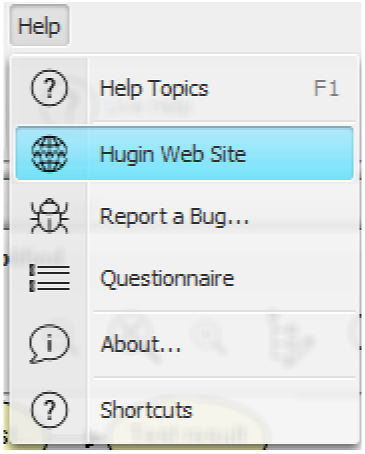


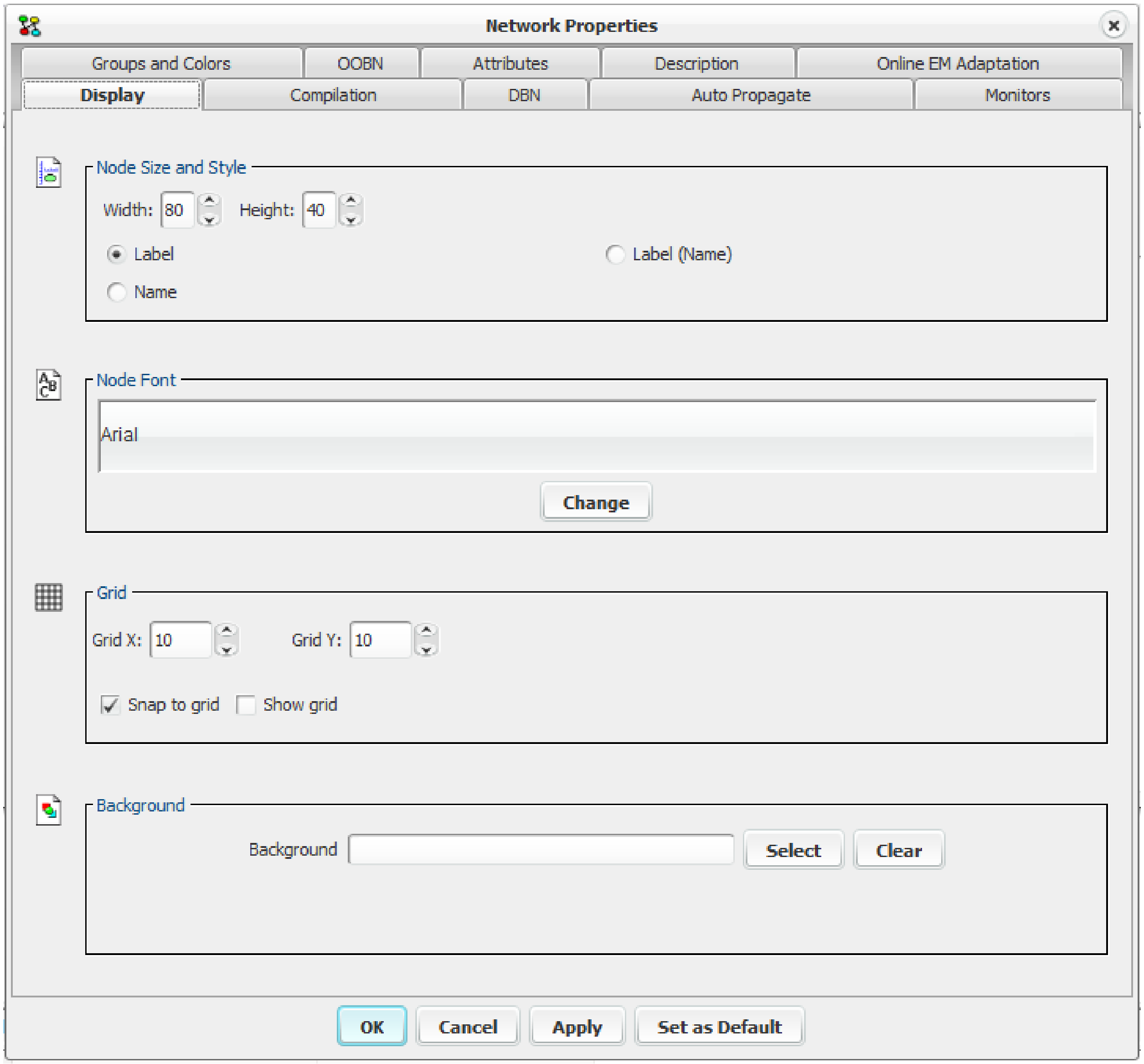
Figure 1: The menu item to open HUGIN web site.

4.2.21 General Help

You are currently looking at the HUGIN general help pages. These pages can be accessed from the HUGIN Graphical User Interface by selecting the Help Topics item of the *Help* (page 191) menu.

4.2.22 Display

In the *Network Properties* (page 178) dialog box you find the Display tab which allows you to change different parameters in the display of the network. Figure 1 shows the Display tab.



Width: 80 Height: 40

● Label ○ Label (Name)

○ Name

Arial

Grid X: 10 Grid Y: 10

☑ Snap to grid □ Show grid

Background

Figure 1: The Network Properties dialog box showing the Display tab.

At the top of the Display tab you can specify the width and height of the nodes in the *Network Pane* (page 177). Also, there are three radio buttons for selecting how the textual identity (i.e., *name* (page 210) and/or *label* (page 210)) of a node should be displayed.

The font used for displaying the textual identity can be selected in the middle region.

At the Grid area you can set up a grid used in the *Network Pane* (page 177) to help arranging the nodes. Keyboard shortcuts are available for controlling grid functions:

- Ctrl + . (dot): Toggles the value of Show Grid.
- Ctrl + , (comma): Toggles the value of Snap to Grid. When set, this makes new nodes be placed with the center at the nearest grid point.
- Shift + , (comma): Snaps the selected nodes to the nearest grid points.

At the bottom area you can set a background image for the network panel.

4.2.23 Groups and Colors

You can specify different node groups as well as link groups in HUGIN which can be used to group nodes and links of a network. Each group has assigned a color so that in the display of the network it is easy to distinguish nodes and links of different groups. You can specify/edit the node groups and link groups of your network in the Groups and Colors tab of the Network Properties dialog box.

Node Groups To modify the node groups of the current network open the the Network Properties dialog, navigate to the 'Groups and Colors' tab followed by the 'Node Groups' tabs as shown in Figure 1.

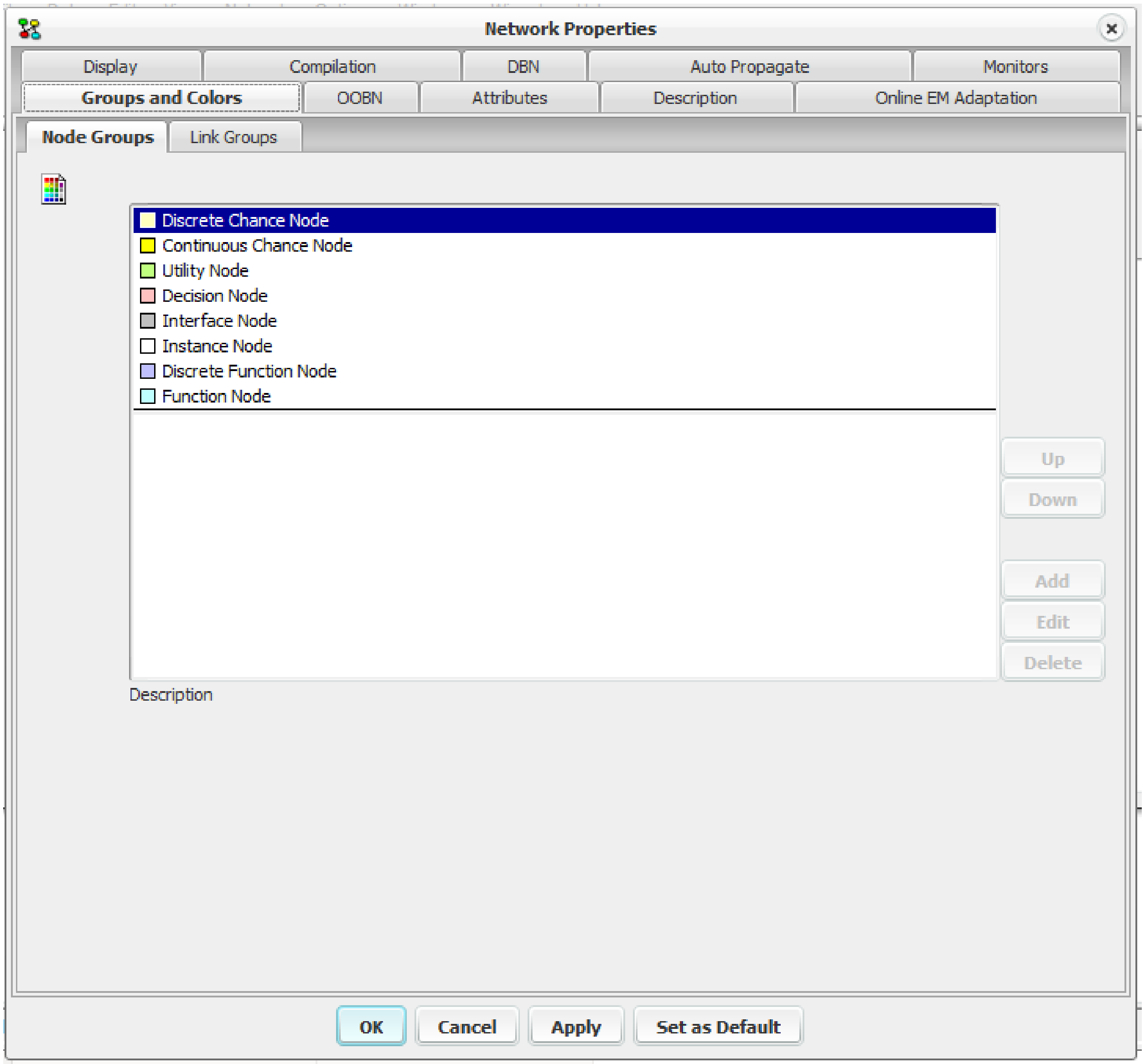


Figure 1: The Network Properties dialog box showing the Node Groups tab.

When you use HUGIN, you always use eight default groups (maybe without recognizing them as groups). Each of the different types of nodes have their own default group: discrete chance nodes, continuous chance nodes, utility nodes, decision nodes, interface nodes, instance nodes, discrete function nodes, and finally function nodes.

You add a new (user defined) group by clicking the 'Add' button and specifying a group name and a group color in the pop-up window. You can also rename a group, delete it, or move it up and down in the node groups list. For the eight default node groups you are allowed to alter the color only.

At the bottom of the Groups and Colors tab, you can add a description to the currently selected user defined node group.

Node Groups The modification of link groups is similar to modification of node groups. Select the 'Node Groups' tab as shown in Figure 2.

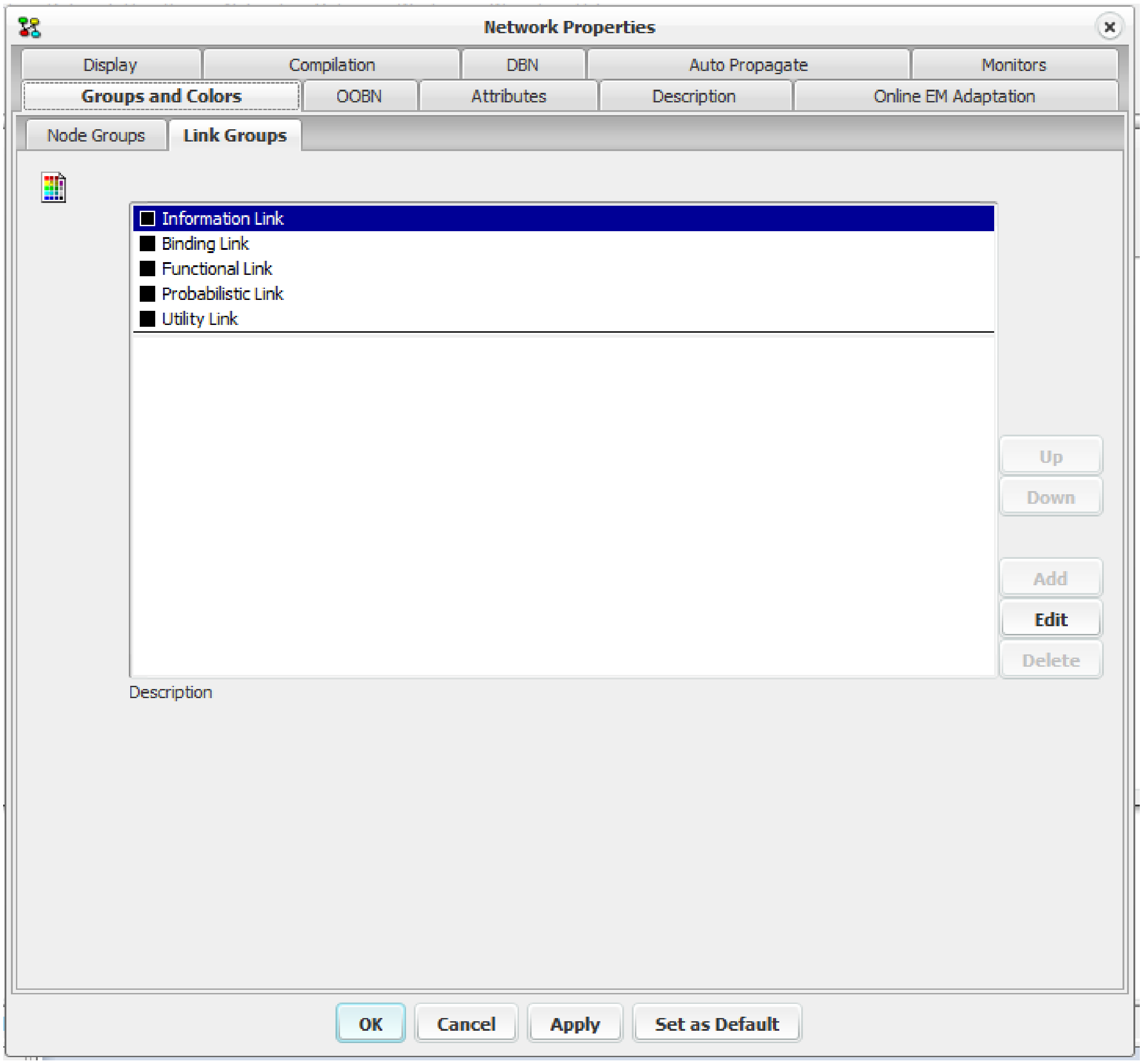


Figure 2: The Network Properties dialog box showing the Link Groups tab.

The node group named 'Default Link' is the group a link belongs to when it has not been assigned any other group. For this link group you are allowed to change the color. User defined link groups are added to, modified in and deleted from the list below. To add a new link group click the 'Add' button and fill in a name and select a color in the pop-up window. You can rename a group, delete it, or move it up and down in the link groups list. Use the description field to enter your own description of any selected user defined link group.

Assigning selected Nodes or Links to Groups You can assign one or more nodes or links in the Network Pane to a group by selecting them (hold down shift and left-click on the links or nodes you wish to select) and clicking the right mouse button which opens a menu. From this menu, use the Assign to Group item to assign the selected nodes or links to a group. The menu is shown in Figure 3. This way you can also remove a number of selected nodes or links from a specific group by selecting the No Group sub item of the Assign to Group item.

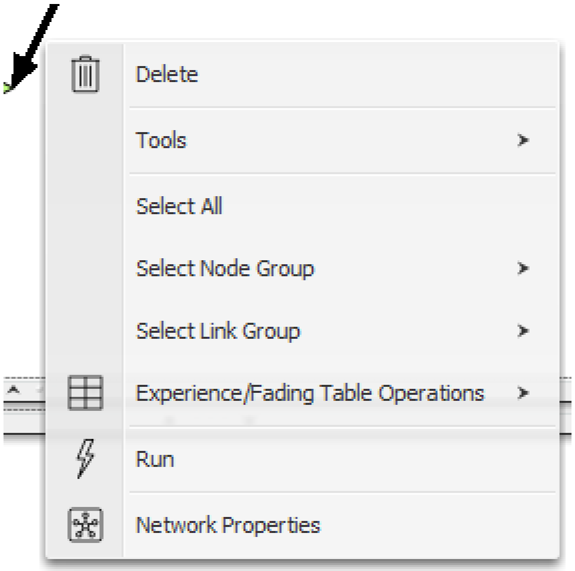


Figure 3: Menu showing the two menu items ‘Select Node Group’ and ‘Select Link Group’.

Note that you cannot assign nodes to link groups or links to node groups, so when a number of links are selected the menu allows you to assign them to a link group and when a number of nodes are selected the menu allows you to assign them to a node group.

You can only assign nodes or links to groups in Edit Mode.

Once, a number of nodes or links have been assigned to a group all the nodes or links of this group can be selected in one operation by clicking the right mouse button which opens a menu. From this menu, use the Select Node Group item to select a node group or use the Select Link Group to select a link group.

Node groups can be selected in this way in both Edit Mode and Run Mode, while links can only be selected in this way in Edit Mode.

Back

4.3 Nodes and Tables

4.3.1 Node

The *nodes* are one of the basics in constructing a HUGIN network (*Bayesian network* (page 345), *LIMID* (page 345), or *object-oriented network* (page 346)). Currently, HUGIN has five types of nodes that differ in appearance and semantics.

- *Discrete Chance Nodes* (page 223) - represent random variables with a discrete finite state space.
- *Continuous Chance Nodes* (page 224) - represent random variables with a continuous infinite satet space having a Gaussian (normal) distribution.
- *Decision Nodes* (page 224) - represent decisions or actions with a discrete state space.
- *Utility Nodes* (page 226) - represent the utility function one or more decisions are to be compared on.

- *Instance Nodes* (page 224) - represent instances of other networks, enabling object-oriented specification of networks (both Bayesian networks and influence diagrams).
- *Function Nodes* (page 225) - computes a value based on expressions

The graph of a network consist of the nodes and the *links* (page 147).

4.3.2 Node Name (Node Identifier)

The *node name* (or *node identifier*) is a unique name which all nodes must have. Initially, HUGIN assigns a name to a new node. If you should later want to change this name, it can be done in *Edit Mode* (page 124) in the *Tables Pane* (page 239) or in the *Node Properties* (page 212) dialog box.

Please beware of the different between the node name and the *node label* (page 210).

4.3.3 Node Label

The *node label* is used by HUGIN to display as the label on the nodes. It does not need to be unique in the actual network. Initially, a node has no node label and HUGIN uses the *node name* (page 210) as the label. If you should want to assign a new node label to a node, it can be done in *Edit Mode* (page 124) in the *Tables Pane* (page 239) or in the *node properties* (page 212) dialog box.

Please beware of the different between the node label and the *node name* (page 210).

4.3.4 Node Monitor Toggle

The *Node Monitor Toggle* is used in the *run mode* (page 123) *network window* (page 182). A node can either be viewed as a monitor or as a node representation when in run mode. The toggle can be activated by selecting a single node or a group of nodes, and pressing the toggle button shown in figure 1 or the menu item shown in figure 2.

Figure 1: The Node Monitor Toggle button.

Alternatively, you can right click on the node/monitor in the network frame or in the node tree, and press the menu item shown in figure 2.

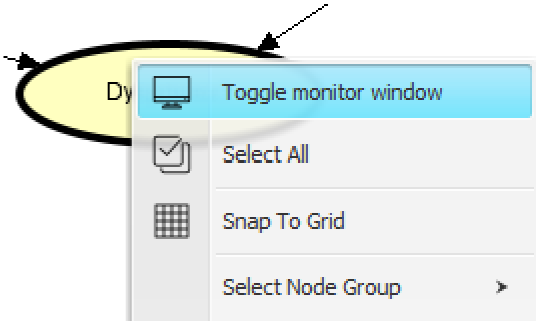


Figure 2: You can also right click on a node/monitor to and toggle through the menu item.

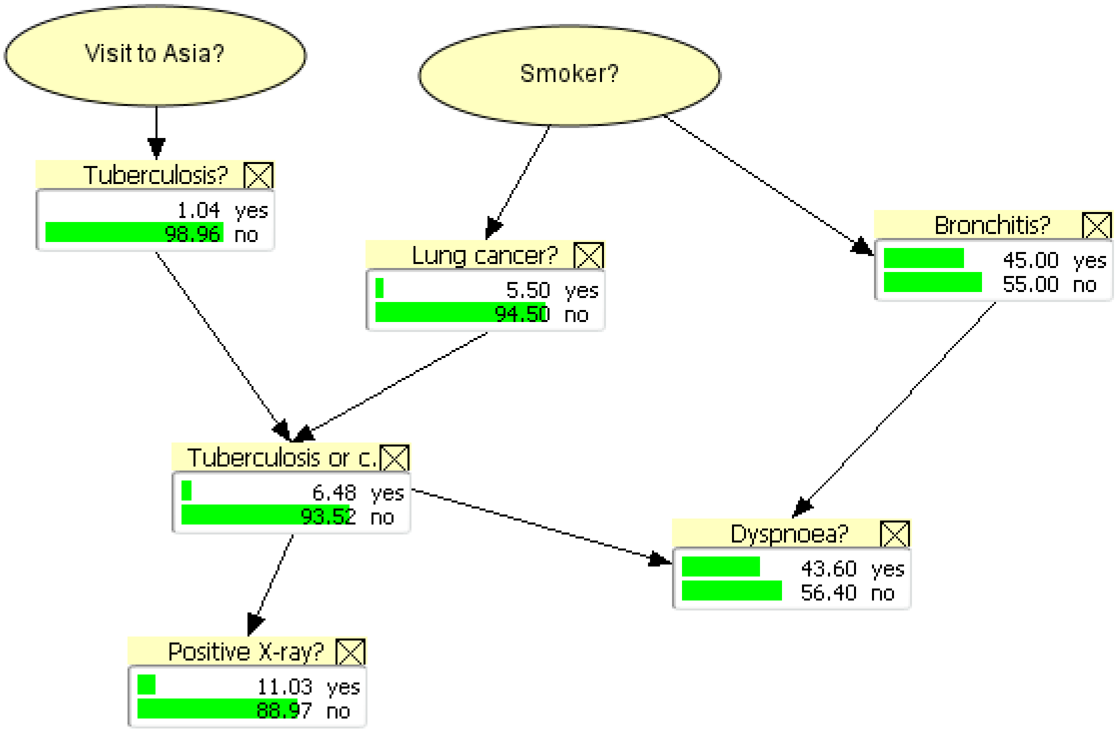


Figure 3: An example of a network in run mode where some nodes are displayed with monitors.

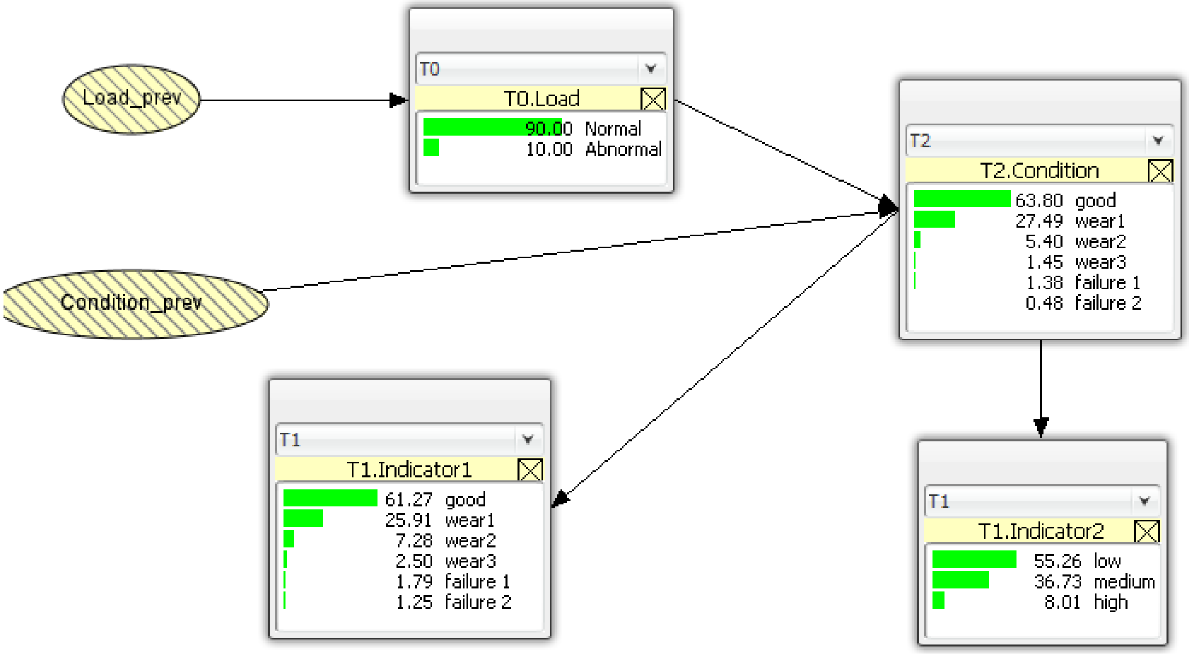


Figure 4: An example of a DBN with monitors on different time slices.

4.3.5 Node Properties

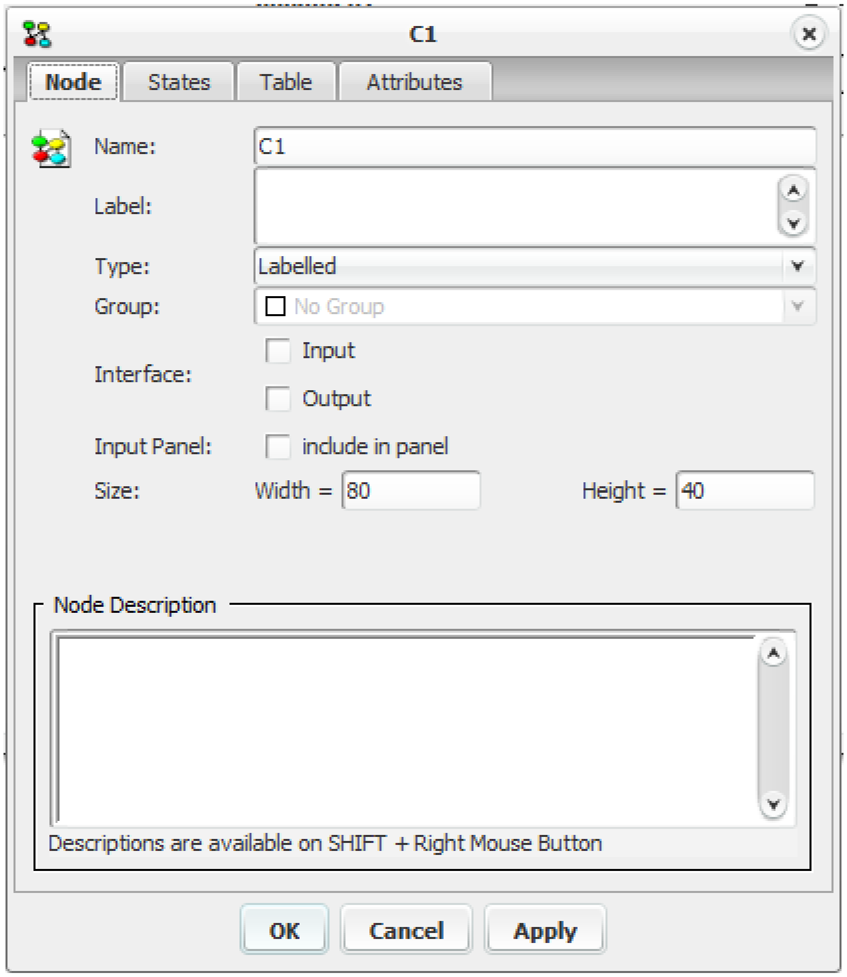
You can open the *node properties* dialog box by selecting a node and then choosing the Node Properties item of the *Edit Menu* (page 192). You can also just select a node and somewhere within the *Network Pane* (page 177) press the right mouse button. This opens a menu from which you can choose the Node Properties item.

Figure 1 shows the node properties dialog box where the user can add a description for the node, as plain text.

Figure 1a shows the node properties dialog box with hyperlinks enabled in the node description. The difference between figure 1 and figure 1a, is the Edit check box in figure 1a which switches the description box between edit mode and display mode . For information on how to enable html, edit in html mode and create hyperlinks go to *Network Properties Panel* (page 178)

The Node properties pane is organized in four tabs which are documented separately:

- *Node tab* (page 216)
- *States tab* (page 235)
- *Table tab* (page 234)
- *Attributes tab* (page 219)



Name: C1

Label:

Type: Labelled

Group: □ No Group

Interface: □ Input

□ Output

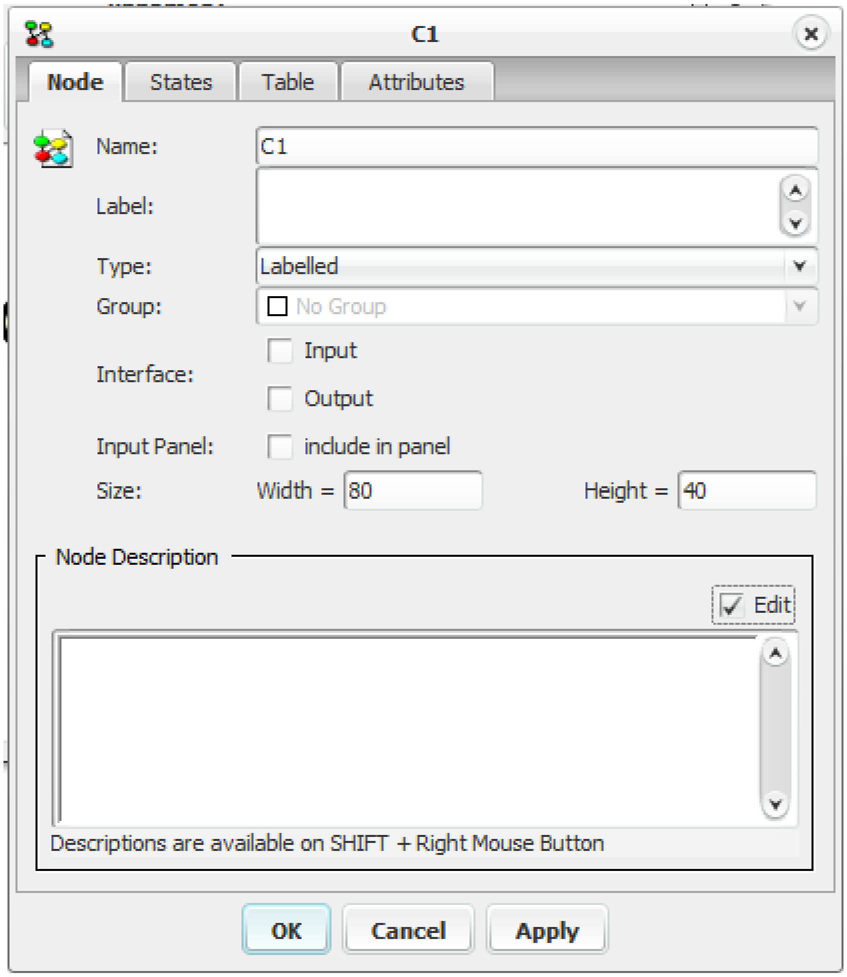
Input Panel: □ include in panel

Size: Width = 80 Height = 40

Node Description

Descriptions are available on SHIFT + Right Mouse Button

Figure 1: The node properties dialog box showing the Node tab.



Name: C1

Label:

Type: Labelled

Group: □ No Group

Interface: □ Input

□ Output

Input Panel: □ include in panel

Size: Width = 80 Height = 40

Node Description

☑ Edit

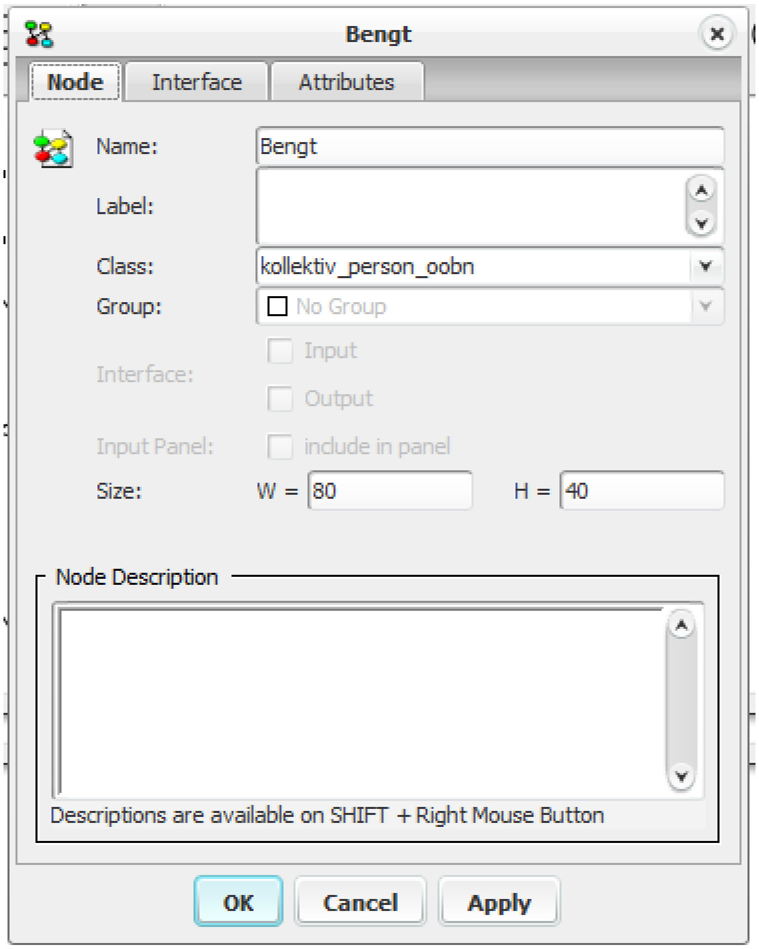
Descriptions are available on SHIFT + Right Mouse Button

Figure 1a: The node properties dialog box showing the Node tab when HTML is selected in Network Properties Pane.

Figure 2 shows the node properties dialog box for an *Instance Node* (page 224). It is organized in three tabs which are documented separately:

- *Node tab* (page 216)
- *Interface tab* (page 222)
- *Attributes tab* (page 219)

Figure 2: The node properties dialog box showing the Node tab of an Instance Node.



Name: Bengt

Label:

Class: kollektiv_person_oobn

Group: □ No Group

Interface: □ Input □ Output

Input Panel: □ include in panel

Size: W = 80 H = 40

Node Description

Descriptions are available on SHIFT + Right Mouse Button

4.3.6 Node Locator

The Node Locator provides an easy way to select nodes based upon (parts of) their name. When first clicked, the Node Locator will show a list of all nodes in the network (Figure 1). It is then possible to select any of these by clicking on them or by using the up/down keys to navigate the list, and enter to select a node.

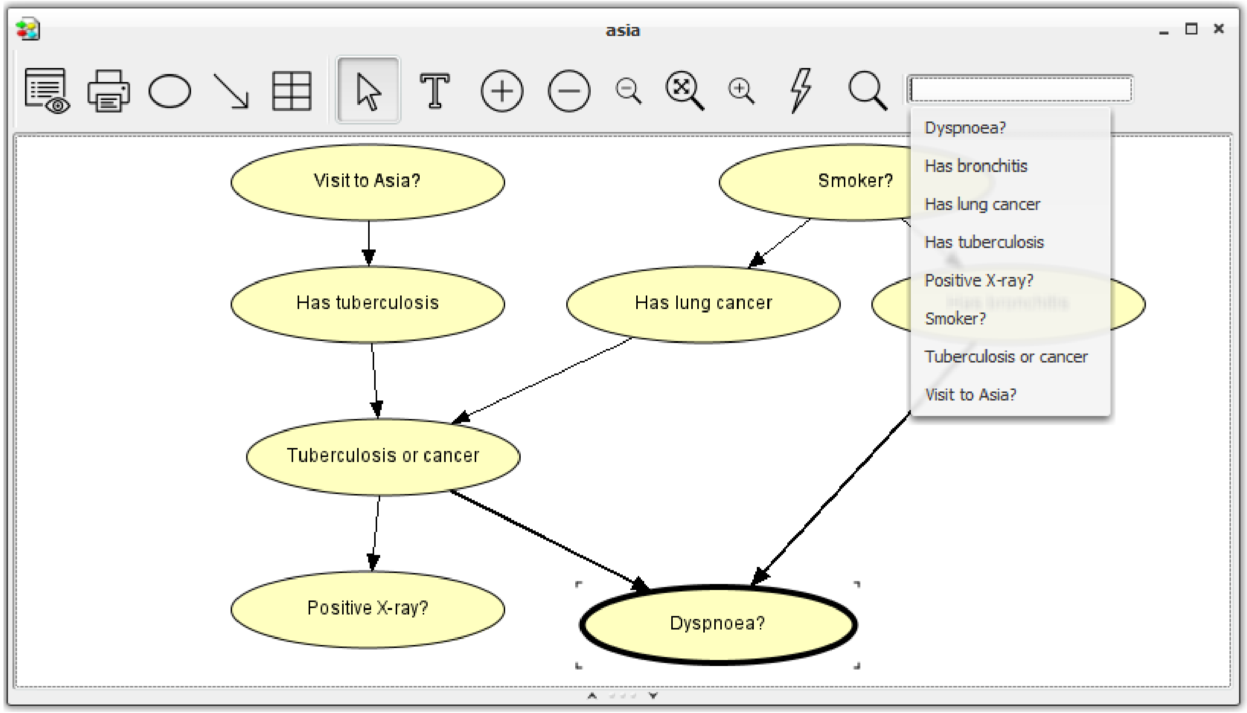


Figure 1: The Node Locator showing all nodes in the network.

It is possible to restrict the Node Locator to showing only nodes whose name matches an entered regular expression (Figure 2).

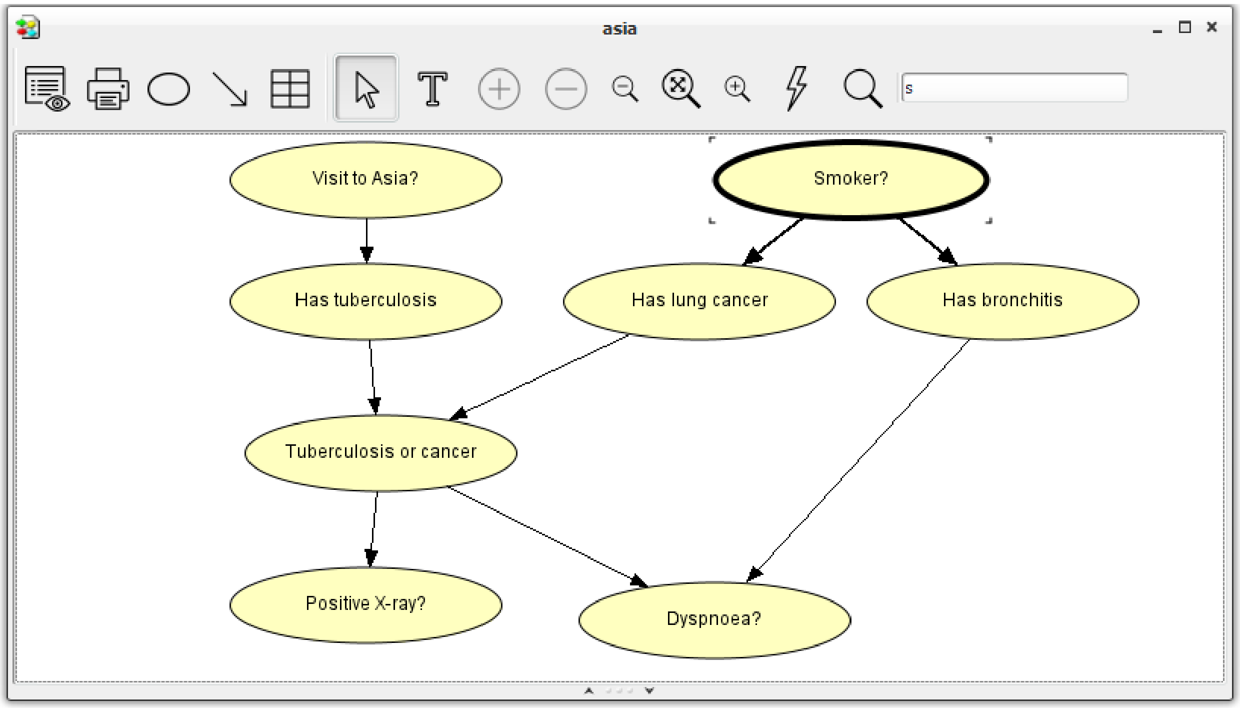


Figure 2: The Node Locator showing only the nodes matching the expression.

When specifying the expression, the following special characters can be used:

- *: Match everything a number of times
- .: Match everything one time
- \w: Match a word character [a-zA-Z_0-9]
- \W: Match a non-word character: [^\w]
- \d: Match a digit: [0-9]
- \D: Match a non-digit: [^0-9]
- \s: Match a whitespace character: [\t\n\x0B\f\r]
- \S: Match a non-whitespace character: [^\s]

Note, in particular, that it is not possible to match a given character a number of times using the "*" character. I.e., the expression a*b will match abbbb as well as aaaab. Note further, that the expression is interpreted as having a trailing "*" character. That is, the written expression must only match the beginning of the node name for the node to be included in the list.

4.3.7 Node Type

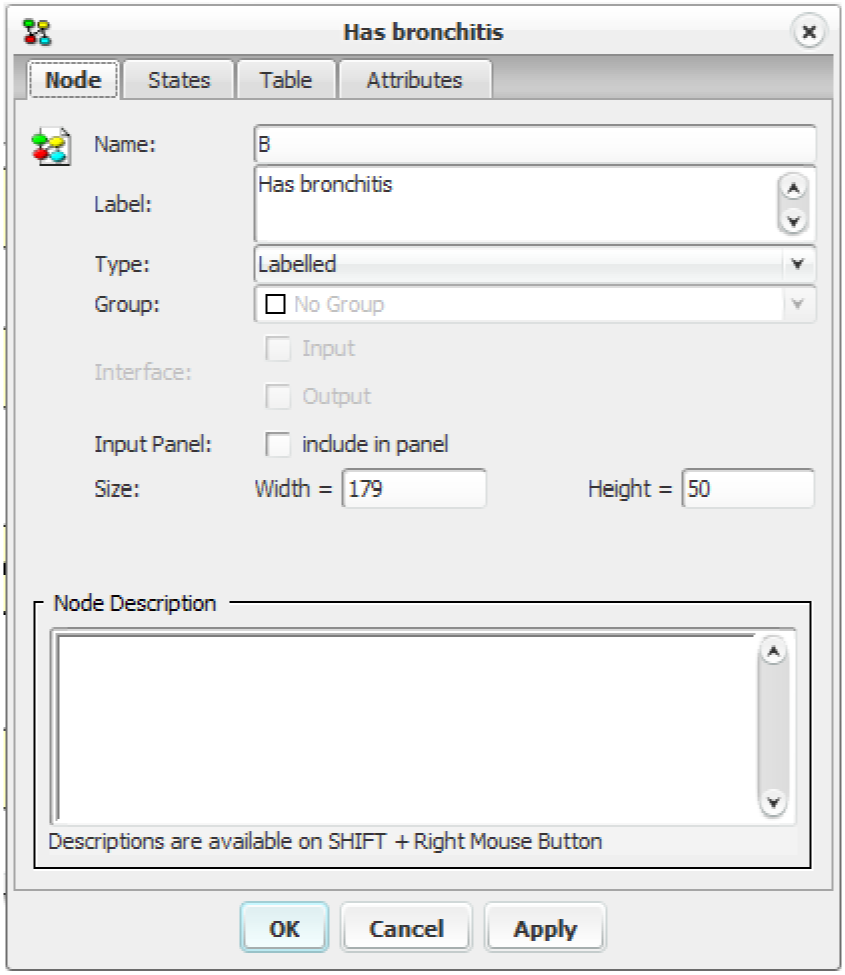
To be able to use nodes as arguments in *expression* (page 150) functions, node types are introduced for discrete chance nodes and decision nodes. Below are listed the different node types:

- Labelled - Each state has a label. This is similar how all discrete nodes looked in old versions of HUGIN.
- Boolean - Has two states: false and true.
- Numbered - Each state has a numeric value. This means that you can make expressions define the sum of two numbered nodes. You can also specify standard discrete distribution functions for numbered nodes if you fit the state values to what is required for the distribution function (typically: 0, 1, 2,…).
- Interval - Each state represents an interval on the real axis. The entire set of states span a continuous interval on the real axis. This is used to approximate a continuous random variable. You can specify standard continuous distribution functions for interval nodes, or you can use them in arithmetic functions.

The type of a node can be specified through its *node table* (page 241) via the Functions menu or via the Type menu of the Node tab of the *Node Properties* (page 212) dialog.

4.3.8 Node Tab

The *Node Tab* of the *Node Properties* (page 212) dialog allows the user to edit different attributes of the nodes of a HUGIN domain. The Node Tab is shown in Figure 1.



Has bronchitis

Name: B

Label: Has bronchitis

Type: Labelled

Group: □ No Group

Interface: □ Input □ Output

Input Panel: □ include in panel

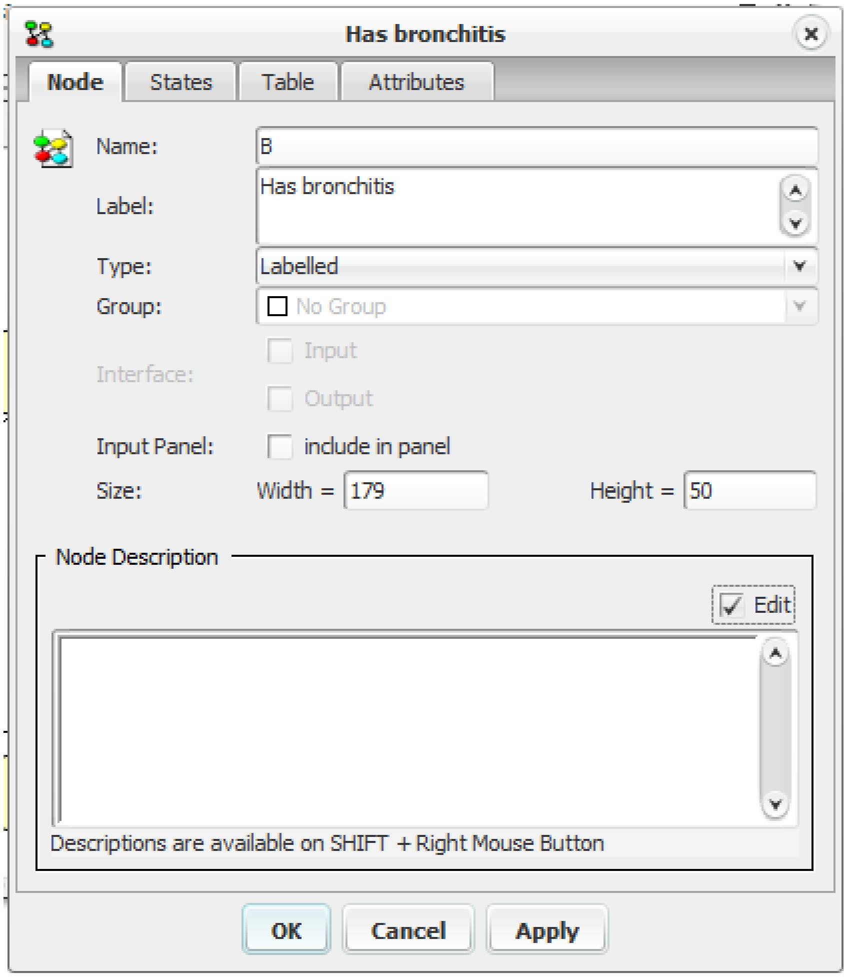
Size: Width = 179 Height = 50

Node Description

Descriptions are available on SHIFT + Right Mouse Button

Figure 1: The node properties dialog box showing the Node Tab.

In the top section of the Node tab you can specify the *name* (page 210), *label* (page 210), *type* (page 216), *group* (page 207), *interface type* (page 229) (if any), and size of the selected node. Please note that, except for size, these can only be specified in *Edit Mode* (page 124). Below these fields, a free-text description for the node can be provided. Figure 2 shows the node properties dialog box with hyperlinks enabled in the node description. The difference between figure 1 and figure 2, is the Edit check box in figure 2 which switches the description box between edit mode and display mode . For information on how to enable html, edit in html mode and create hyperlinks go to *Network Properties Panel* (page 178).



Has bronchitis

Name: B

Label: Has bronchitis

Type: Labelled

Group: □ No Group

Interface: □ Input □ Output

Input Panel: □ include in panel

Size: Width = 179 Height = 50

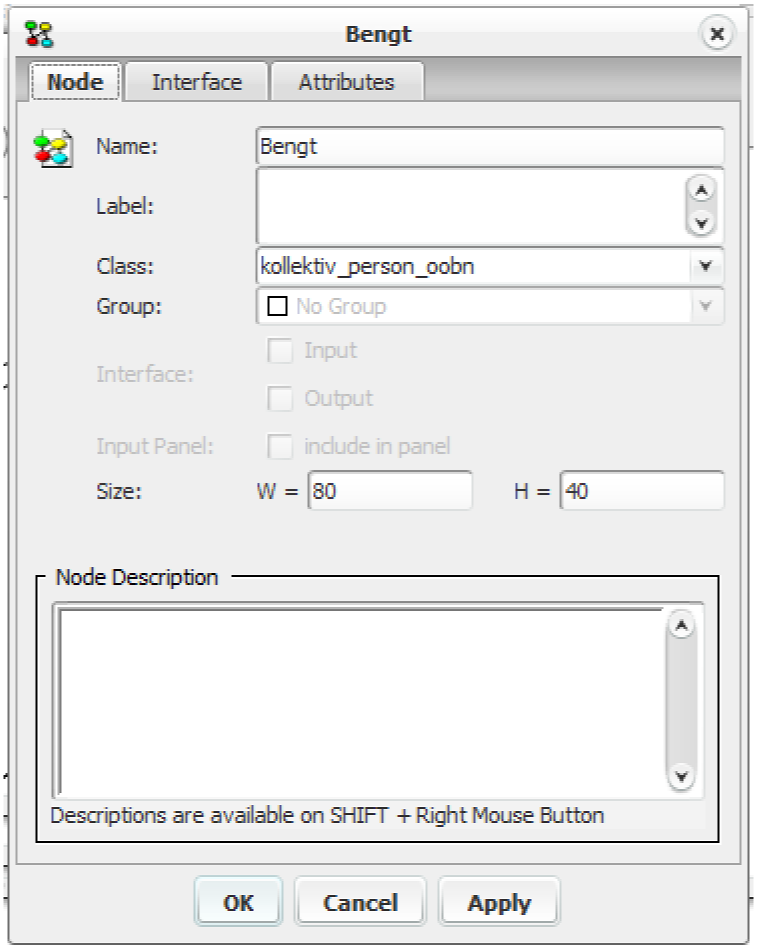
Node Description

☑ Edit

Descriptions are available on SHIFT + Right Mouse Button

Figure 2: The node properties dialog box showing the Node tab when HTML is selected in *Network Properties Pane* (page 178).

A Node Properties dialog showing the Node Tab of an Instance Node is shown in Figure 2.



Bengt

Node | Interface | Attributes

Name: Bengt

Label:

Class: kollektiv_person_oobn

Group: □ No Group

Interface: □ Input □ Output

Input Panel: □ include in panel

Size: W = 80 H = 40

Node Description

Descriptions are available on SHIFT + Right Mouse Button

Figure 3: The node properties dialog box showing the Node tab of an *Instance Node* (page 224).

Note that the dialog in Figure 3 has a field named *class*, where it is possible to configure which network class the node is an instance of.

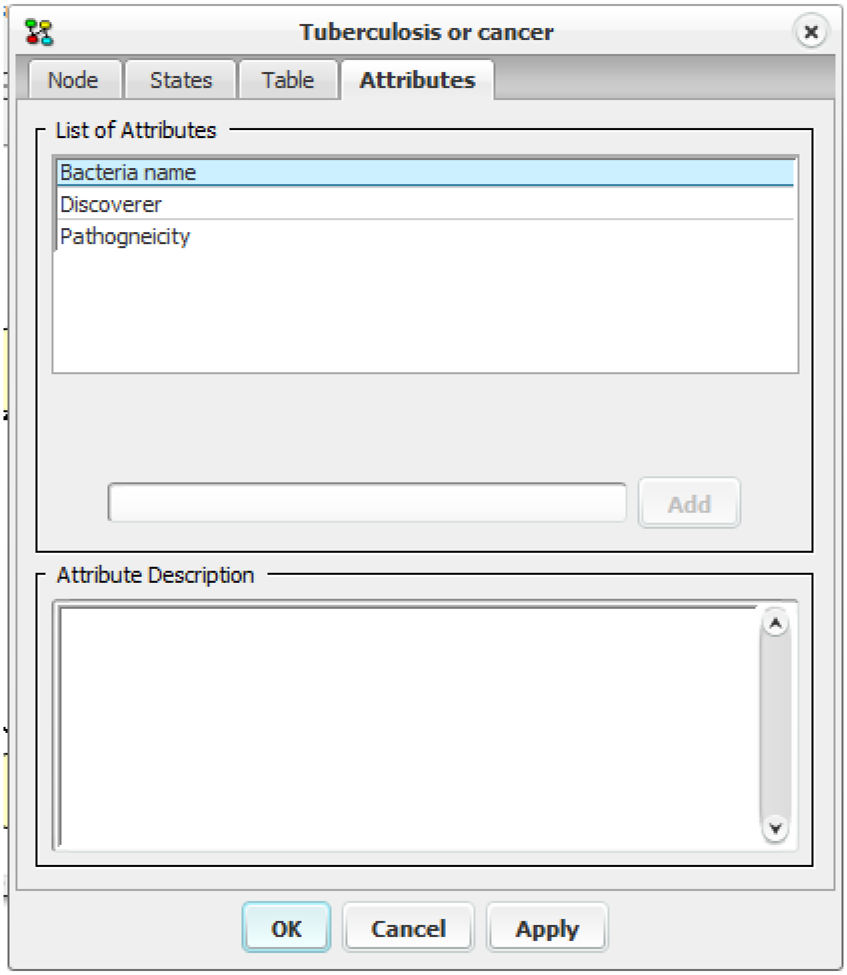
4.3.9 Node Attributes

The *Attribute tab* of the *Node Properties* (page 212) is used to specify the attributes of a node. Each Attribute consists of an *Attribute Name* and an *Attribute Description*. The attribute name (or *attribute identifier*) is a unique name to identify an attribute. Node attribute name can only consists of *C* like identifier characters and must not start with *HR_* since the HUGIN Graphical User Interface uses attributes starting with *HR_*. *Attribute description* consists of a string which can be used to give a more detailed description of the node. It is possible to modify, delete, add attribute both in *Edit Mode* (page 124) *Run Mode* (page 123). To add a new attribute enter a name in the name field along with a description in the Attribute Description field and click on the 'Add' button.

To rename an attribute, select its cell and start typing. A cell can be selected by tabbing the tabulator key untill it is highlighted or by selecting it with the mouse. As soon as an attribute is selected its attribute description is displayed in the Attribute Description field. The attribute description can be edited by editing the text in the Attribute Description field.

To delete an attribute, select it and press the delete or back-space key.

Figure 1 shows the Attribute tab of a node.



Bacteria name
Discoverer
Pathogneicity

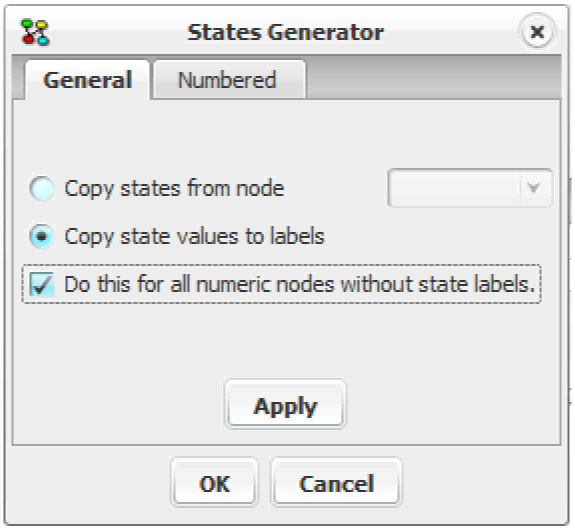
Figure 1: The node properties dialog box showing the Attributes Tab.

Attributes are useful in situations where you want to store additional information (knowledge) in the network apart from cause effect relations. This feature can be very useful when using the HUGIN API and can save a lot of programming.

4.3.10 Node State Generator

The *Node States Generator* function is available in *Table View* (page 156). Using the *Node States Generator* it possible to copy states from other nodes, to transfer state values to state labels, and to automatically generate states for numbered and interval nodes.

In case the active frame has numerous numeric nodes that need to achieve state labels it is possible to request the operation to be performed on each numeric node.



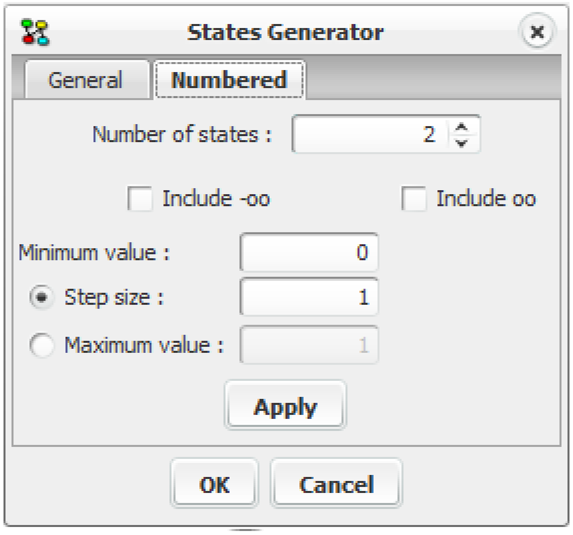
○ Copy states from node

● Copy state values to labels

☑ Do this for all numeric nodes without state labels.

Figure 1: The functionality for copying states and state labels.

Figure 1 shows the functionality that applies to the selected node in general.



Number of states : 2

☐ Include -oo ☐ Include oo

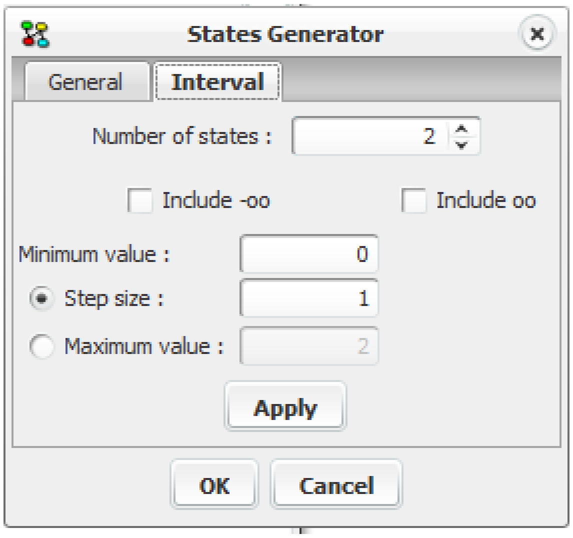
Minimum value : 0

● Step size : 1

○ Maximum value : 1

Figure 2: The functionality for generating states of numberede nodes.

Figure 2 shows the functionality for generating states of numbered nodes while Figure 3 show the functionality for generating states of interval nodes.



Number of states : 2

□ Include -oo □ Include oo

Minimum value : 0

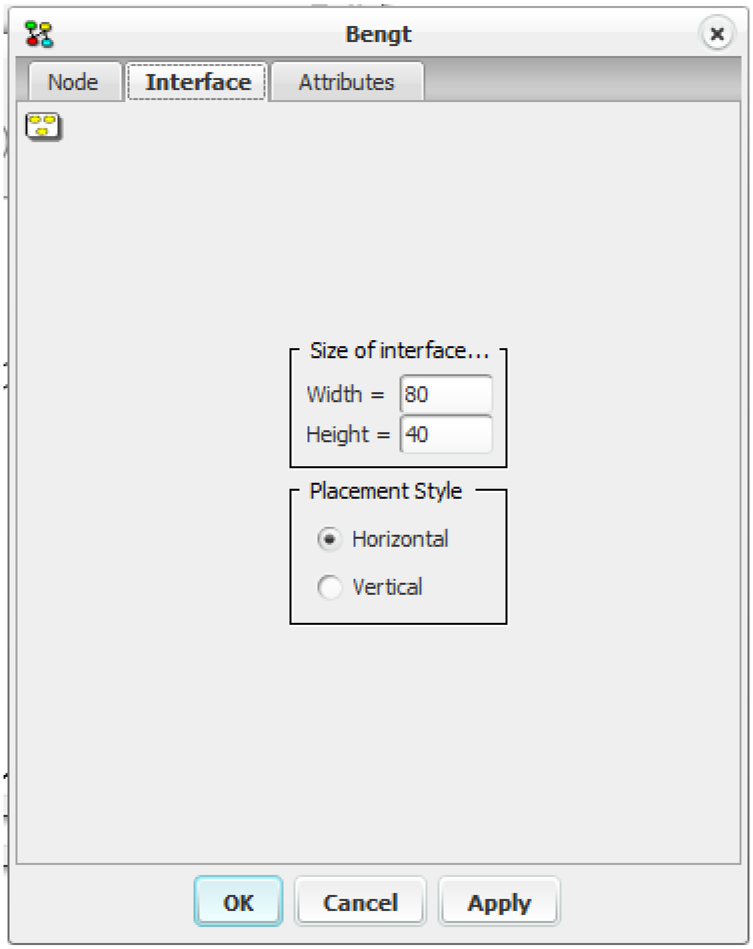
● Step size : 1

○ Maximum value : 2

Figure 3: The functionality for generating states of interval nodes.

4.3.11 Interface Tab

The *Interface Tab* of the *Node Properties* (page 212) dialog allows the user to edit different attributes of an *Instance Node* (page 224). The Interface Tab is shown in Figure 1.



Size of interface...
Width = 80
Height = 40

Placement Style
● Horizontal
○ Vertical

Figure 1: The node properties dialog box showing the Interface Tab.

On the Interface tab you can specify the *size* (page 145) of the selected node, and the placement style of the *input nodes* (page 227) and *output nodes* (page 228) in the *Instance Node* (page 224).

4.3.12 Discrete Chance Node

A *discrete chance node* represents a discrete random variable with a finite number of states. A discrete chance node can be added to the current network by activating the *discrete chance tool* (page 153) and clicking somewhere in the *Network Pane* (page 177).

When *propagating* (page 245), you can see the probabilities of each state in a discrete chance node either in the *Node List Pane* (page 243) or by opening a *monitor window* (page 194) for the node.

Discrete chance nodes are assigned a *type* (page 216).

4.3.13 Continuous Chance Node

A continuous chance node (or CG node = continuous Gaussian node) represents a random variable with a *Gaussian (normal) conditional distribution function* (page 117). A continuous chance node can be added to the current network by activating the *continuous chance tool* (page 153) and clicking somewhere in the *network pane* (page 177).

Currently, there are some restrictions to the use of continuous chance nodes, partly because the underlying theory is still being developed. The restrictions are listed below:

- The only continuous chance nodes currently supported represent variables with Gaussian (normal) distribution functions.
- A continuous chance node cannot be parent of a *discrete chance node* (page 223).
- Continuous nodes cannot be used in influence diagrams. That is, continuous nodes cannot exist in a network also containing *utility nodes* (page 226) or *decision nodes* (page 224).

4.3.14 Decision Node

A decision node represents a decision to be made by the user. A decision node can be added to the current network by activating the *Decision Tool* (page 154) and clicking somewhere in the *Network Pane* (page 177). A *Bayesian network* (page 345) extended with decision nodes and *utility nodes* (page 226) is called a *Limited Memory Influence Diagram (LIMID)* (page 345). The decision nodes represent decisions controlled by an external user. This means that compared to *discrete chance nodes* (page 223), decision nodes have no conditional probability tables (CPTs). Instead an initial policy for the decision is specified.

In the LIMID, *monitor windows* (page 194) for decision nodes and *Node List Pane* (page 243) show the expected utility of each decision option and the probability of choosing each option under the current strategy.

Decision nodes cannot appear in a network together with *continuous chance nodes* (page 224).

Decision nodes are assigned a *type* (page 216).

4.3.15 Instance Node

An instance node of a network represents an instance of another network. In other words, an instance node represents a sub-network. A network containing one or more instance nodes is to be considered an *object-oriented network* (page 346). An instance node is created using the *Instance Tool* (page 152). In order for an instance node to be connected to other (basic) nodes, the network class of which it is an instance must contain *interface nodes* (page 229) (i.e., *input nodes* (page 227) and/or *output nodes* (page 228)).

An instance node is displayed as a rounded rectangle containing the interface nodes of the instance that it represents. See Figure 1 for an example.

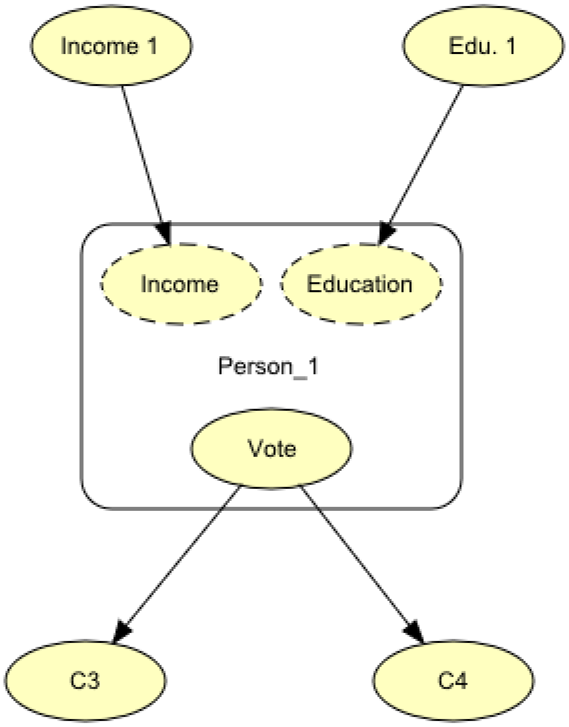
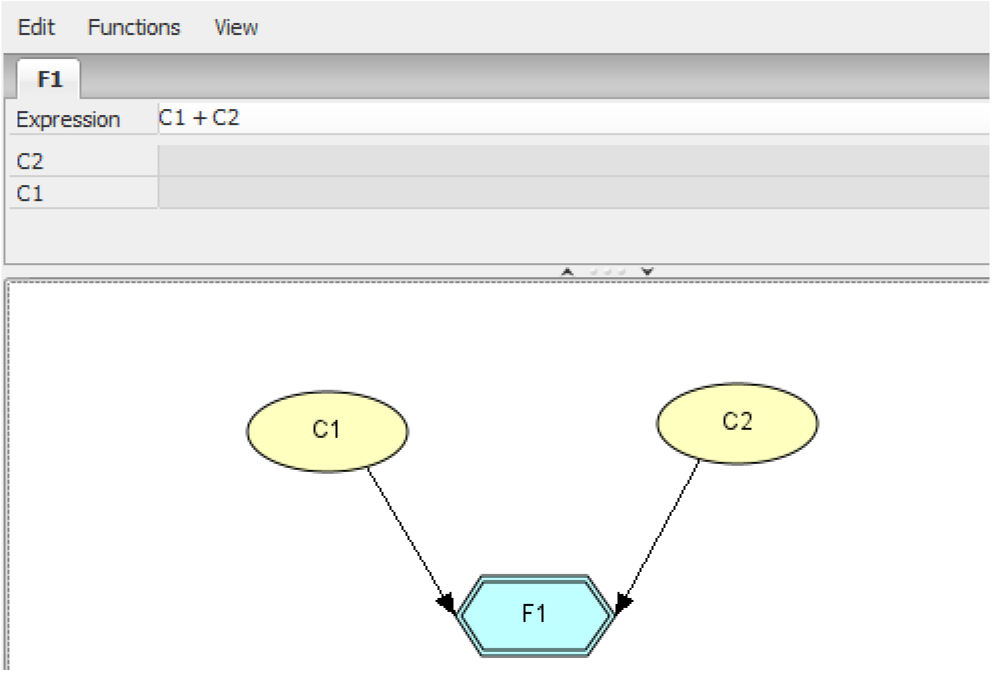


Figure 1: This object-oriented network contains an instance node, named Person_1, representing an instance of the Person network.

4.3.16 Function Node

A function node represents a real-valued function that depends on (some or all of) the parents of the node. Function nodes are not (directly) involved in the inference process - evidence cannot be specified for function nodes, but the function associated with the node can be evaluated using the results of inference or simulation as input. Add a Function node by using the *Function Tool* (page 152).



Expression	C1 + C2
C2	
C1	

Figure 1: A Function node (F1) with a discrete and a continuous parent.

Function node values are specified as an *expression* (page 150) in the node *model* (page 242). A number of rules apply for Function Nodes:

- If Function node has non-function node children, there must not exist a cyclic functional dependency such that one of the function node ancestors depends on the value of the function node.
- Function value expressions can use parent Function-, Boolean-, Numbered- and Continuous node values. (*currently Interval nodes are not supported*).
- When a Continuous node appear in the expression, the mean of the node is used when computing the function value.

4.3.17 Utility Node

A utility node represents an additive contribution to the utility function in a (*limited-memory) influence diagram* (page 345). Thus, the utility function is the sum of all the utility nodes in the influence diagram.

Each utility function has assigned a *utility table* (page 233) which specifies the utility of each configuration of the parents of the utility node (in an influence diagram, a *link* (page 147) from a *decision node* (page 224) or a *discrete chance node* (page 223) represents the decision node/chance node to contribute directly to the utility function).

A utility node can be added to the current network (which will then be an influence diagram) by activating the *Utility Tool* (page 154) and clicking somewhere in the *Network Pane* (page 177).

4.3.18 Input Node

In order for an instance, I, of a network, C, to be applied in another network, N, (via an *instance node* (page 224) in N representing I), a mechanism is required for connecting I (or rather nodes of I) to other nodes of N. As we wish to enforce modularity and information hiding, we cannot allow nodes of I to have parents outside I, as that would violate the modularity constraint (the domain of a CPT could then contain nodes from several networks). Therefore, we need the concept of *input nodes*. An input node of I (declared as such in the network class C) is to be considered a placeholder node for a node in N to be bound to that input node.

The network in Figure 1 contains an instance of a network class, Person, describing the relationships between some indicator parameters (say income, education, etc.) and how these affect the probability distribution over political sympathy, say Republican or Democrat, for a person. The indicator variable Income, for example, might be a labelled discrete chance variable with three states, “High”, “Medium”, and “Low”, and have associated with it the probability distribution P(Income) = (0.1, 0.7, 0.2), reflecting the population average. Now, for an instance of this network to be applied in our network, we need to be able to bind the indicator variable Income to a variable representing a different probability distribution, P(Income 1) = (0.3, 0.65, 0.05), which matches our specific sub-population of interest. Therefore, the indicator variables of the Person network are declared as input variables, which should be bound to the actual variables of the networks in which instances of Person appears. Thus, we have created a *binding link* (page 149) from the node labeled “Income 1” to the input variable “Income” of the instance node. Similarly, the node “Edu. 1” is bound to the input node “Education”.

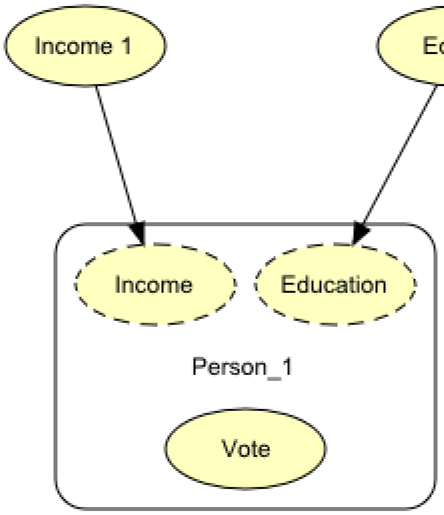


Figure 1: The variables “Income 1” and “Edu. 1” are bound to input variables Income and Education of the instance node Person_1.

Input nodes shouldn’t be confused with real nodes. A real node, which is type consistent with an input node, can be bound to that input node. That is, an input node becomes identical with the node that is bound to it. However, if an input node hasn’t got a binding associated with it, the network containing the input node can still be used (i.e., compiled in to a junction tree and used for inference). In that case the input node is treated as a real node. That is, each input node has a CPT associated with it just as any ordinary node, but this CPT is used only if no nodes have been bound to the input node in a network containing an instance of the network in which the input node is defined.

In the tutorial on *object-oriented networks* (page 56), an example shows how to use input nodes.

See also the description of *output nodes* (page 228). Input nodes and output nodes are collectively referred to as interface nodes.

4.3.19 Output Node

In order for an instance, I, of a network, C, to be applied in another network, N, (via an *instance node* (page 224) in N representing I), a mechanism is required for connecting I (or rather nodes of I) to other nodes of N. As we wish to enforce modularity and information hiding, we cannot allow all nodes of I to have children outside I, as that would violate the information hiding constraint. Therefore, we need the concept of *output nodes*. An output node of I (declared as such in the network class C) is visible outside I, and can thus be used as a parent node of nodes outside I. Note that an output node cannot be a child of nodes outside I, as this would violate the modularity constraint (the domain of a CPT could then contain nodes from several networks).

The network in Figure 1 contains an instance of a network class, Person, describing the relationships between some indicator parameters (say income, education, etc.) and how these affect the probability distribution over political sympathy, say Republican or Democrat, for a person. Given the bindings of the indicator variables Income and Education (see the description of *input nodes* (page 227) for details) to "Income 1" and "Edu. 1", we are interested in the resulting probability distribution for Person_1's political sympathy. Therefore, the variable Vote must be visible outside instances of the network class Person. Thus, the variable Vote is made an output node of Person.

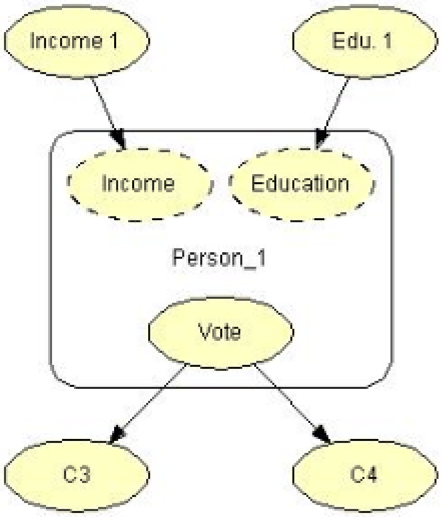


Figure 1: The variables "Income 1" and "Edu. 1" are bound to input variables Income and Education of the instance node Person_1.

In our network, we have used the output variable Vote as a parent variable of the nodes C1 and C2.

In the *tutorial on object-oriented networks* (page 56), an example shows how to use input and output nodes.

See also the description of *input nodes* (page 227). Input nodes and output nodes are collectively referred to as *interface nodes*.

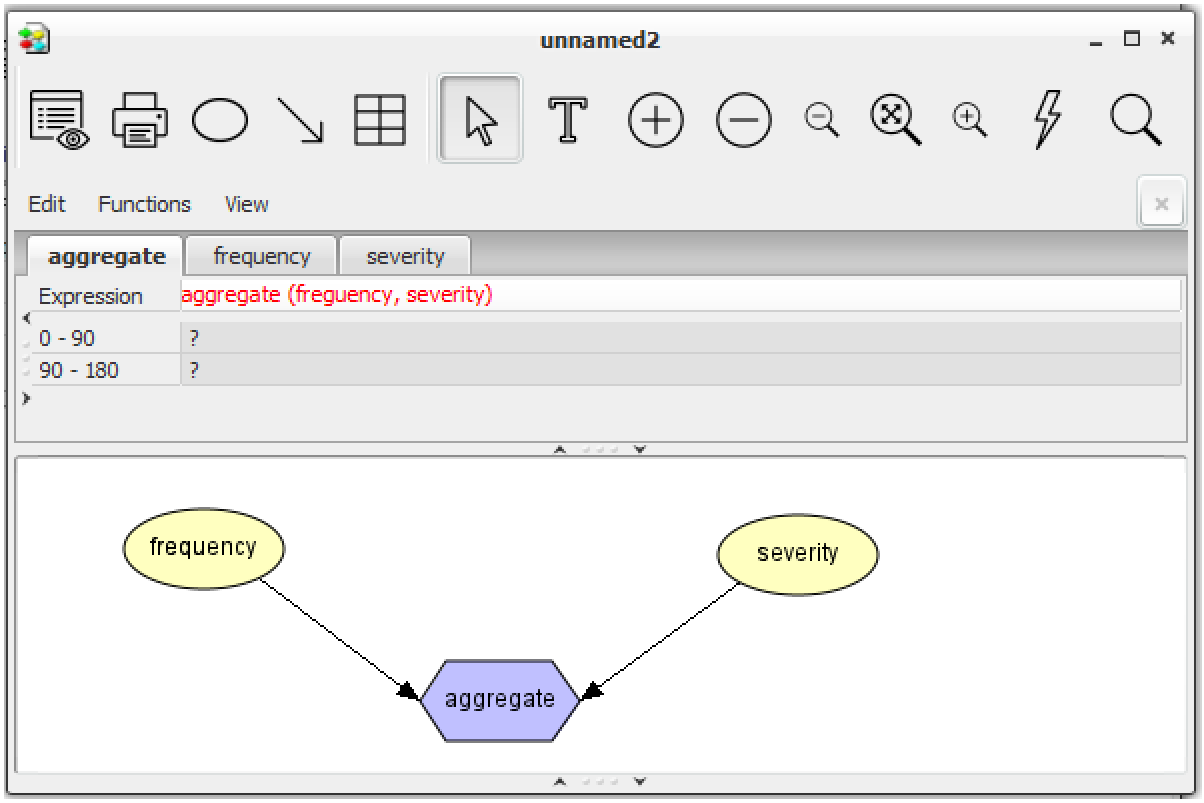
4.3.20 Interface Node

An interface node is either an *input node* (page 227) or an *output node* (page 228).

4.3.21 Discrete Function Node

The table of a discrete function node is not a conditional distribution given the parents (unlike the other discrete nodes). Instead, it is a marginal distribution which can be a function of the values of the parents.

Add a Discrete Function node by using the *Discrete Function Tool* (page 154).



Expression	aggregate (freguency, severity)
0 - 90	?
90 - 180	?

Figure 1: A Discrete Function node (aggregation) with a numbered and interval discrete parent and the aggregate operator as expression.

The Discrete Function node table is specified using an *expression* (page 150) in the node *model* (page 242).

4.3.22 Expand Instance Node

The Expand Instance Node function can be activated by clicking the left mouse button just outside a collapsed *instance node* (page 224). The active region surrounding a collapsed instance node is indicated by the special cursor image shown in Figure 1.

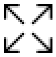


Figure 1: The Expand Instance Node Cursor that appears when the mouse cursor is located right outside a collapsed instance node.

The Expand Instance Node function expands an instance node so that the *interface nodes* (page 229) of the network instance that the instance node represents become visible and the size of the instance node is increased. An instance node must be expanded to create *binding links* (page 149) to the *input nodes* (page 227) of the instance node and to create ordinary (causal) *links* (page 148) from its output nodes. You can also *expand all* (page 230) instance nodes in one operation.

Please note that links to and from an instance node can only be selected when the node is expanded.

See also *Collapse Instance Node* (page 230).

4.3.23 Collapse Instance Node

The *Collapse Instance Node* function can be activated by clicking the left mouse button just outside an expanded *instance node* (page 224). The active region surrounding an expanded instance node is indicated by the special cursor image shown in Figure 1.

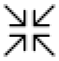


Figure 1: The Collapse Instance Node Cursor that appears when the mouse cursor is located right outside an expanded instance node.

The Collapse Instance Node function collapses an instance node so that the *interface nodes* (page 229) of the network instance that the instance node represents become hidden and the size of the instance node is reduced. This is useful for making the display of a network less cluttered. You can also *collapse all instance nodes* (page 230) in one operation.

Please note that links to and from an instance node can only be selected when the node is expanded.

See also *Expand Instance Node* (page 229).

4.3.24 Expand Instance Nodes

The Expand Instance Nodes function can be activated from the *View Menu* (page 188). This function expands all *instance nodes* (page 224) so that the *interface nodes* (page 229) of the network instance that the node represents become visible. An instance node must be expanded to create *binding links* (page 149) to the input nodes of the instance node and to create ordinary (causal) *links* (page 148) from its output nodes. Instance nodes can also be *expanded individually* (page 229).

Please note that links to and from an instance node can only be selected when the node is expanded.

See also *Collapse Instance Nodes* (page 230).

4.3.25 Collapse Instance Nodes

The *Collapse Instance Nodes* function can be activated from the *View Menu* (page 188). This function collapses all *instance nodes* (page 224) so that the display of the network becomes less cluttered. Instance nodes can also be *collapsed individually* (page 230).

Please note that links to and from an instance node can only be selected when the node is expanded.

See also *Expand Instance Node* (page 229).

4.3.26 Expand Node List

The expand node list function can be activated from the Expand Node List item in the *View Menu* (page 188) or from the expand node list button in the *Run Mode* (page 123) Tool Bar (the button shown in Figure 1).

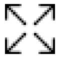


Figure 1: The expand node list tool button from the Run Mode Tool Bar.

The expand node list function enables the display of states/decisions and their belief/expected utility of all nodes in the *Node List Pane* (page 243).

You can expand a single node in the node list by clicking it with the mouse.

This function has an inverse in the *Collapse Node List* (page 231) function.

4.3.27 Collapse Node List

The collapse node list function can be activated from the Collapse Node List item in the *View Menu* (page 188) or from the collapse node list button in the *Run Mode* (page 123) Tool Bar (the button shown in Figure 1).

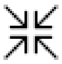


Figure 1: The collapse node list tool button from the Run Mode Tool Bar.

The collapse node list function disables the display of states/decisions and their belief/expected utility in the *Node List Pane* (page 243).

You can collapse a single node in the node list by clicking its close icon.

This function has an inverse in the *expand node list* (page 231) function.

4.3.28 Experience Tables

Each discrete or continuous chance node in a Bayesian network or a LIMID may be assigned a experience table. Experience tables play an important role in (batch) parameter estimation using the *EM Algorithm* (page 352) and sequential parameter update using *Adaptation* (page 244). Consider the Bayesian network shown in Figure 1.

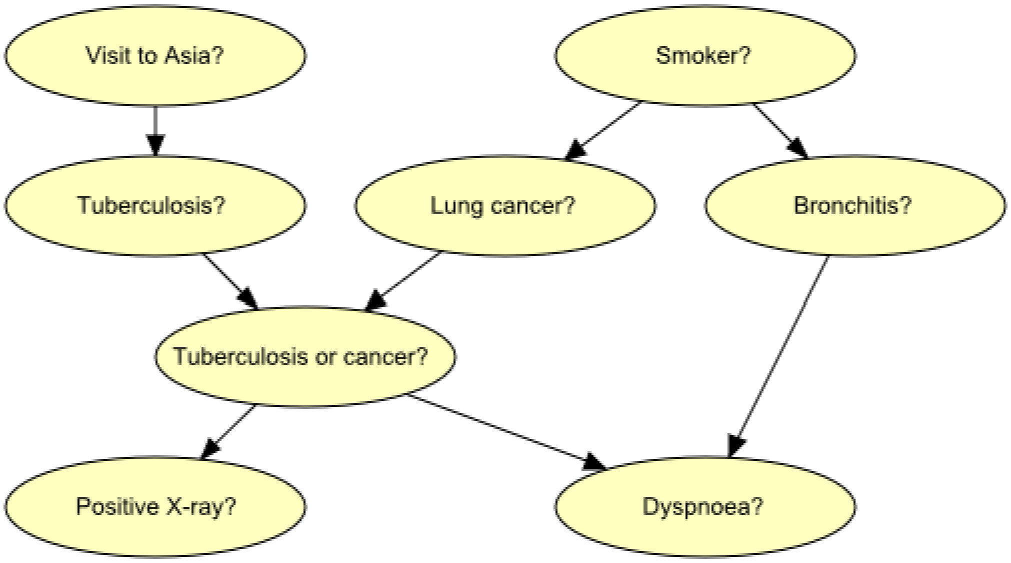
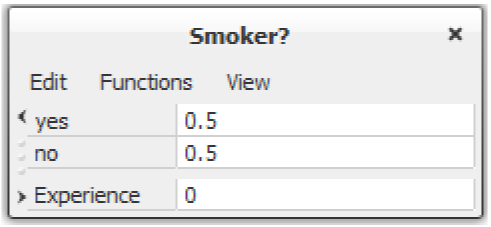


Figure 1: Bayesian-network representation of "Chest Clinic".

Experience table(s) are required in the parameter estimation process as well as in the adaptation process (adaptation is not supported for continuous chance nodes though). To explain what happens when we start using experience tables, let's consider the "Smoker?" variable. To add an experience table to the variable, select the variable, do a right click and choose "Add Experience Table" in the "Experience/Fading Table Operations" submenu. To see the created experience table, first activate the Tables Pane by clicking the tables-pane button (as you probably did in the *How to Build BNs tutorial* (page 26)), select the "Smoker?" node (if not already selected), select the "Show experience table" item of the View menu for the *node table* (page 63) of "Smoker?". The node table for the "Smoker?" variable is shown in Figure 2.



yes	0.5
no	0.5
Experience	0

Figure 2: Node table for the "Smoker?" discrete chance variable.

The count in the experience table represents the number of observations made so far on the "Smoker?" variable. The value of the experience count must be greater than 0 to activate adaptation whereas the experience table must exist to activate parameter estimation. It is possible to turn off parameter estimation for a specific parent configuration by specifying a negative experience count for that parent configuration. If we believe the correctness of the present conditional probability distribution or density is high, then the experience count must have a high value; otherwise, the value of the count should be low. In the present example we assume that our belief in the correctness of the current conditional distribution is low, thus we set the initial experience count to a small number, say 10. To modify the experience count, just click on count cell of the experience table in the Tables Pane and enter the value 10. This means that HUGIN assumes we have observed the value of the "Smoker?" variable 10 times so far.

You can change a specific numeric value by selecting it with the mouse cursor and typing in a new value from the keyboard.

Experience tables can be added to many variables at the same time. To add experience tables to all the chance variables click somewhere in the Network Pane, click the right mouse button and then choose the “Add Experience Table to All Chance Nodes” item of the “Experience/Fading Table Operations” submenu. To add experience tables to a subset of chance variables, select these nodes, right click, and select “Add Experience Table” in the “Experience/Fading Table Operations” submenu.

4.3.29 Fading Tables

Each discrete chance node in a Bayesian network or a LIMID may be assigned a fading table. Fading tables only play a role when performing sequential parameter update using *Adaptation* (page 244). The use of fading table is described in detail under *Adaptation* (page 244) and *Adaptation Tutorial* (page 111).

4.3.30 Open Tables

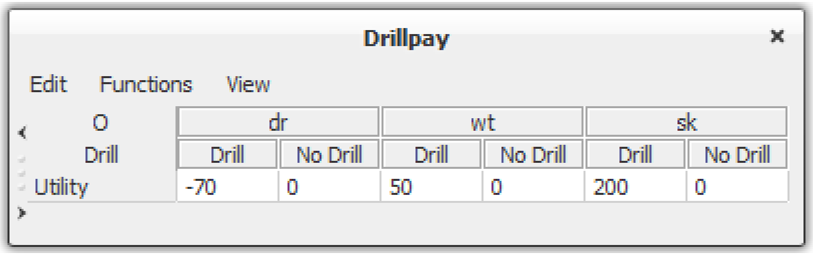
The Open Tables function can be activated by selecting the “Open Tables” item of the *Network Menu* (page 185). The tables of the selected nodes will be shown in the *table panel* (page 239). If the table panel is not open, it will be opened automatically.

4.3.31 Utility Table

To each *utility node* (page 226) is associated a special instance of a *node table* (page 241), called a utility table.

The table of a utility node is found in the *Tables Pane* (page 239) when the node is selected.

The utility table specifies a *utility function*. For each configuration of the states of the parents, which can be discrete chance and decision nodes, the utility table has an entry specifying a utility value of this configuration. Figure 1 shows the table for the utility node Utility with the discrete chance node Oil and the discrete decision node Drill as parents.



O	dr		wt		sk	
Drill	Drill	No Drill	Drill	No Drill	Drill	No Drill
Utility	-70	0	50	0	200	0

Figure 1: The table for utility node Utility with parents Drill and Oil.

Each numeric value in the utility table is the utility of the parent nodes being in the states found in the top of the actual column.

You can change a specific numeric value by selecting it with the mouse cursor and typing in a new value from the keyboard.

4.3.32 Decision Table

With each *decision node* (page 224) is associated a decision table, which is very simple, as it just contains a single column of the various decision options associated with the decision node.

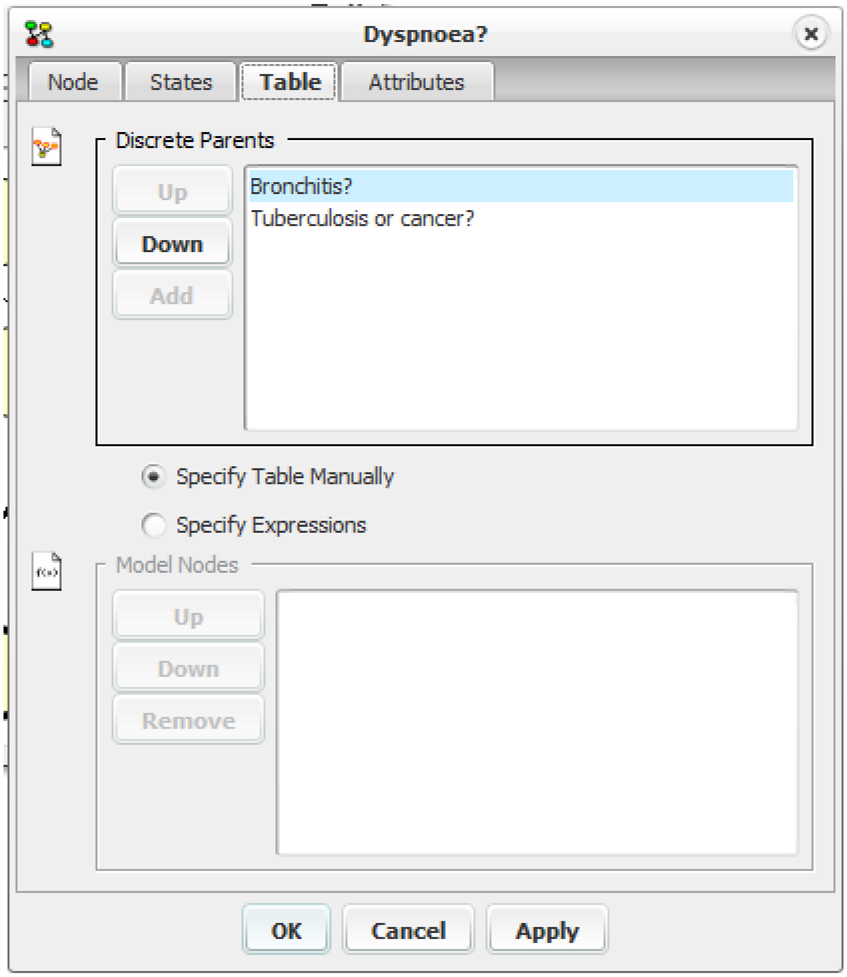
As for all other *node tables* (page 241), a decision table appears in the *Tables Pane* (page 239) whenever opened and the associated decision node is selected node in the *Network Pane* (page 177).

The decision table specifies the policy for the decision node.

4.3.33 Table Tab

The Table tab of the *Node Properties* (page 212) is used to specify different aspects about how you want to edit node tables.

Figure 1 shows the Table tab. A list of the *discrete parents* (page 223) of the node appears at the top section. The order of these parents can be altered using the Up and Down buttons. This reflects the order in which the parents appear in the table of the node (which is displayed in the *Tables Pane* (page 239)) and hence determine which parents have indices that run faster in the table.



Bronchitis?
Tuberculosis or cancer?

● Specify Table Manually

○ Specify Expressions

Figure 1: The Table tab of the Node Properties dialog box.

For *discrete chance nodes* (page 223) and *utility nodes* (page 226) one can either specify, respectively, their probability and utility tables manually or via *expressions* (page 150). This choice can be indicated via the select buttons below the Discrete Parents list (it can also be indicated via the "Switch to expressions" item of the Expressions sub-menu of the Functions menu of the *Node Table* (page 241)).

If the table is specified via an expression, it might be convenient to introduce one or more of the parent nodes as Model Nodes. A separate expression can be specified for each configuration of the model nodes. Alternatively, one may choose to specify one big expression, typically including a number of if-then-else expressions. For more information on this topic, please consult the *Table Generator Tutorial* (page 73).

4.3.34 States Tab

The States tab of the *Node Properties* (page 212) is used to specify the states of the current node.

Figure 1 shows the States tab. Here, you can specify the state names, state values, or state intervals depending on the *type* (page 216) of the node you are currently editing.

- If the type is "Labelled", you can type in anything as the state label.
- If the type is "Boolean", you cannot change the two states "false" and "true".
- If the type is "Numbered", you can type in a number. Please note that the numbers must form an increasing sequence.
- If the type is "Interval", you can type in intervals in the format: "<number> - <number>".

To add a new state label/value/interval enter a state name in the state name field and press Add. Optionally, a state description can be entered.

To rename a state label/value/interval, select its cell and start typing. A cell can be selected by tabbing the tabulator key untill it is highlighted or by selecting it with the mouse.

To delete a state, select it and press the delete or back-space key.

If the endpoints of an interval are changed for an interval node, the endpoints of the neighboring states will automatically be updated such that there will be no gaps between state intervals. The symbols "infinity" or "inf" may be used to specify an infinite right endpoint. Similarly, "-infinity" or "-inf" may be used to specify a negative infinite left endpoint.

To rearrange the order of the state labels, first select a state and then move it up or down pressing the "Up" and "Down" buttons.

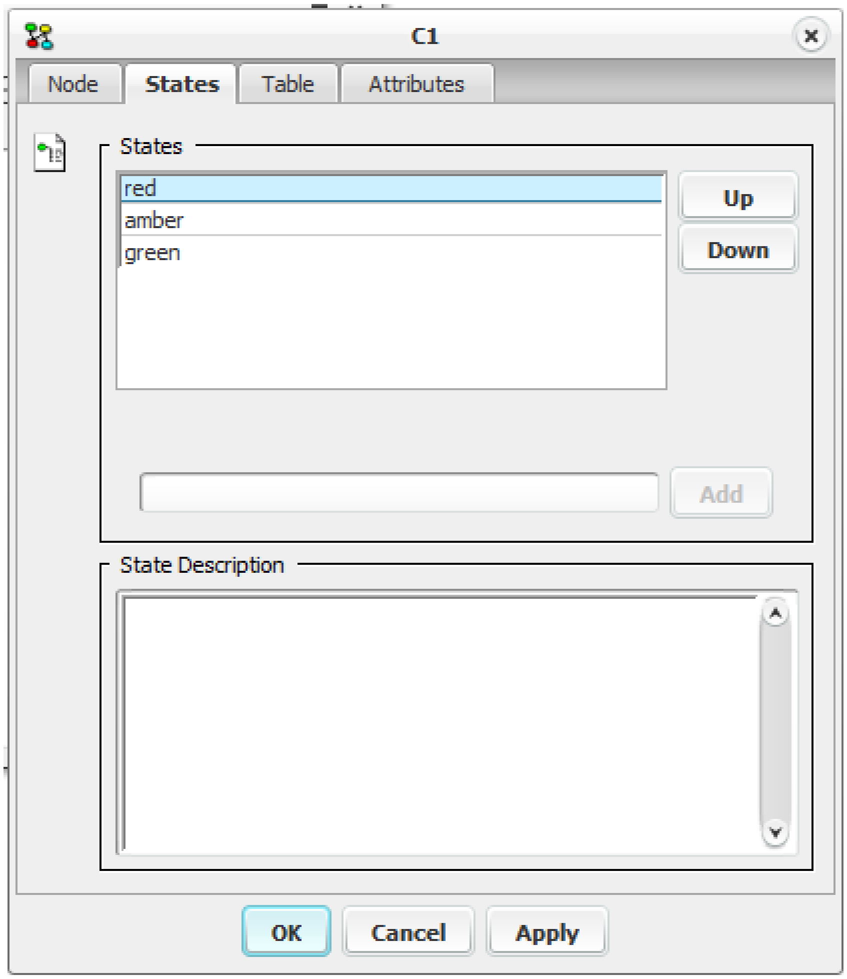


Figure 1: The Node Properties dialog box showing the States tab of a node of type “Numbered”.

4.3.35 Add Experience Table to all Chance Nodes

Before performing *adaptation* (page 244) or *EM-learning* (page 352), the nodes, whose tables are to be learned, must have an experience table added to them. The “Add Experience Table to all Nodes” function is a utility function, which add these to all relevant nodes. It can be accessed by selecting the “Add Experience Table to all Nodes” item from either the *Network Menu* (page 185), or from the popup menu (brought up by right-clicking on the network).

4.3.36 Add Fading Table to all Discrete Nodes

Before performing *adaptation* (page 244) on a network, one might consider adding fading tables to the nodes, since this allows “old” values to be faded out, so that the newest learned values carry a higher weight. The “Add Fading Table to all Discrete Nodes” function is a utility function, which add these to all relevant nodes. It can be accessed by selecting the “Add Fading Table to all Discrete Nodes” item from either the *Network Menu* (page 185), or from the popup menu (brought up by right-clicking on the network).

4.3.37 Add State

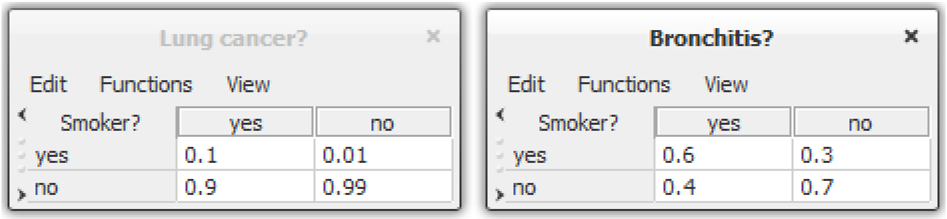
The add state function can be activated from the Add State item in the *Edit Menu* (page 192) or from the add state button in the *Edit Mode* (page 124) Tool Bar (the button shown in Figure 1).

Figure 1: The add state tool button from the Edit Mode Tool Bar.

The add state function adds a new state (or decision option) to the node currently selected. The function only works for *discrete chance nodes* (page 223) and *decision nodes* (page 224). If the node is a decision node, this state is named "State *n*" (where *n* is the total number of states of the node after the new state has been added). If it is a decision node, the new decision option is named "Action *n*".

4.3.38 Copy Table

Copy Table applies to *discrete chance nodes* (page 223), *decision nodes* (page 224), *continuous chance nodes* (page 224), and *utility nodes* (page 226). The only condition that must be met before a table can be copied from one node to another is that the two nodes must have the same number of states and the same number of parent configurations (see Figure 1).



Lung cancer?

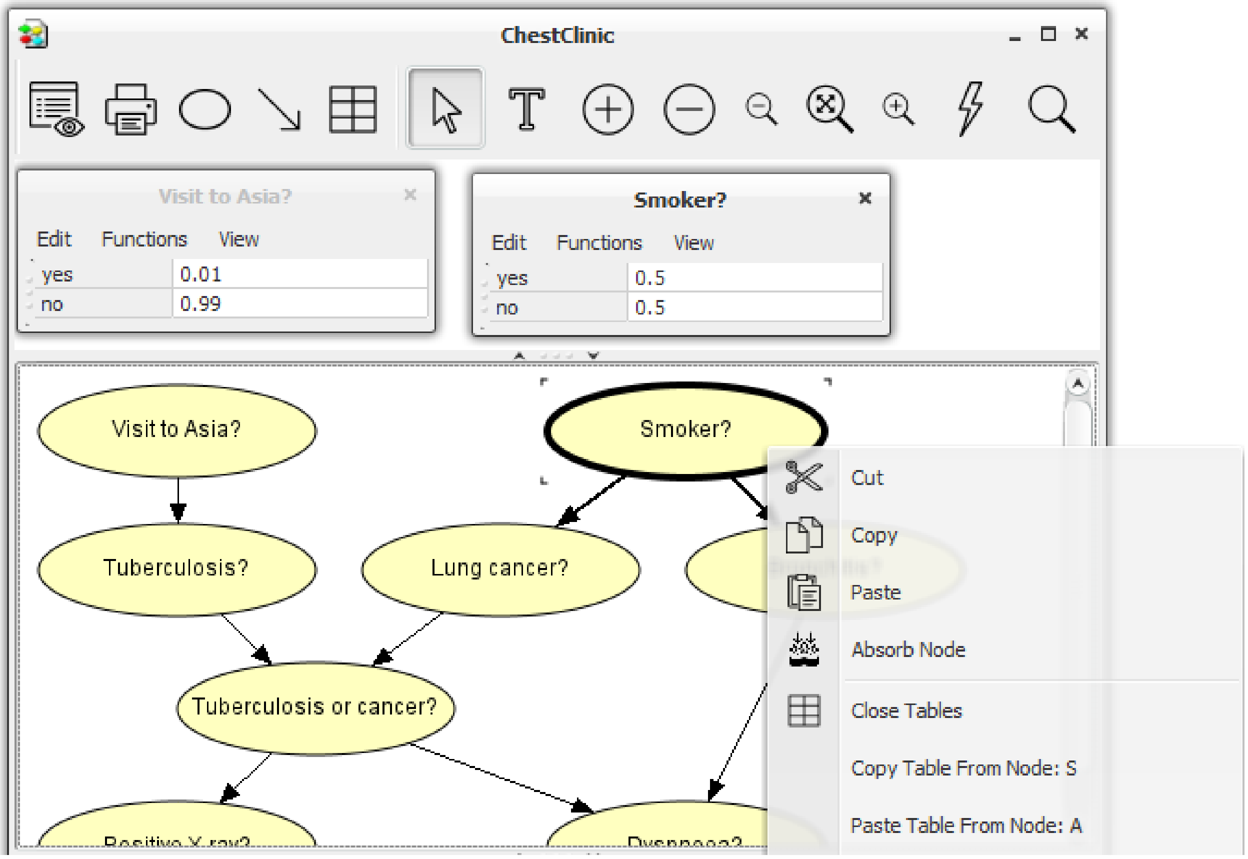
Smoker?	yes	no
yes	0.1	0.01
no	0.9	0.99

Bronchitis?

Smoker?	yes	no
yes	0.6	0.3
no	0.4	0.7

Figure 1: Sample CPTs of identical dimensions.

The Copy Table function is activated in *Edit Mode* (page 124) by selecting the node from which you wish to copy the table and clicking the right mouse button (see Figure 2). As shown in Figure 2 the menu item shows which node the table is copied from. To paste the copied table to a given node, you have to select that node, click the right mouse button, and choose the Paste Table item.



Visit to Asia?

yes	0.01
no	0.99

Smoker?

yes	0.5
no	0.5

Figure 2: Copy / Paste Table items.

4.3.39 Manually Specify Table

Specifying the table of a discrete chance node or utility node manually as an alternative to specifying *Expressions* (page 150) allows the user to edit each number in the node table directly. The table is shown in the *Tables Pane* (page 239).

You can select to specify a table manually by selecting the Switch to manual item of the Expressions submenu of the Functions menu of the *Node Table* (page 241).

4.3.40 Arrange Tables

The tables appearing in the *Tables Pane* (page 239) are displayed beginning in the upper left corner of the pane. If, however, the node that has its table displayed in the upper left corner of the pane gets deselected (making its table disappear and leave an empty area in the pane) or the tables have been moved around and are overlapping, it can be desirable to rearrange the tables so that empty space gets filled and the tables are not overlapping. To do so, the *Arrange Tables Button* in the *Network Window* (page 182) Tool Bar can be activated. Figure 1 shows a sample arrangement of the *node tables* (page 241) in the Tables Pane, and Figure 2 shows the same tables after rearrangement has taken place.

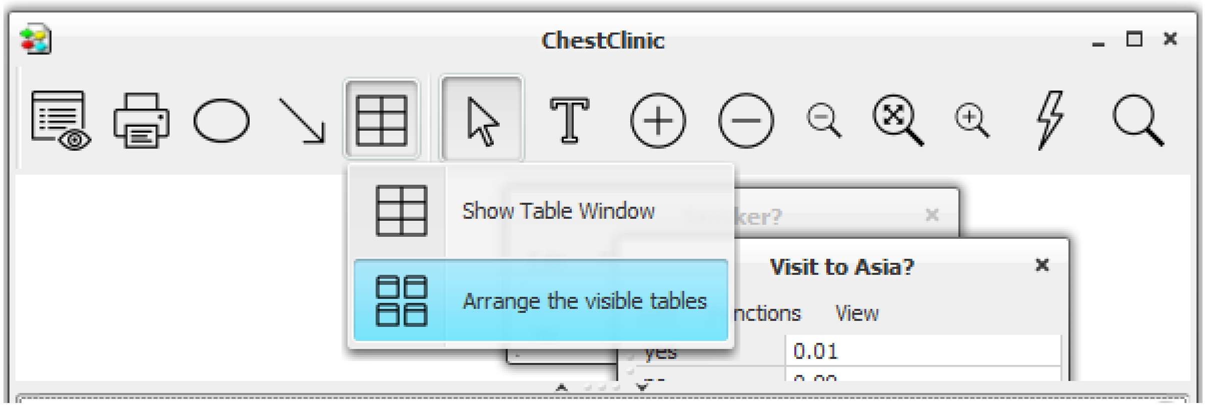


Figure 1: The tables displayed in the Tables Pane can be rearranged by activating the Arrange Tables Button (red arrow) in the Network Window Tool Bar.

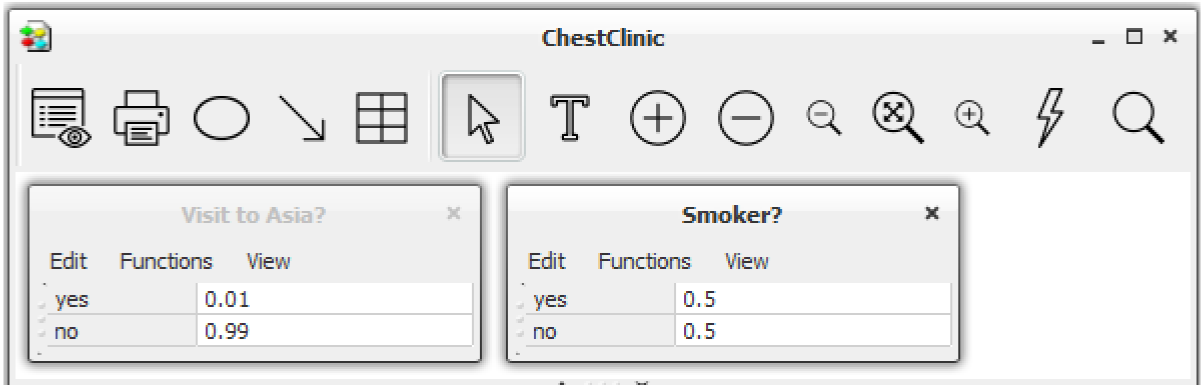


Figure 2: The rearranged tables.

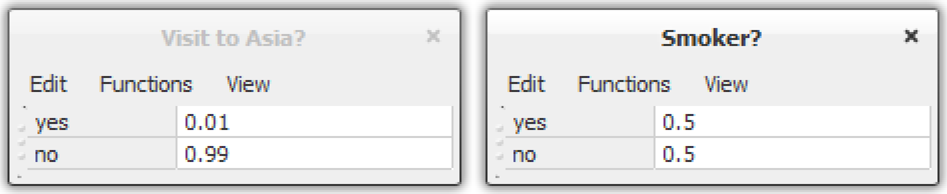
4.3.41 Close Tables

The Close Tables function can be activated by selecting the "Close Tables" item of the *Network Menu* (page 185). The tables of the selected nodes will be removed from the *table panel* (page 239).

4.3.42 Tables Pane

The Tables Pane is used in the *Edit Mode* (page 124). It displays the table (if any) of the kind selected in the *node table* (page 241) for each selected node. For each *discrete chance node* (page 223) and *utility node* (page 226) an actual table is displayed (if it has one of the kind selected). For each *decision node* (page 224) it shows the initial policy and for each *continuous chance node* (page 224) the distribution function is specified through a mean and a variance for each configuration of the discrete parents.

Figure 1 shows the Tables Pane containing CPTs for two discrete chance nodes.



yes	0.01
no	0.99

yes	0.5
no	0.5

Figure 1: The Tables Pane.

The Tables Pane appears when the *Tables-Pane Button* in the Network Window Tool Bar is activated (see Figure 2). This button is a radio button, so to make the Tables Pane disappear, simply reactivate the button.

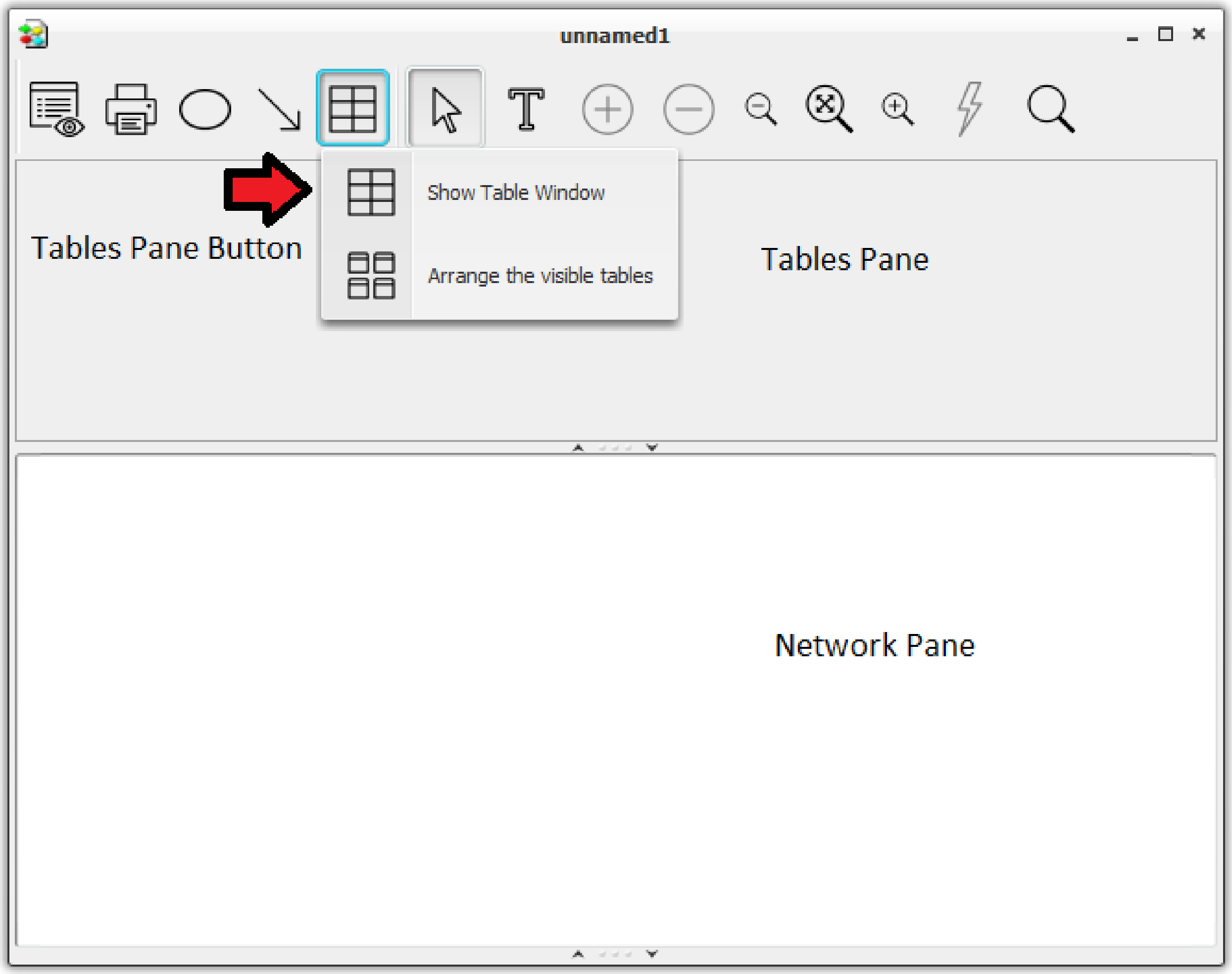


Figure 2: The Tables Pane appears when the *Tables-Pane Button* in the Network Window Tool Bar is activated, and it disappears when the button is reactivated.

See also *Arrange Tables* (page 238).

Under the *Functions* menu item it is possible export and import the content of a node table to text file.

4.3.43 Node Table

To each chance node and each utility node in a Bayesian network or a (limited-memory) influence diagram is assigned a table, specifying a function associated with the node. Also, each decision node has a table associated with it, specifying an initial policy for the decision variable represented by the node. These tables are referred to as *node tables*. The types of the tables differ depending on the types of the nodes.

The HUGIN Graphical User Interface provides a number of powerful features for displaying node tables, including resizing, collapsing selected sets of columns, graphics display modes, etc.

The probabilities and utilities of discrete chance nodes and utility nodes, respectively, can be specified manually (i.e., directly through click-and-type operations) or they can be specified indirectly through specification of powerful *expressions* (page 150), which (if appropriate) can save a lot of work. The initial policy for a decision node can also be specified both manually and using expressions. Notice that the initial policy of a decision node is updated, if *Single Policy Updating* (page 258) is invoked.

When working in *Edit Mode* (page 124) with the *Tables Pane* (page 239) open, the currently selected nodes will have their tables displayed in the Tables Pane.

See the *Node Table Tutorial* (page 63) for more information on node tables.

4.3.44 Conditional Probability Table (CPT)

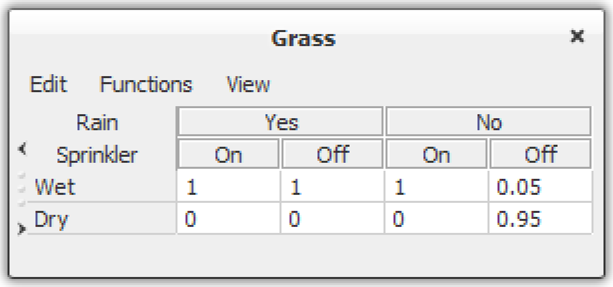
Each discrete chance node in a Bayesian network or a LIMID is assigned a conditional probability table (abbreviated CPT). The *node name* (page 210) and *node label* (page 210) displayed in the *Tables Pane* (page 239) when working in *Edit Mode* (page 124) specifies a currently selected node. If this node is a discrete chance node, the table shown in the *Tables Pane* (page 239) is a CPT.

Only if you have chosen to specify the table *manually* (page 238) (alternative to using *expressions* (page 150)), you can see the entire CPT. Otherwise, you see the table of expressions specifying the table.

The table shown in the Tables Pane is different depending on the type of the currently selected node:

- If the currently selected node is a *discrete chance node* (page 223), the table is a true CPT.
- If the currently selected node is a *utility node* (page 226), the table is a *utility table* (page 233) rather than a CPT.
- If the currently selected node is a *decision node* (page 224), the table represent the policies associated with the decision.
- If the currently selected node is a *continuous chance node* (page 224), the table specifies a *Gaussian (normal) distribution function* (page 117) rather than a CPT.

Figure 1 shows a CPT for the discrete chance node Grass with two parents, Sprinkler and Rain.



Rain	Yes		No	
Sprinkler	On	Off	On	Off
Wet	1	1	1	0.05
Dry	0	0	0	0.95

Figure 1: The CPT for Grass.

The states of the currently selected node are those found in the left most column below the parent labels. You can change these state names by selecting them with the mouse cursor and typing in a new name from the keyboard.

Each numeric value in the CPT is the probability of the currently selected node being in the state found in the left most column in the actual row - when the parents (if any) are in the states found in the top of the actual column.

You can change a specific numeric value by selecting it with the mouse cursor and typing in a new value from the keyboard.

4.3.45 Absorb Node

A *discrete chance node* (page 223) of a Bayesian network or a LIMID can be absorbed. The absorb node function can be activated in the *Edit Menu* (page 192).

A discrete chance node is absorbed by reversing all arcs emerging from A using the *Arc Reversal* (page 149) operation.

As node absorption proceeds by a sequence of arc-reversal operations, new links may be added in the process. New links are added in order to maintain the conditional independence and dependence statements encoded in the network with respect to the unabsorbed variables.

Once all links emerging from A have been reversed, they are removed from the network. This completes the absorption of A. The node A is not deleted from the network.

4.3.46 Toggle Private Nodes

The *Toggle Private Nodes* function can be activated by selecting the “Toggle Private Nodes” item of the *View Menu* (page 188). It toggles the display of private nodes in the instance nodes, when the network is in *Run Mode* (page 123). That is, if the private nodes are hidden, the *Node List* (page 243) will only display the input and output nodes of an expanded *instance node* (page 224).

4.3.47 Move Interface Node

To make the display of a network containing *instance nodes* (page 224) less cluttered it is often desirable to alter the ordering of *the input nodes* (page 227) and/or *output nodes* (page 228) (known as *interface nodes*) of the instance nodes so as to minimize the number of links crossing each other. An input node (output node) of an instance node can be moved one position to the left (right if the Shift key is pressed) simply by clicking an input node (output node) with the left mouse button.

4.3.48 Models

Conditional probability tables (CPTs) for discrete chance nodes, utility tables for utility nodes, and initial policies for decision nodes can be specified compactly through models. A model consists of a list of model nodes and a set of *Expressions* (page 150) (one expression for each configuration of the states of the model nodes). The model nodes constitute a subset of the parents of the node to which the model belongs. This subset is specified under the *Table Tab* (page 234) of the Node Properties dialog box.

Specification of CPTs through model is particularly useful when the conditional probability distribution for a variable is complex and follows (at least approximately) certain functional or distributional forms. In such cases it is cumbersome to specify the conditional probability table (CPT) manually.

When one needs different expressions depending on the states of one or more parent nodes, the use of model nodes can be quite convenient, as complicated expressions can be considerably simplified.

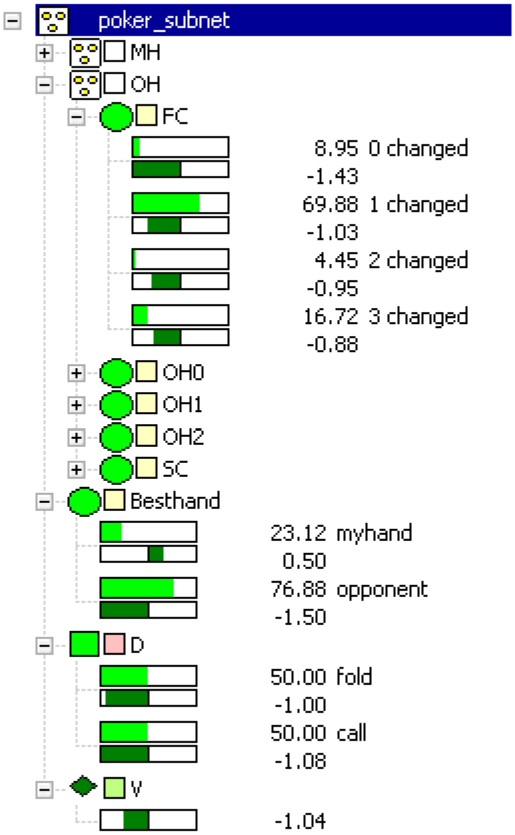
If there are no model nodes, the model contains a single expression.

Examples of the use of models and model nodes are given in the *Table Generator Tutorial* (page 73).

4.3.49 Node List Pane

The *Node List Pane* is used in the *run mode* (page 123) *network window* (page 182). It displays the nodes of the current network in a collapsible list. For instance nodes, it is possible to limit the displayed nodes to the public nodes (that is, to the *input* (page 227) or *output* (page 228) nodes of the instance). The Node List Pane is placed to the left of the *network pane* (page 177), unless the display of it has been turned off via the Toggle Node List item of the *View Menu* (page 188).

Figure 1 shows a Node List Pane of a HUGIN network (*Bayesian network* (page 345), *LIMID* (page 345), or *object-oriented network* (page 346)).



```
poker_subnet
  MH
  OH
    FC
           8.95 0 changed
          -1.43
          69.88 1 changed
          -1.03
           4.45 2 changed
          -0.95
          16.72 3 changed
          -0.88
    OH0
    OH1
    OH2
    SC
  Besthand
          23.12 myhand
           0.50
          76.88 opponent
          -1.50
  D
          50.00 fold
          -1.00
          50.00 call
          -1.08
  V
          -1.04
```

Figure 1: The Node List Pane.

When you *expand a node in the node list* (page 231), you can see the computed results of the last *propagation* (page 245) in a format similar to *monitor windows* (page 194).

4.4 Algorithms

4.4.1 Adaptation

The adaptation algorithm (*sequential updating*) will update the conditional probability distribution of a Bayesian network in the light of inserted and propagated evidence (i.e., experience). The adaptation algorithm only applies to the discrete chance nodes. The algorithm is useful when the graphical structure and an initial specification of the conditional probability distributions are present but the modeled domain changes over time, the model is incomplete, or the model simply does not reflect the modeled domain properly.

In this section we give a quick overview of sequential updating of the conditional probability tables. It is assumed that the reader is familiar with the methodology of Bayesian networks and influence diagrams as well as usage of the HUGIN Graphical User Interface (the graphical user interface). The basic concepts of Bayesian networks are described in *Introduction to Bayesian Networks* (page 19). You can also learn more about influence diagrams in the same section. To get an introduction to the HUGIN Graphical User Interface refer to the tutorial *A Small Bayesian Network* (page 26).

Sequential updating, also known as *adaptation* or *sequential learning*, makes it possible to update and improve the conditional probability distribution for a domain as observations are made. Adaptation is especially useful if the model is incomplete, the modeled domain is drifting over time, or the model quite simple does not reflect the modeled domain properly. Note that the graphical structure and initial specifications of conditional probability distributions must be present prior to adaptation.

The *adaptation algorithm* implemented in HUGIN is developed by *Spiegelhalter & Lauritzen (1990)* (page 541). See also the papers by *Cowel & Dawid (1992)* (page 541) and *Olesen et al. (1992)* (page 541) for a more detailed mathematical description of the algorithm.

Spiegelhalter and Lauritzen introduced the notion of experience. The experience is quantitative memory which can be based both on quantitative expert judgment and past cases. *Dissemination* of *experience* refers to the process of computing prior conditional distributions for the variables in the network. *Retrieval of experience* refers to the process of computing updated distributions for the parameters that determine the conditional distributions for the variables in the network.

In short, the adaptation algorithm will update the conditional probability distributions of a Bayesian network in light of inserted and propagated evidence (i.e., experience). *Note that adaptation can only be applied to discrete chance variables*.

The experience for a given discrete chance node is represented as a set of experience counts Alpha_0,...,Alpha_{n-1}, where *n* is the number of configurations of the parents of the node and $\text{Alpha}_i > 0$ for all i; Alpha_i corresponds to the number of times the parents have been observed to be in the ith configuration. However, note that the "counts" do not have to be integers – they can be arbitrary (positive) real number, thus the counts are only conceptual. The experience counts are stored in a table, known as the *experience table*.

When an experience table is created, it is filled with zeros. Since zero is an invalid experience count, positive values must be stored in the tables before adaptation can take place. The adaptation algorithm will only adapt conditional distributions corresponding to parent configurations having a positive experience count. All other configurations (including all configurations for nodes without experience tables) are ignored. This convention can be used to turn on/off adaptation at the level of individual parent configurations: setting an experience count to a positive number will turn on adaptation for the associated parent configuration; setting the experience count to zero or a negative number will turn it off.

Experience tables can be deleted. Note that this will turn off adaptation for the node associated with the experience table and the initial conditional distribution will be equal to conditional distribution of the node at deletion time.

The adaptation algorithm also provides an optional *fading* feature. This feature reduces the influence of past (and possibly outdated) experience in order to let the domain model adapt to changing environments. This is achieved by discounting the experience count Alpha_i by a fading factor Delta_i, which is a positive real number less than but typically close to 1. The true fading amount is made proportional to the probability of the parent configuration in question.

To be precise: if the *i*th parent given the propagated evidence is p_i, then Alpha$_i$ is multiplied by (1-p_i)+p_i Delta$_i$ before adaptation takes place. Note that the experience counts corresponding to parent configurations that are inconsistent with the propagated evidence (i.e., configurations with p_i = 0) remain unchanged. The fading for a given discrete chance node is stored in a fading table, where each value in the table corresponds to a parent configuration. The fading factor Delta$_i$ can be set to 1: this implies that cases are accumulated (that is, no fading takes place). Setting Delta$_i$ to a value greater than 1 or less than or equal to 0 will disable adaptation for the ith parent configuration (just as setting Alpha$_i$ to an invalid value will do).

To activate adaptation, the *Adaptation Button* (page 157) may be used.

Adaptation is not supported for continuous nodes.

4.4.2 Online EM Adaptation

The Online EM adaptation algorithm in HUGIN is an adaptation algorithm which - like the regular adaptation algorithm serves to update the conditional probability distribution in the light of inserted and propagated evidence (i.e., experience).

Find more details on how Online EM Adaptation works by turning the HUGIN C API Help Manual localized in the folder where you installed HUGIN.

The Online EM adaptation algorithm implemented in HUGIN is developed by O. Cappe and E. Moulines(2009).

Configuration Parameters Configuration parameters for Online EM Adaptation are found in the Network Properties Pane under the Online EM Adaptation tab.

Figure 1: Configuration parameters for Online EM Adaptation.

4.4.3 Propagation

Propagation describes the act of computing posterior probability distributions and expected utilities for unobserved *discrete chance nodes* (page 223) and *decision nodes* (page 224), and distribution functions for *continuous chance nodes* (page 224) given the values for the observed nodes (i.e., the evidence).

Four different propagation algorithms are available: *Sum normal* (page 250), *max normal* (page 251), *sum fast retraction* (page 251), and *max fast retraction* (page 251). Each one of these can be activated by selecting the appropriate Propagate item of the *Network Menu* (page 185). In addition, two of the most common propagation algorithms (sum normal and max normal) can be accessed by pressing one of the propagation buttons of the *Run Mode* (page 123) Tool Bar (see Figure 1). If *auto propagation* (page 129) has been specified, propagation will take place each time new evidence is entered.

Figure 1: The propagation button *sum normal* (page 250) in the run mode toolbar or in th propagation menu in the network menu

4.4.4 PC Algorithm

Two constraint-based algorithms are available for structure learning: The PC algorithm and the *NPC algorithm* (page 246).

The HUGIN PC algorithm, which is a variant of the original PC algorithm due to *Spirtes, Glymour & Scheines (2000)* (page 541), belongs to the class of constraint-based learning algorithms. The basic idea of these algorithms is to derive a set of conditional independence and dependence statements (CIDs) by statistical tests.

The algorithm performs the following steps:

- Statistical tests for conditional independence are performed for all pairs of variables (except for those pairs for which a structural constraint has been specified – for more information, see the help page for the Structural Constraints page of the Learning Wizard).
- An undirected link is added between each pair of variables for which no conditional independences were found. The resulting undirected graph is referred to as the skeleton of the learned structure.
- Colliders are then identified, ensuring that no directed cycles occur. (A collider is a pair of links directed such that they meet in a node.) For example, if we find that A and B are dependent, B and C are dependent, but A and C are conditionally independent given S, not containing B, then this can be represented by the structure A –> B <– C.
- Next, directions are enforced for those links whose direction can be derived from the conditional independences found and the colliders identified.
- Finally, the remaining undirected links are directed randomly, ensuring that no directed cycles occur.

One important thing to note about the PC algorithm is that, in general, it will not be able to derive the direction of all the links from data, and thus some links will be directed randomly. This means that the learned structure should be inspected, and if any links seem counterintuitive (e.g., sweat causes fever, instead of the other way around), one might consider using the *Learning Wizard* (page 267) (which provides a means of specifying structural domain knowledge) or the *NPC algorithm* (page 246), which allows the user to interactively decide on the directionality of undirected links.

Traditional constraint-based learning algorithms produce provably correct structures under the assumptions of infinite data sets, perfect tests, and DAG faithfulness (i.e., that the data can be assumed to be simulated from a probability distribution that factorizes according to a DAG). In the case of limited data sets, however, these algorithms often derive too many conditional independence statements. Also, they may in some cases leave out important dependence relations.

Generally, it is recommended to use the NPC algorithm, as the resulting graph will be a better map of the (conditional) independence relations represented in the data. In particular, when the data set is small, the NPC algorithm should be the one preferred. The NPC algorithm, however, has longer running times than the PC algorithm.

4.4.5 NPC Algorithm

Two constraint-based algorithms are available for structure learning: The *PC algorithm* (page 246) and the NPC algorithm.

NPC stands for *Necessary Path Condition*, and it is a criterion developed by researchers at Siemens in Munich for solving some of the problems of constraint-based learning algorithms like the PC algorithm. The basic machinery is the same in the PC and the NPC algorithms (i.e., they are both based on generating a skeleton derived through statistical tests for conditional independence).

The NPC algorithm seeks to repair the deficiencies of the PC algorithm, which occur especially in the face of limited data sets. The solution provided by the NPC algorithm is based on inclusion of a criterion known as the *necessary path condition* (page 247). This criterion forms the basis for introducing the notion of *ambiguous regions* (page 247), which in turn provide a language for selecting among sets of inter-dependent uncertain links. The resolution of ambiguous regions is performed in interaction with the user.

In constraint-based learning algorithms, the skeleton of the graph is constructed by not including a link in the induced graph whenever the corresponding nodes are found to be conditional independent. There can, however, be inconsistencies among the set of conditional independence and dependence statements (CIDs) derived from limited data sets. That is, not all CIDs can be represented simultaneously. The inconsistencies are assumed to stem solely from sampling noise; i.e., we still assume that there exists a perfect independence map of the (unknown) probability distribution from which the data is generated.

The number of inconsistencies in the set of CIDs reflects structural model uncertainty. Thus, the number of uncertainties is a confidence measure for the learned structure and can as such be used as an indication of whether or not sufficient data has been used to perform the learning. The inconsistent CIDs produce multiple solutions when inducing a directed acyclic graph (DAG) from them. These solutions differ with respect to the set of links included.

To resolve the inconsistencies, the NPC algorithm relies on user interaction where the user gets the opportunity to decide on directionality of undirected links and to resolve the *ambiguous regions* (page 247).

The Necessary Path Condition

Informally, the necessary path condition says that in order for two variables X and Y to be independent conditional on a set S, with no proper subset of S for which this holds, there must exist a path between X and every Z in S (not crossing Y) and between Y and every Z in S (not crossing X). Otherwise, the inclusion of Z in S is unexplained. Thus, in order for an independence statement to be valid, a number of links are required to be present in the graph.

Ambiguous Regions

When the absence of a link, a, depends on the presence of another link, b, and vice versa, we define a and b to be inter-dependent. Both a and b constitute what we call uncertain links. An ambiguous region is a maximal set of inter-dependent links. That is, an ambiguous region consists of a set of uncertain links.

The main goal is to obtain as few and small ambiguous regions as possible. It should be noted that deterministic relations between variables will also produce ambiguous regions (see example below).

If there are some uncertain links (or links that need to be directed), the user is offered the possibility of providing information as to how the ambiguous regions should be resolved. A special, intuitive graphical interface is provided for this interaction.

Resolving Ambiguous Regions and Undirectedness

When using the NPC algorithm, an intuitive graphical interface is provided for resolving the structural uncertainties (if any) found by the algorithm. Figure 1 shows an example of a structure with unresolved uncertainties.

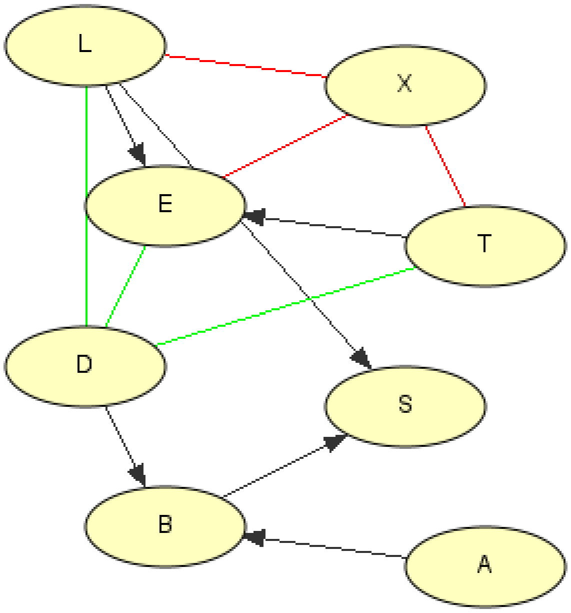
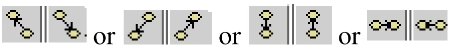


Figure 1: The structural uncertainties found by the NPC algorithm. Black (undirected) links are selectable and their directionality can be determined by the user. The links of each ambiguous region are drawn in the same color. These links are also selectable and will be removed or kept, depending on the action performed by the user.

Should you wish not to provide such information, just click the Done button, and the NPC algorithm will resolve the uncertainties. Note, however, that directionality for undirected links will be decided on a random basis, and that for those ambiguous regions with no information provided, all of the uncertain links will be removed, possibly resulting in a poor model.

Resolving Undirected Links Instead of randomly determining the directionality of the links of the learned structure that cannot be determined automatically from the data, the NPC algorithm gives the user the opportunity to determine the directionality of such links.

The undirected links (not belonging to ambiguous regions) are drawn in black. When selected, such a link gets highlighted and the two directionality buttons get enabled:



Which of the above appearances of the pair of buttons are used depends on the relative position of the nodes of the selected link. Thus, it should be readily apparent which of the two buttons should be selected for achieving the desired directionality for the selected link. However, when positioning the mouse cursor on top of a button, a tool tip will appear after a short while, explaining precisely which direction is associated with the button in question.

Note that assigning directionality to a link may cause other links to automatically be directed. If, for example, there are undirected links between variables x and y, between y and z, and between x and z, then enforcing the directionalities x –> y and y –> z implies x –> z; otherwise, a directed cycle would occur.

Resolving Ambiguous Regions

An ambiguous region consists of a set of inter-dependent uncertain links: Absence of a link in an ambiguous region depends on the presence of one or more of the other links of the region, and vice versa.

Each ambiguous region is easily identified as consisting of links in the same color. (Note that there can be so many ambiguous regions that colors have to be reused, making it difficult from the coloring alone to distinguish them.) Also, when selecting a link of an ambiguous region, all links of the region will be highlighted; the one selected will be drawn bold and the other links will be drawn with double lines.

When a link of an ambiguous region has been selected, the include/exclude link buttons get enabled:

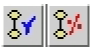


Figure 3: The include/exclude link buttons.

When a decision has been made that a link should be present or absent, each of the other links of the ambiguous region will be affected in one of several ways:

- Unaffected.
- Disappear.
- Turned into an undirected link, not belonging to the region anymore.
- Turned into a directed link, not belonging to the region anymore.

Which of these consequence will be observed depends on the conditional independence and dependence statements (CIDs) found from the statistical tests performed by the NPC algorithm.

Identify Minimal Resolutions

An ambiguous regions may consist of a large number of uncertain edges. This implies that it ay be very difficult for the user to find an appropriate resolution of an ambiguous region.

Each ambiguous region will have at least one minimal resolution. A minimal resolution is a minimal set of edges, which resolves the ambiguous region.

Figure 4: The identify minimal resolutions button.

When an uncertain link of an ambiguous region has been selected the button shown in Figure 4 appears. If this button is depressed, a window showing all minimal resolutions is displayed to the user. The user may select an minimal resolution.

Undo

Often it is impossible (or very hard) to predict the consequences of a decision made regarding directionality of an undirected link, or regarding the presence or absence of a link of an ambiguous region. Therefore, you will probably find the undo facility quite useful.

The most recent (non-undo) action performed can be undone by clicking the undo button:

Figure 5: The undo button.

Please note that an arbitrary number of operations performed can be undone by repeatedly clicking the undo button.

References

The interested reader is referred to the literature for a more detailed description of the notions behind the NPC algorithm. See e.g.

- Steck, H. (2001). *Constrained-Based Structural Learning in Bayesian Networks Using Finite Data Sets*, PhD Thesis, Institut für der Informatik der Technischen Universität München.
- Steck, H. and Tresp, V. (1999). Bayesian Belief Networks for Data Mining, Proceedings of *The 2nd Workshop on Data Mining und Data Warehousing als Grundlage Moderner Entschidungsunterstuezender Systeme*, DWDW99, Sammelband, Universität Magdeburg, September 1999.
- Steck, H., Hofmann, R., and Tresp, V. (1999). *Concept for the PRONEL Learning Algorithm*, Siemens AG, Munich.

4.4.6 Prop Sum Normal

The prop sum normal propagation is activated either by selecting the Prop Sum Normal item of the *Network Menu* (page 185) or by pressing the corresponding button in the *Run Mode* (page 123) Tool Bar (see Figure 1). If you are using *auto propagation* (page 129), you can specify HUGIN to use this propagation method.

The "prop sum normal" is the most commonly used propagation method. It updates all probabilities, distribution functions, and expected utilities of the *discrete chance nodes* (page 223), the *continuous chance nodes* (page 224), and the *decision nodes* (page 224), according to entered evidence.

Figure 1: The prop sum normal button from the Run Mode Tool Bar.

A propagation of evidence does not update policies in a (limited-memory) influence diagram. To update policies *Single Policy Updating* (page 258) should be used.

4.4.7 Prop Max Fast Retract

The *prop max fast retract* propagation is activated by selecting the Prop Max Fast Retract item of the *Network Menu* (page 185). If you are using *auto propagation* (page 129), you can specify HUGIN to use this propagation method.

The prop max fast retract propagation method can be used in HUGIN networks (*Bayesian networks* (page 345), *influence diagrams* (page 345), or *object-oriented networks* (page 346)) containing only *discrete chance nodes* (page 223). It is similar to the *max normal* (page 251) method except that it shows the nodes which have received evidence differently. Instead of showing the values of the states of the node which has received evidence, the values of the states are shown as if this node had not received evidence. That is, the values of the node being in each of its states given the evidence entered in all other nodes. To indicate that the node has in fact received evidence, the belief bars use the color code for evidence (default: Red).

This propagation method can be used to check if the evidence corresponds to (one of) the most likely configuration(s) or if the evidence suggests that one of the less likely configurations has been observed (if a state of a node has been observed which does not have the value 100).

4.4.8 Prop Sum Fast Retract

The prop sum fast retract propagation can be activated if you are using auto propagation, you can specify HUGIN to use this propagation method. The “prop sum fast retract” propagation method can be used in HUGIN networks (Bayesian networks object-oriented Bayesian networks) containing only discrete chance nodes. It is similar to the sum normal method except that it shows the nodes which have received evidence differently. Instead of showing the true posterior probability distribution for the node, the distribution is shown as if the node had not received evidence. That is, the distribution shown is the posterior probability distribution given the evidence entered in all other nodes. To indicate that the node has in fact received evidence, the belief bars are painted using the color code for evidence (default: Red).

This propagation method can be used to check if the evidence entered in a node meets the probabilities of the evidence or if there are signs of some kind of conflict (the evidence entered is one of the most unlikely states of the node).

4.4.9 Prop Max Normal

The *prop max normal* propagation is activated either by selecting the Prop Max Normal item of the *Network Menu* (page 185) or by pressing the corresponding button in the *Run Mode* (page 123) Tool Bar (see Figure 1). If you are using *auto propagation* (page 129), you can specify HUGIN to use this propagation method.

The prop max normal propagation method can be used in HUGIN networks (*Bayesian networks* (page 345), *influence diagrams* (page 345), or *object-oriented networks* (page 346)) containing only discrete chance nodes to find states belonging to the most probable configuration (a configuration is a list of states (a_1, a_2,. . . , a_n) of the list of all nodes in the network (A_1, A_2,. . . , A_n)).

If a state of a node belongs to the most probable configuration it is given the value 100. All other states are given the relative value of the probability of the most probable configuration they are found in compared to the most probable configuration. That is, assume a node N has two states a and b, and b belongs to the most probable configuration of the entire network which has the probability 0.002. Then, b is given the value 100. Now, assume that the most probable configuration which a belongs to has probability 0.0012. Then, a is given the value 60.

The prop max normal button which is found in the *network window* (page 182) Tool Bar in *Run Mode* (page 123) is shown in Figure 1.

Figure 1: The prop max normal button from the Run Mode Tool Bar.

4.4.10 Hierarchical Naive Bayes

Naive Bayes models are a popular type of classification model. The *target* (or *class*) variable has a state for each possible class. The attribute nodes are the observations used to decide the class. In a Naive Bayes model, the attribute nodes are children of the target node. When this structure hs been created and case data has been entered, it is simply a matter of running the *EM algorithm* in order to estimate the parameters of the model.

One of the problems with the Navie Bayes model is that some of the attribute variables can be more or less correlated. If two attribute variables are strongly correlated, then their combined effect on the target variable can be overestimated. We can compensate for this overestimation by introducing an extra (latent)variable, this variable will be the (only) parent of the two correlated attribute variables, and the latent variable will be a child of the target variable (or some other latent variable). This is called a *Hierarchical Naive Bayes* model by Langseth and Nielsen (2006).

HUGIN provides a simplified version of the algorithm proposed by Langseth and Nielsen (2006). The algorithm successively identifies pairs of variables to be combined via a latent variable.

One iteration of the algorithm follows this scheme:

- **Identify a pair of variables[8] that are strongly**

  correlated given the *target* variable. This is done using the same test that is used to determine [in]dependence in the PC algorithm. The pair with the highest score (correlation) is selected.

- The algorithm stops when no pair of correlated variables can be found, that is, the test of the PC algorithm indicates *independence* for all remaining pairs of variables. The threshold value for this decision is controlled by *significance level*.

- Let *X* and *Y* be the selected pair of varaibles. Create a latent variable *L* with *X* and *Y* as children. The states of *L* are defined as one state for each combination of states of *X* and *Y*, and the conditional probability tables for *X* and *Y* are defined (according to the meaning of the states of *L* to be deterministic functions of *L*. Then it is tested (see Langseth and Nielsen (2006)) for details) whether some of the states of *L* contribute the *same information* with respect to the classification task.[9] These states are then *collapsed* to a single state. An additional latent node *L'* is created with *L* as the only child to represent the collapsed state space, and a conditional probability table for *L* is defined in the natural way.[10] Since the state spaces of *L* and *L'* are deterministically determined by *X* and *Y*, we can create case data for *L* and *L'* to be used in the following iterations of the algorithm.

  Finally, *X* and *Y* are removed from the list of eligible variables from which the next pair of variables will be chosen, and *L'*[11] is added to that list.

When the algorithm above stops, the nodes (except the *target* node) without parents will be given *target* as parent. These nodes also have no conditional probability table. In order to finalize the model, these nodes can be equipped with experience tables before estimating the CPTs using the EM algorithm. The last part must be done explicitly (i.e., parameter estimation is not performed as part of the algorithm).

Authors

- H. Langseth, T. D. Nielsen (2006). *Classification using Hierarchical Naive Bayes models*.

[8] The list of eligible variables initially contains all non-*target* variables.

[9] If *X* and *Y* are irrelevant to the classification task, then all states of *L* will be collapsed to a single state.

[10] If the collapsed state space is identical to the original state space, *L'* will not be created.

[11] If *L'* has not been created, then *L* is used instead.

4.4.11 Rebane-Pearl Polytree

The Rebane-Pearl polytree algorithm constructs a polytree-shaped belief network approximation to a probability distribution (Rebane and Pearl, 1987).

The Rebane-Pearl algorithm is similar to the *Chow-Liu tree algorithm* (page 254) for constructing the "best possible" tree-shaped belief network approximation to a probability distribution such that all edges are directed away from the root. Instead of constructing a tree-shaped belief network approximation, the algorithm constructs a polytree-shaped belief network.

Rebane-Pearl polytree recovery algorithm: The Chow-Liu algorithm constructs a simple tree structure (each node has one parent — except the root). The Rebane-Pearl algorithm turns this tree into a polytree (each node can have multiple parents, but the underlying network structure is still a tree).

References

- C. K. Chow, C. N. Liu (1968). *Approximating Discrete Probability Distributions with Dependence Trees*.
- G. Rebane, J. Pearl, (1987). *The recovery of causal polytrees from statistical data*.

4.4.12 Tree Augmented Naive Bayes

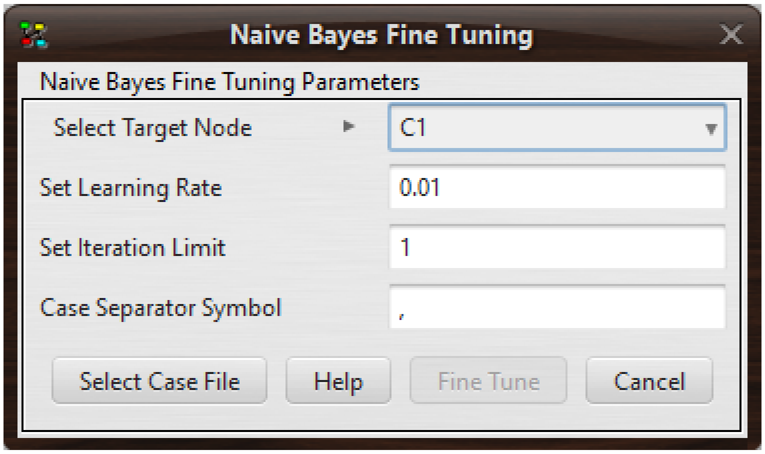
The Tree Augmented Naive (TAN) Bayes algorithm is based on the *Chow-Liu algorithm* (page 254).

The TAN algorithm is useful for constructing classification networks where a specific node of the model is target of the reasoning. The target node is used to construct a *conditional* Chow-Liu tree (i.e., a Chow-Liu tree over all nodes except the selected *target*) with the selected *root* as root of the tree. Weights are defined as conditional mutual information (conditional on the target), and all nodes (except *target*) have target as an extra parent.

4.4.13 Naive Bayes Fine Tuning

The Fine-Tuning Naive Bayes (FTNB) function is based on the algorithm of Khalil El Hindi (*Fine tuning the Naive Bayesian learning Algorithm*, in *AI Communications*, 27(2):133–141, 2014). The aim of the FTNB algorithm is to improve the classification accuracy of a Naïve Bayes classifier by adjusting some the entries of the conditional probability distributions of the feature nodes. Iterating over the training cases, we compute the classfication of each case according to the model. If the classfication is incorrect, we change some of the conditional probabilities used in this classification to increase the likelihood of a correct classification. The conditional probabilities are updated after each incorrectly classified case implying the order of the cases matters.

After a full iteration over all cases, the improvement of the model is computed. The FTNB algorithm continuous until no improvement occurs (or a few iterations after no improvement). The FTNB algorithm assumes that the model is a Naïve Bayes and takes four arguments: 1) the target node, 2) a learning rate, 3) an upper limit on the number of iteration after no improvement, and 4) a case separator symbol (the separator used in the data file). Figure 1 displays the NB Fine Tuning dialogue. It can be found in the network menu item under the learning submenu.



Naive Bayes Fine Tuning Parameters

Select Target Node: C1

Set Learning Rate: 0.01

Set Iteration Limit: 1

Case Separator Symbol: ,

Figure 1: Naive Bayes Fine Tuning Dialog.

4.4.14 Chow-Liu Tree

A Chow-Liu tree is the "best possible" tree-shaped belief network approximation to a probability distribution such that all edges are directed away from the root. The quality of the approximation is measured using the Kullback-Leibler distance between the true distribution and the distribution defined by the Chow-Liu tree. When we learn from data, the "true" distribution is defined by the frequencies of the observations.

Chow and Liu (1968) show that the optimum tree can be found as the maximum-weight spanning tree over all variables, where the weight of each edge is given as the mutual information between the variables connected by the edge.

A Chow-Liu tree can be constructed as follows:

- Calculate the mutual information MI(X_i, X_j) for each pair (X_i, X_j).
- Consider the complete MI-weighted graph: the complete undirected graph over $\{X,\ldots,X_n\}$, where the links (X_i, X_j) have the weight MI(X_i, X_j).
- Build a maximal-weight spanning tree for the complete MI-weighted graph.
- Direct the resulting tree by choosing any variable as a root and setting the directions of the links to be outward from it.

Authors

- C. K. Chow, C. N. Liu (1968). *Approximating Discrete Probability Distributions with Dependence Trees*.

4.4.15 Junction Tree

Inference in Bayesian networks and influence diagrams are performed in a secondary structure known as a *junction tree*. The junction tree for a network is generated when one switches from *Edit Mode* (page 124) to *Run Mode* (page 123). The junction tree of the network can be displayed in a separate window, opened by selecting the Show Junction Tree item of the *View Menu* (page 188).

Figure 1 shows a junction tree for the `ChestClinic network`.

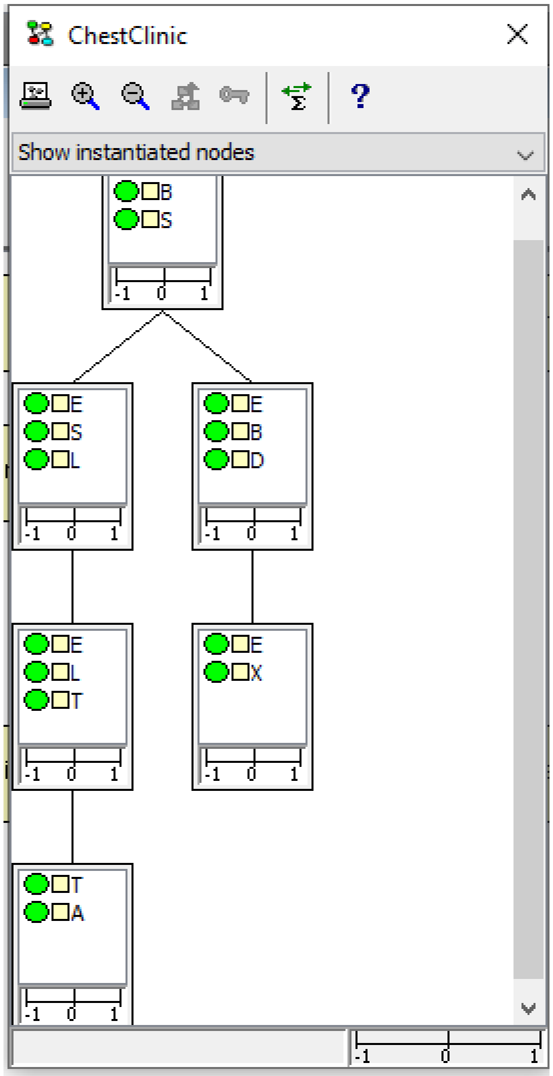


Figure 1: Junction tree for the ChestClinic network.

To understand what a junction tree is and what it can be used for, there are several issues that may be of interest:

- *What is a Junction Tree?* (page 256) What is it and how is it generated?
- *Inference in a Junction Tree:* (page 256) How is a junction tree used for inference?
- *Tools and Functionalities:* (page 257) How to use the tools and functionalities offered?

Also of interest is how one can use a junction tree to detect and trace *conflicting evidence* (page 292).

What is a Junction Tree?

Inference in a network is conducted through message passing in a junction-tree representation of it. The junction tree is obtained through transformation that involves the processes of moralization and triangulation. It can be shown that exact inference in a Bayesian network requires (implicit or explicit) moralization and triangulation.

In the moralization step, an undirected link is first added between each pair of unconnected parent nodes that have a common child node. Next, all directed links are converted to undirected links. This process results in a so-called *moral graph*.

The triangulation step adds (undirected) links to the moral graph such that the resulting graph is *triangulated (or chordal)*. An undirected graph is chordal whenever it does not contain a cycle (i.e., a path of distinct nodes, except that the first and the last nodes are identical) of length greater than 3 without a chord (i.e., a link between non-consecutive nodes of the path). The maximal completely connected components or sub-graphs (i.e., where each pair of nodes are connected to each other) of a graph are called *cliques*.

It is the cliques of the triangulated graph that form the nodes of the junction tree. Each clique has associated with it a table with indices spanning the Cartesian product of the state spaces of the nodes of the clique. Thus, each clique holds a representation of a function over all nodes of the clique. That is, a joint probability distribution over the nodes of the clique can be represented in the clique table.

Obviously, as each clique has associated with it a table that grows exponentially in size as more nodes are added to it, we want the cliques to be as small as possible. However, as the number of different triangulations may be as high as *n* factorial, where *n* is the number of nodes of the network, and as the representational and computational complexities of the resulting junction trees vary heavily over the different triangulations, it can be very difficult to find an optimal triangulation. In fact, the complexity of this optimization problem is NP-hard (mathematical way to say that it is impossible within reasonable time).

The HUGIN Decision Engine (HDE) offers a number of different triangulation algorithms, most of which are based on greedy one-step look ahead heuristics. In many cases, however, an optimal (or near-optimal) triangulation can in fact be found. The HDE offers a method for (near-)optimal triangulation, which in many cases results in junction trees of much lower complexity than those generated using the other (heuristic) triangulation methods provided. For details, see the description of the *compile* (page 126) function.

Please note that if the Bayesian network is disconnected, so is the junction tree. A disconnected junction tree is sometimes called a *junction forest*.

Inference in a Junction Tree

As mentioned above, inference in a network amounts to message passing in a junction-tree representation of the network.

An inference step (also referred to as a propagation) consists of two main phases:

- *Inward message* passing in which messages are sent from the leaves of the junction tree towards a root of the tree.
- *Outward message* passing in which messages are sent from the root of the tree towards to leaves.

A clique, C, are allowed to send a message to a neighboring clique, D, whenever C has received messages from all other neighbors and it has not already sent a message to D.

A message from a clique C to a neighboring clique D consists of a *marginal table* over the nodes that are members of both C and D. More precisely, the message is computed from the table of C by eliminating the dimensions corresponding to the nodes of C that are not members of D. Typically, elimination is performed through summation. To compute the most likely configuration (also known as the *most probable explanation* (MPE)) and the probability of this configuration, elimination is, however, performed through maximization.

Tools and Functionalities

The cliques of the junction tree(s) are displayed as rectangles (see Figure 2), where the upper part of the rectangle contains a list of the nodes that are members of the clique, and the lower part contains a conflict bar indicating the conflict measure pertaining to the evidence entered in the sub-tree rooted at the clique. For details on data conflict analysis, please consult the *data conflict analysis* (page 292) section.

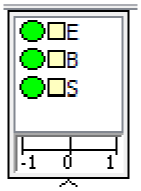


Figure 2: A clique of the ChestClinic junction tree.

A clique can be selected by clicking the left mouse button. When selected the border of a clique change from gray to black. It is possible to select multiple cliques by holding down the shift key while clicking. When selecting one or more cliques, the nodes of the selected cliques will get selected in the *Network Panel* (page 177).

The Junction Tree Panel offers a number of other functionalities, which are available through a tool bar of buttons and a pull-down menu (see Figure 3).

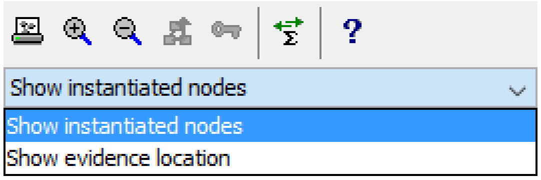


Figure 3: A clique of the Asia junction tree.

The printer button enables print out of the junction tree. The junction tree can be printed either to a printer, or to a file. Similarly to *printing of the networks* (page 168), it is possible to split the printout into several pages by setting an appropriate scale factor in a "Junction Tree" tab in the print dialog.

The magnification glass buttons offer possibilities to zoom in and out for better viewing.

The button with the symbol offers a possibility to select a different clique as root of the tree. The functionality gets enabled only when exactly one clique is selected. Selecting a different clique as root can be useful in the investigation of local conflicts.

The button with the 🔑 symbol gets enabled when the global conflict measure is positive. See the *data conflict analysis* (page 292) for details.

The button with the Σ symbol provides the usual sum propagation functionality.

The button with the ? symbol activates a help page providing much the same information as provided in this and the *data conflict analysis* (page 292) section.

Evidence is indicated by highlighting evidence nodes (the color of their node symbols are changed to the evidence color). Two different modes exist for displaying evidence nodes. Either all instances of the evidence nodes will be highlighted or only those instances that appear in the cliques where the evidence has been entered. (Note that a node can be a member of several cliques.) A pull-down menu (see above) is provided for selection of preferred display mode.

4.4.16 Single Policy Update

A LIMID is solved by Single Policy Updating (SPU). SPU is an iterative algorithm that updates one policy at a time and terminates when all policies have converged (i.e., more iterations change nothing). The algorithm usually finds the globally optimal policies, but it is possible that the algorithm may get stuck at a local maximum.

The parents specified for the decision nodes determine which observations should be taken into account when decisions are made. Ideally, we would specify all observations to be taken into account, but this may not be practical because the size of a policy table is exponential in the number of parents. We therefore often don't specify less important observations as parents of decision nodes (for example, old observations are typically less important than new ones) in order to reduce the size of the policy tables.

Unless all relevant information has been specified as parents, then it can be useful to recompute policies whenever new information becomes available. This is because the computations take all existing observations (in addition to future observations specified as parents of decision nodes) into account when policies are computed.

Propagation of evidence and calculation of posterior distributions are performed under the strategy computed by the latest Single Policy Update (or under the initial strategy specified by the user, if Single Polcy Update has not been performed). SPU assumes that entered evidence has been propagated.

The SPU algorithm computes the probability distribution and expected utility function over the states of each chance and decision node in the LIMID.

Single Policy Updating is invoked by pressing the 'SPU'-button of the *Run Mode* (page 123) Tool bar.

See also *Reset uninstantiated policies* (page 174), *Store policies* (page 174) and *Recall stored policies* (page 173).

Figure 1: The SPU button from the *Run Mode* (page 123) Tool Bar.

4.4.17 Greedy Search-And-Score Structure Learning

The greedy search-and-score algorithm for learning the structure of a Bayesian network uses a score function to evaluate the *goodness* of a candidate network structure. The algorithm performs a search through the space of possible network structures and returns the structure with the highest score. The operators used to perform the search given a current candidate are add an arc, remove an arc and reverse an arc.

The candidate network structure is provided with a maximum likelihood estimate of the conditional probability tables associated with the nodes of the network in order to compute the score of the candidate structure. If the data is incomplete, the EM algorithm must be used. However, if the data is complete, this can be done by counting cases and normalizing the counts.

For simplicity, we assume that the data is complete (by ignoring the missing data).

We also assume that the score function is *decomposable*. This means that the total score can be computed as a sum of scores of the nodes in the network. This makes it easy to compute the effect of a simple modification to the network (such as adding or removing an edge).

The user can chouse between using the *Akaike Information Criterion* (AIC) or the *Bayesian Information Criterion* (BIC) as the score function.

The user can specify an upper limit on the number of parents of each node in the network structure.

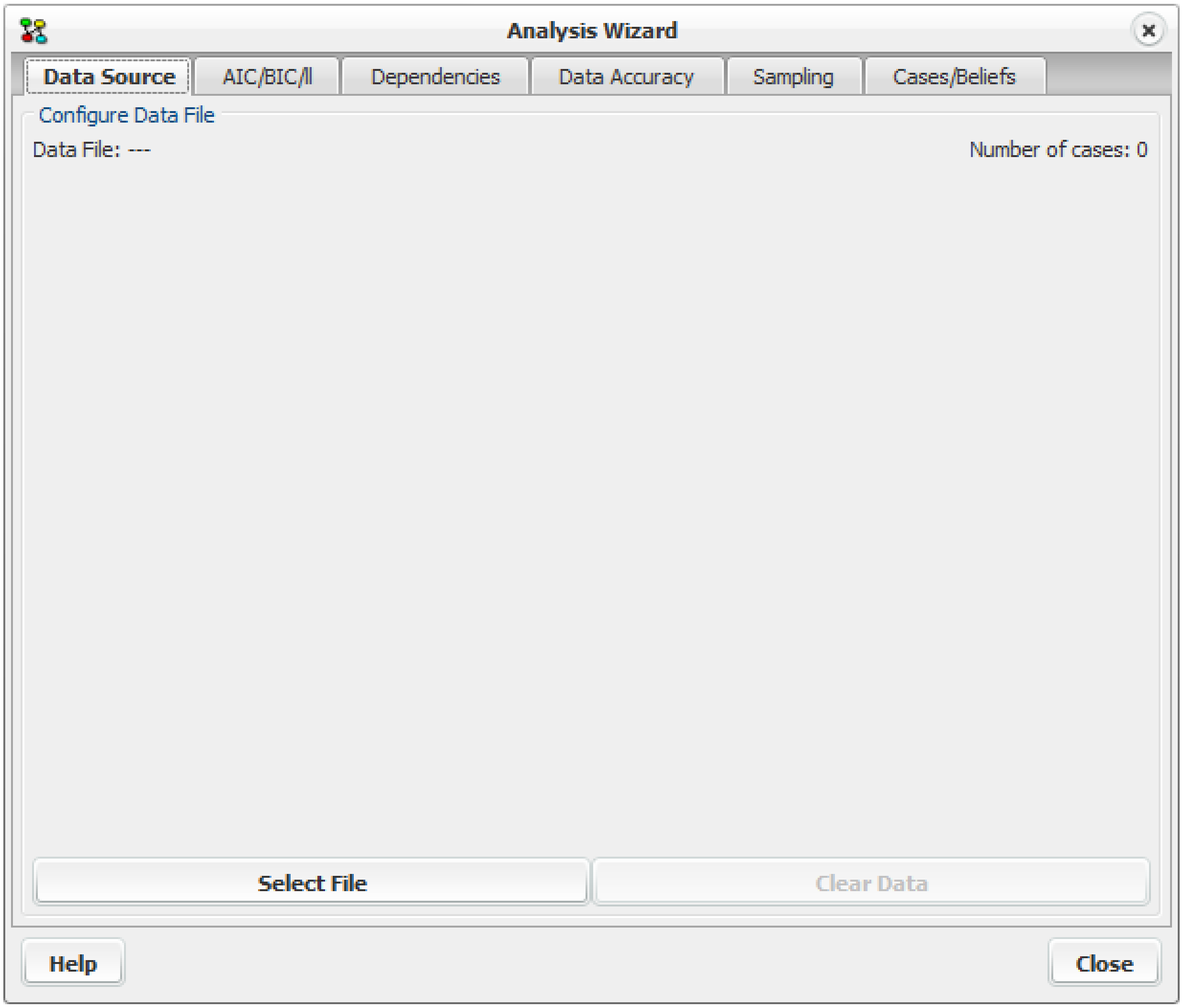
4.5 Wizards

4.5.1 Analysis Wizard

The Analysis Wizard performs several kinds of analysis on a network; Based on a set of cases the wizard will calulate the AIC-, BIC- and log-likelyhood scores, perform various data dependency and accuracy analysis.

The Analysis Wizard is activated by selecting "Analysis Wizard" in the Wizards Menu. *Note that to activate the Analysis Wizard, the network must be in runmode and consist only of discrete chance nodes*.

When the Analysis Wizard is activated a window displaying the "Data Source" pane will pop up, see figure 1. The window has six tabs: "Data Source, "AIC/BIC/ll", "Dependencies", "Data Accuracy", "Sampling" and "Cases/Beliefs".



Data File: ---

Number of cases: 0

Figure 1: Empty Data Source pane; Window displayed when the Analysis Wizard is activated.

Data Source pane

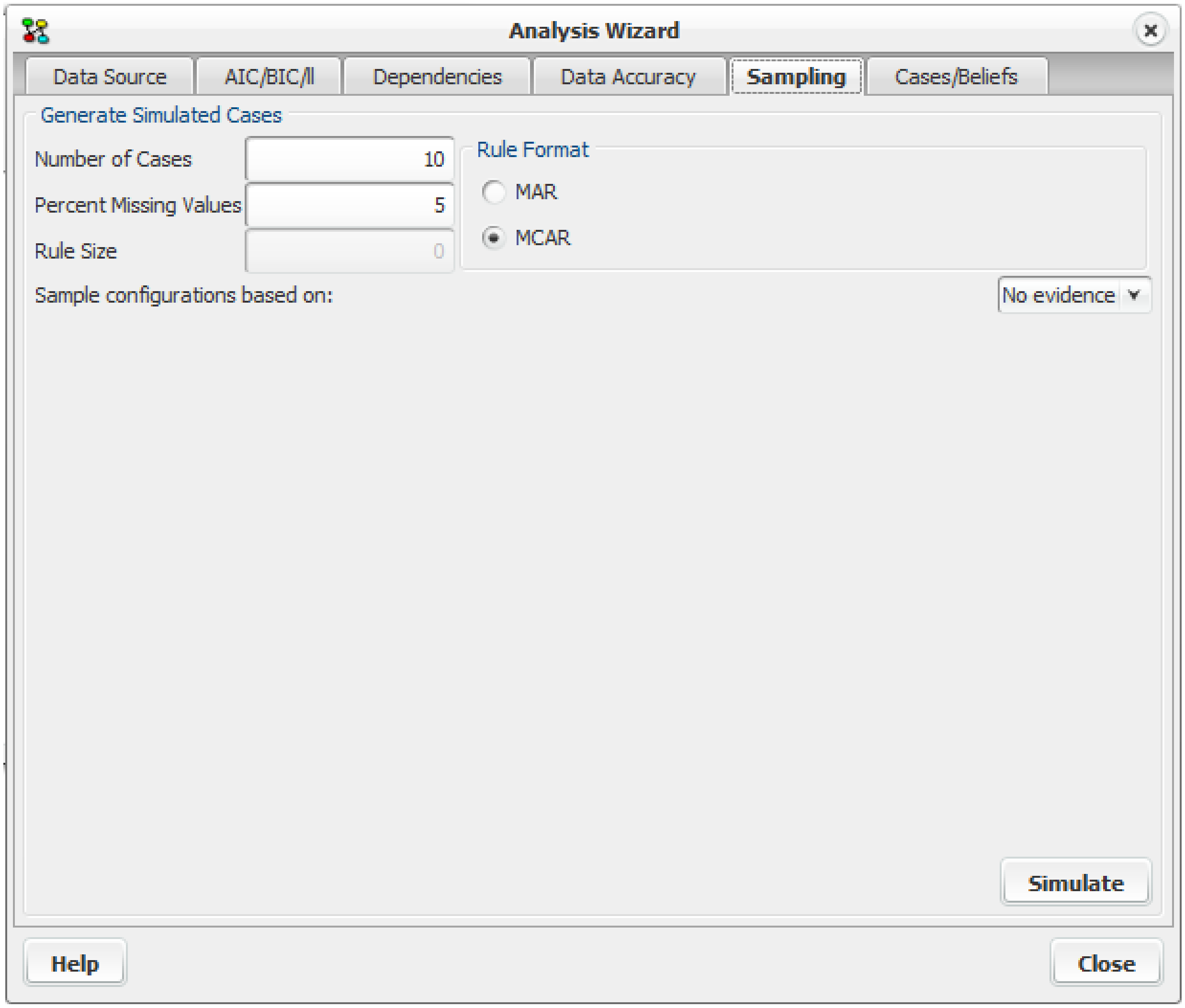
To perform any analysis the wizard will first need a set of cases. Cases can be imported by

- loading a HUGIN data file,
- or sampling a number of random cases.

Click on button “select file” to load a HUGIN data file or click on the tab “Sampling” to sample a number of random cases to use as data source.

Sampling pane

Clicking on the tab “Sampling” will bring forward the sampling pane. The sampling pane is shown in figure 2.



Number of Cases: 10

Percent Missing Values: 5

Rule Size: 0

Rule Format

○ MAR

● MCAR

Sample configurations based on: No evidence

Figure 2: Sampling pane.

Cases can be generated in two different ways: MCAR (Missing Completely at Random) or MAR (Missing at Random). MCAR sets some values to N/A by removing values of some nodes in the generated case set randomly (i.e., MCAR considers all the values in the generated cases and not one case at a time).

MAR randomly sets some values in a case to N/A based on auto-generated templates. These templates specify that if some nodes have a specific value, then the values of a subset of the nodes are randomly set to N/A. Note that the specification of the templates is generated randomly.

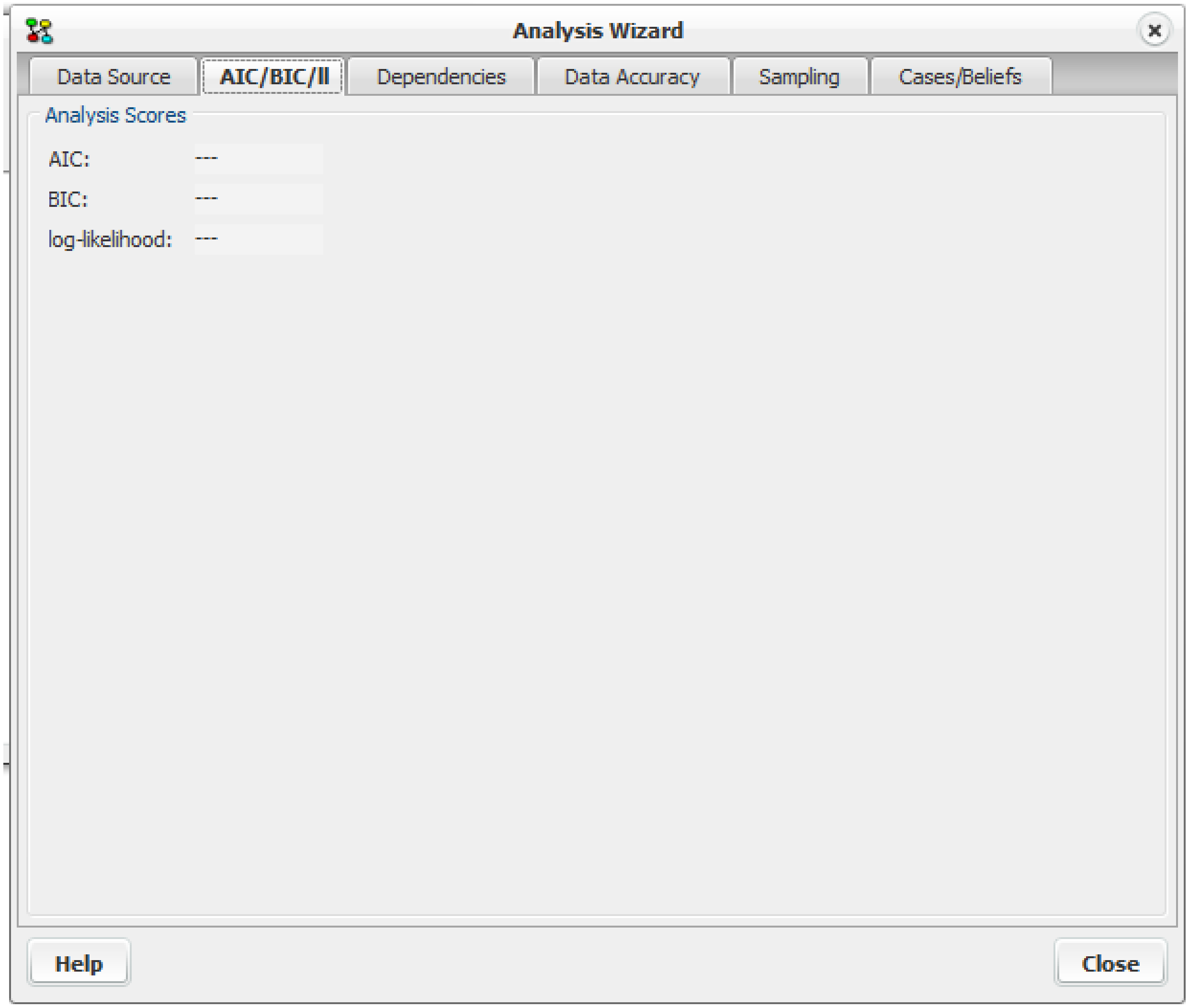
In both cases the conditional probability distribution is considered as a factor to the number of cases generated with a specific set of observations. Also in both MCAR and MAR it is possible to indicate the percentage of missing values.

The cases generated will be based on the conditional probability distribution given by the scenario chosen in the combo box "Sample configurations based on".

Any cases generated will be added to the current set of imported cases.

AIC/BIC/ll pane

Clicking on the "AIC/BIC/ll" tab will bring forward the AIC/BIC/ll pane, see figure 3. The wizard calculates the AIC, BIC and log-likelihood scores of the model given the case data.



AIC: ---

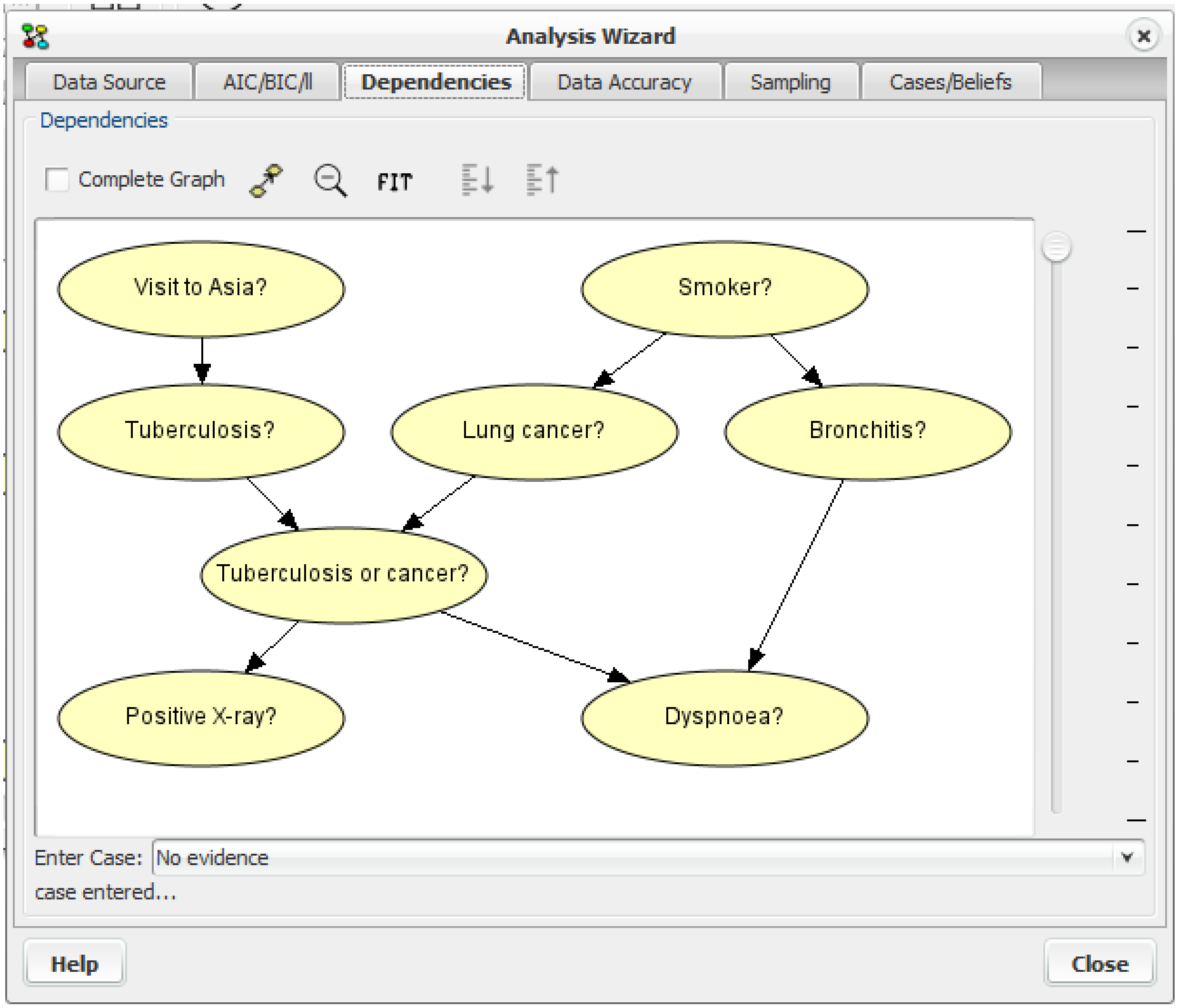
BIC: ---

log-likelihood: ---

Figure 3: AIC/BIC/ll pane.

Dependencies pane

Clicking on the “Dependencies” tab will bring forward the data dependencies pane, see figure 4.



Enter Case: No evidence

case entered...

Figure 4: Data dependencies pane.

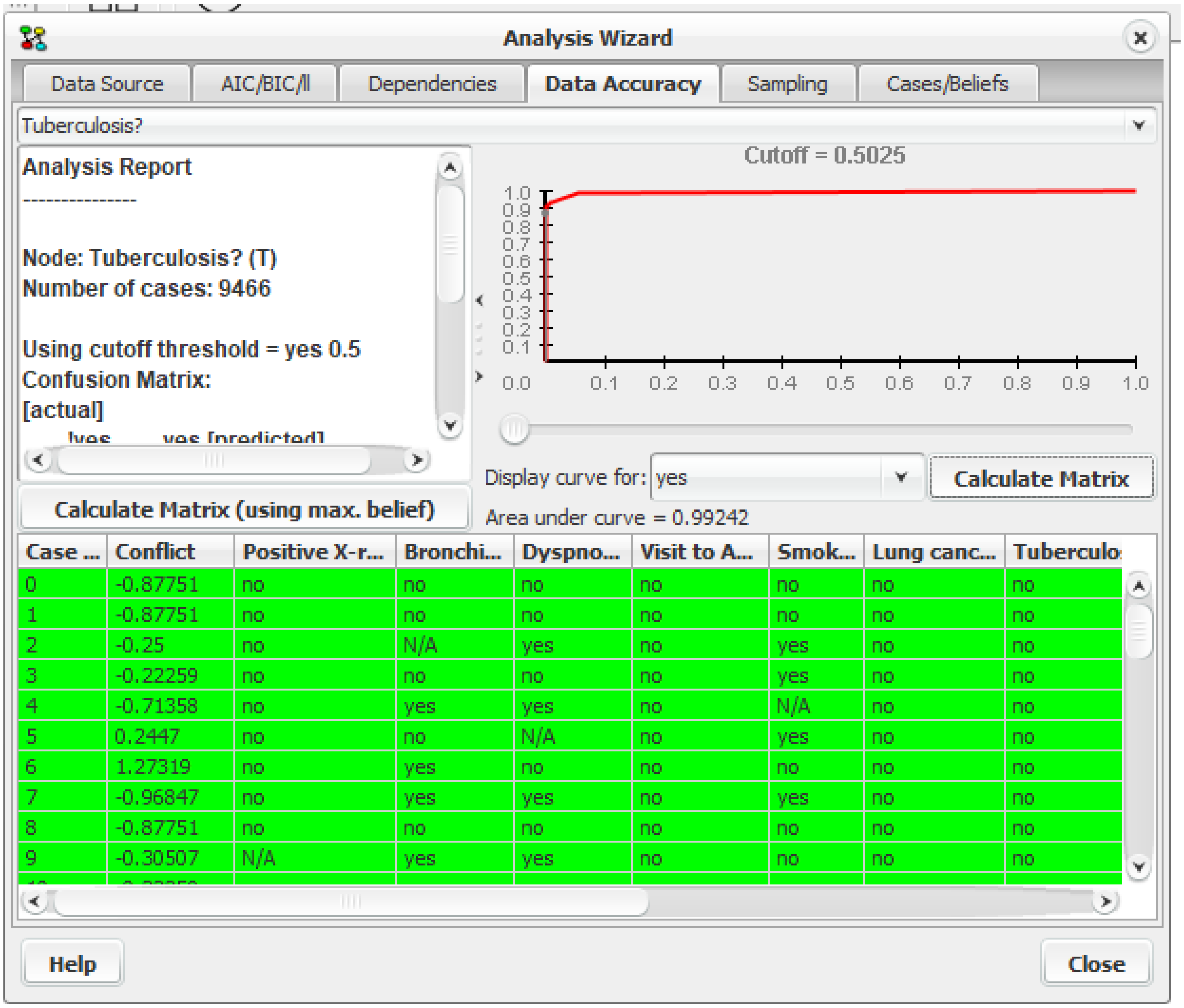
The Dependencies pane allow for inspecting the mutual information of nodes. Depending on the threshold set by the slider, edges will be hidden according to the mutual information of the nodes. Toggle between inspecting mutual information for all nodes or nodes with directed edges by clicking the checkbox “Complete graph”. Mutual information will be computed based on the case selected in the “Enter Case” drop down box.

Buttons at the top of the pane allows for zooming the view of the network and adjusting the scale of the slider if possible.

- Stronger and weaker dependencies can be identified by moving the threshold slider, as edges are only displayed if the mutual information between the connected nodes are above the threshold.
- Mutual information between all nodes can be inspected by clicking ‘Complete Graph’.
- Mutual information computation will be based on the the network instantiation chosen in the ‘Enter Case’ drop-down list

Data Accuracy pane

Clicking on the “Data Accuracy” tab will bring forward the Data accuracy analysis pane, see Figure 5. Given a node and a set of cases, this pane generates an analysis report with information about how well the predictions of the network match the cases.



Tuberculosis?

Analysis Report

Node: Tuberculosis? (T)
Number of cases: 9466

Using cutoff threshold = yes 0.5
Confusion Matrix:
[actual]
|yes yes [predicted]

Cutoff = 0.5025

Display curve for: yes

Area under curve = 0.99242

Case ...	Conflict	Positive X-r...	Bronchi...	Dyspno...	Visit to A...	Smok...	Lung canc...	Tuberculo
0	-0.87751	no	no	no	no	no	no	no
1	-0.87751	no	no	no	no	no	no	no
2	-0.25	no	N/A	yes	no	yes	no	no
3	-0.22259	no	no	no	no	yes	no	no
4	-0.71358	no	yes	yes	no	N/A	no	no
5	0.2447	no	no	N/A	no	yes	no	no
6	1.27319	no	yes	no	no	no	no	no
7	-0.96847	no	yes	yes	no	yes	no	no
8	-0.87751	no	no	no	no	no	no	no
9	-0.30507	N/A	yes	yes	no	no	no	no

Figure 5: Data Accuracy pane.

This pane consists of four elements, a drop down box at the top to select what node to analyze, an ROC curve, an analysis report and a table showing the cases.

Analysis Report

The analysis report list the number of cases used for analysis (only cases without observations of the actual node is ignored). A confusion matrix is generated showing how well the observed states match the predicted. For prediction we can use either of the following options:

1. random state with the highest belief to be the predicted state*.
2. specific state with a belief greater than or equal to the ROC cutoff threshold.

- (if a node has two or more states that qualify to be the predicted state, one of them is randomly selected as the predicted state)

The button *Calculate Matrix (using max. belief)* makes a report where prediction is based on (1). Selecting a specific state and threshold for the ROC curve and clicking the button *Calculate Matrix* makes a report where prediction is based on (2).

The report also contains a number of distance measures, the average euclidian distance and Kulbach-Leibler divergence. The distance measures are from the true distribution **x** inferred from the case data e with respect to the selected node **X**, and the distribution **y** resulting from propagating eX.

The Euclidian distance is computed as:

$$dist_Q(x,y) = \sum_i (x_i - y_i)^2$$

The Kulbach-Leibler divergence is computed as:

$$dist_K(x,y) = \sum_i (log_2 x_i - log_2 y_i)^2$$

The reported distance measure is averaged over all cases where X has been observed.

ROC Curve

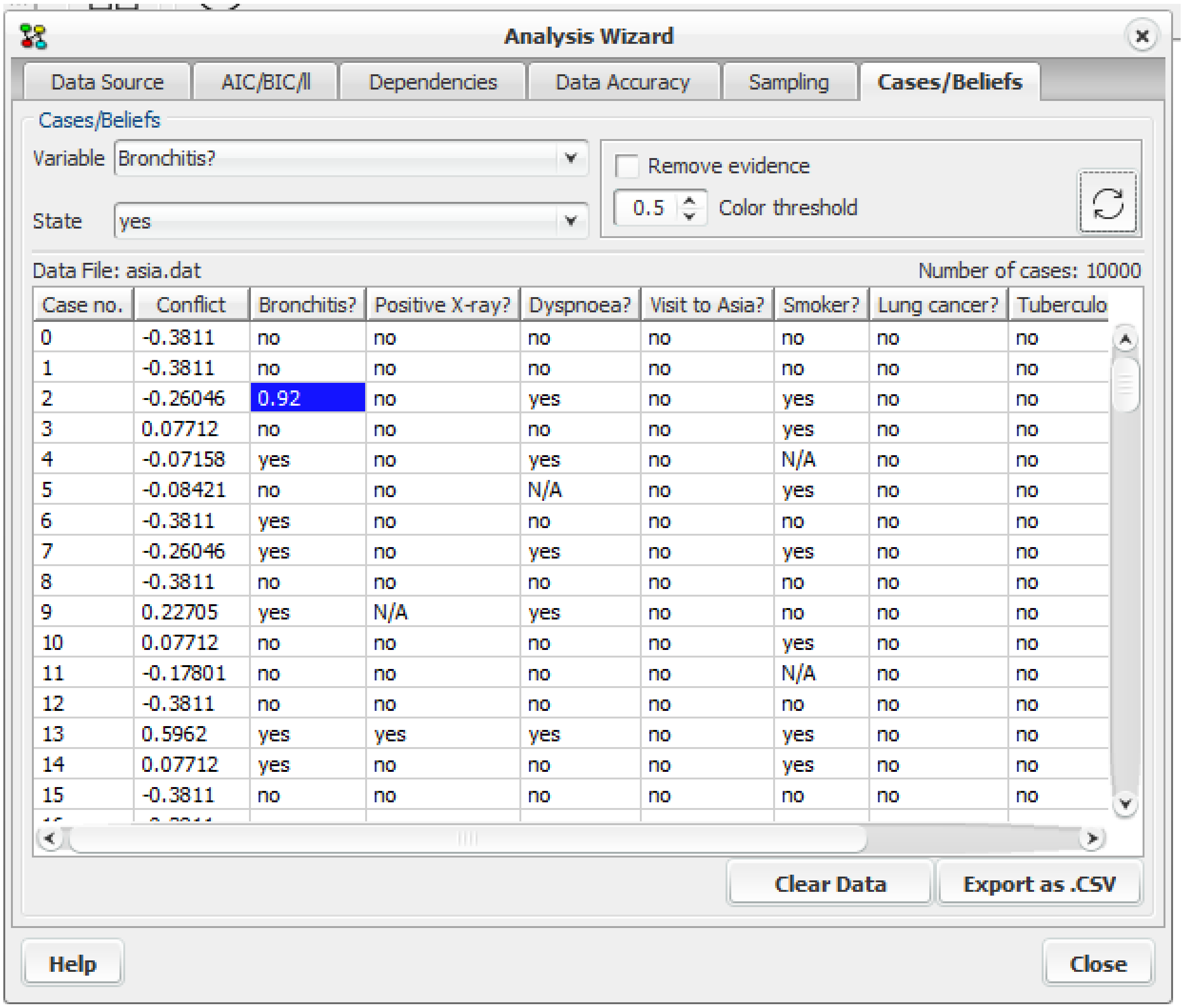
The ROC curve lets you inspect the performance of a given variable as a classifier for the data set. X-axis is the false positive rate, and the Y-axis is the true positive rate. The ROC curve is based on the performance for predicting specific states. The area under the can be used as a measure for the “goodness” of the network as a classifier for the given variable. The button *Calculate Matrix* generates a report based on the selected curve and ROC cutoff threshold as described above in *Analysis Report*. The threshold can be specified by moving the threshold slider left and right. The current threshold is also signified by a point on the curve, signifying the expected performance with regards to the ratio between true and false predictions.

Case table

The table shows the cases used for analysis. The node selected for analysis will always be located in the second last column of the table. The last column in the table report the probability of the node being in the state observed in the case, given the case without information about the state of the node itself. Cases are colored green for a match, red for no match and blue if ignored and orange if unable to make a prediction. A match is when the predicted state for the node is the same state observed in the case.

Case Beliefs pane

Clicking on the “Cases/Beliefs” tab will bring forward the cases pane, see Figure 6. The case pane offers the option of selecting a node and one of it’s states and displaying the beliefs for that state in each case, highlighted by a color.



Variable Bronchitis?

State yes

☐ Remove evidence

0.5 Color threshold

Data File: asia.dat

Number of cases: 10000

Case no.	Conflict	Bronchitis?	Positive X-ray?	Dyspnoea?	Visit to Asia?	Smoker?	Lung cancer?	Tuberculo
0	-0.3811	no	no	no	no	no	no	no
1	-0.3811	no	no	no	no	no	no	no
2	-0.26046	0.92	no	yes	no	yes	no	no
3	0.07712	no	no	no	no	yes	no	no
4	-0.07158	yes	no	yes	no	N/A	no	no
5	-0.08421	no	no	N/A	no	yes	no	no
6	-0.3811	yes	no	no	no	no	no	no
7	-0.26046	yes	no	yes	no	yes	no	no
8	-0.3811	no	no	no	no	no	no	no
9	0.22705	yes	N/A	yes	no	no	no	no
10	0.07712	no	no	no	no	yes	no	no
11	-0.17801	no	no	no	no	N/A	no	no
12	-0.3811	no	no	no	no	no	no	no
13	0.5962	yes	yes	yes	no	yes	no	no
14	0.07712	yes	no	no	no	yes	no	no
15	-0.3811	no	no	no	no	no	no	no

Figure 6: Cases - Beliefs pane

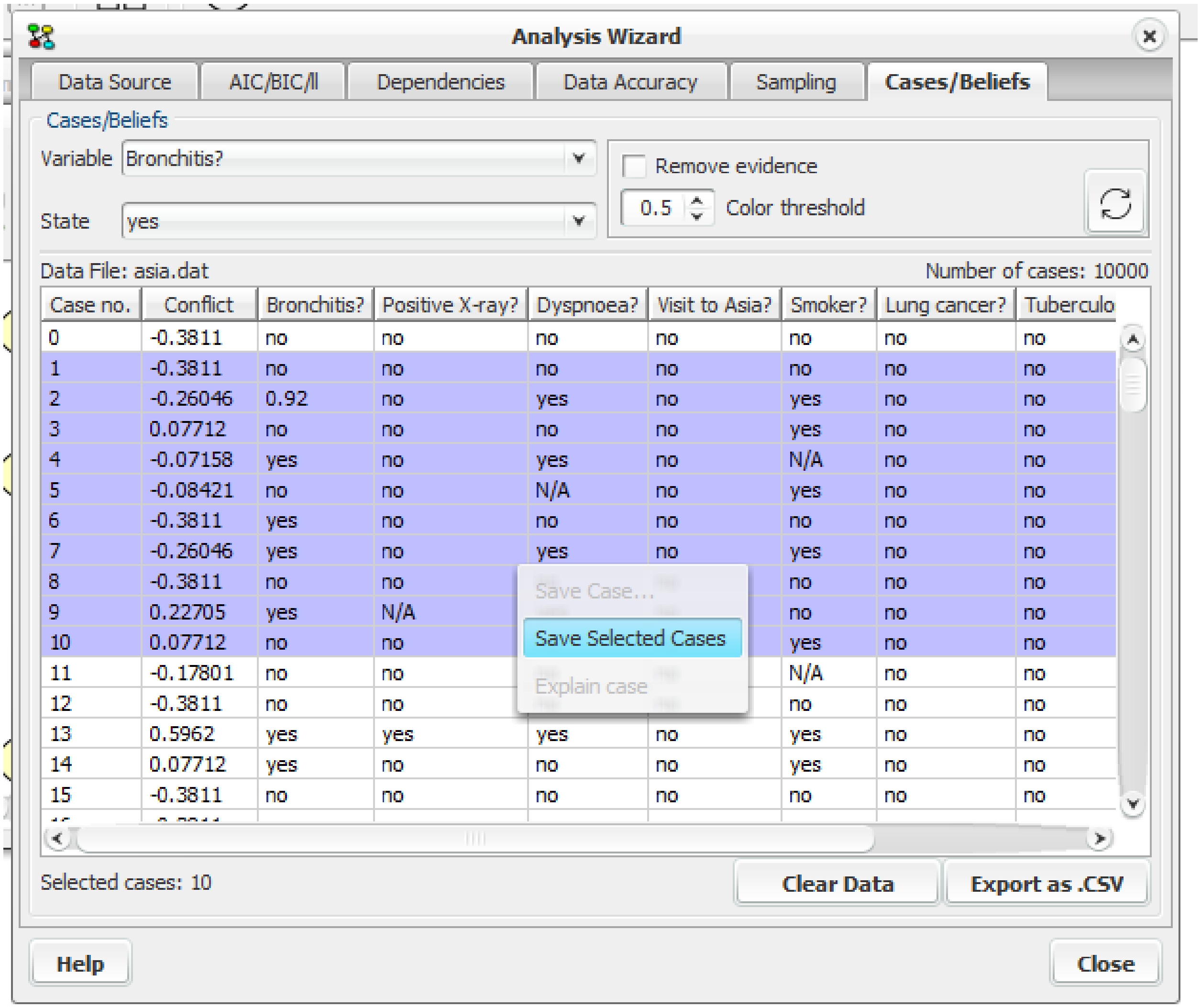
This pane consists of six elements, a drop down box at the top to select what node to analyse, a dropdown box to select one of the node’s states, a check box (Remove evidence), the color threshold spinner, a refresh button and a table showing the cases.

By selecting a node and one of its states, the case table displays the beliefs for that node in each case for which no evidence is entered. The information for the selected node will always be displayed in the first column after the *“Case no.”* column (second column).

By checking the *Remove evidence* check box and pressing the refresh button, beliefs are retracted and displayed for cases where evidence was entered in the node under analysis

The color *threshold spinner* offers the option of selecting the belief values to be colored (the values higher than the selected are colored). The tone of the color indicates how high the value is (the darker the color the higher the value is). The coloring ability together with the ability to sort the column makes it easier to notice the values desired.

To sort the column click, once on it's header for ascending order or twice for descending. It is possible to save one or more cases to a file so they can be used in another wizard (f.ex *Parameter Sensitivity* (page 335)). After selecting one or more cases right click on the mouse and select "Save Case" for saving one case, or "Save Selected Cases" for saving many cases. (see Figure 7).



Variable Bronchitis?

State yes

☐ Remove evidence

0.5 Color threshold

Data File: asia.dat

Number of cases: 10000

Case no.	Conflict	Bronchitis?	Positive X-ray?	Dyspnoea?	Visit to Asia?	Smoker?	Lung cancer?	Tuberculo
0	-0.3811	no	no	no	no	no	no	no
1	-0.3811	no	no	no	no	no	no	no
2	-0.26046	0.92	no	yes	no	yes	no	no
3	0.07712	no	no	no	no	yes	no	no
4	-0.07158	yes	no	yes	no	N/A	no	no
5	-0.08421	no	no	N/A	no	yes	no	no
6	-0.3811	yes	no	no	no	no	no	no
7	-0.26046	yes	no	yes	no	yes	no	no
8	-0.3811	no	no			no	no	no
9	0.22705	yes	N/A			no	no	no
10	0.07712	no	no			yes	no	no
11	-0.17801	no	no			N/A	no	no
12	-0.3811	no	no			no	no	no
13	0.5962	yes	yes	yes	no	yes	no	no
14	0.07712	yes	no	no	no	yes	no	no
15	-0.3811	no	no	no	no	no	no	no

Selected cases: 10

Figure 7: Cases - Beliefs pane "save selected cases"

4.5.2 Learning Wizard

To get assistance in the process of learning a Bayesian-network model from data, the Learning Wizard can be quite useful. The Learning Wizard can be invoked from the “Wizards” item of the menu bar of the *Main Window* (page 186).

The Learning Wizard read data from a plain text file, provides a *data matrix* (page 431) for editing and *preprocessing of the data* (page 408) in different ways (e.g., variable exclusion, value replacements, and discretization). The wizard also enables you to provide any structural or distributional domain knowledge that you might have, to resolve any structural uncertainties resulting from the *NPC learning algorithm* (page 246), to investigate the strengths of the dependences found in the data, and to provide prior knowledge about the conditional probability distributions.

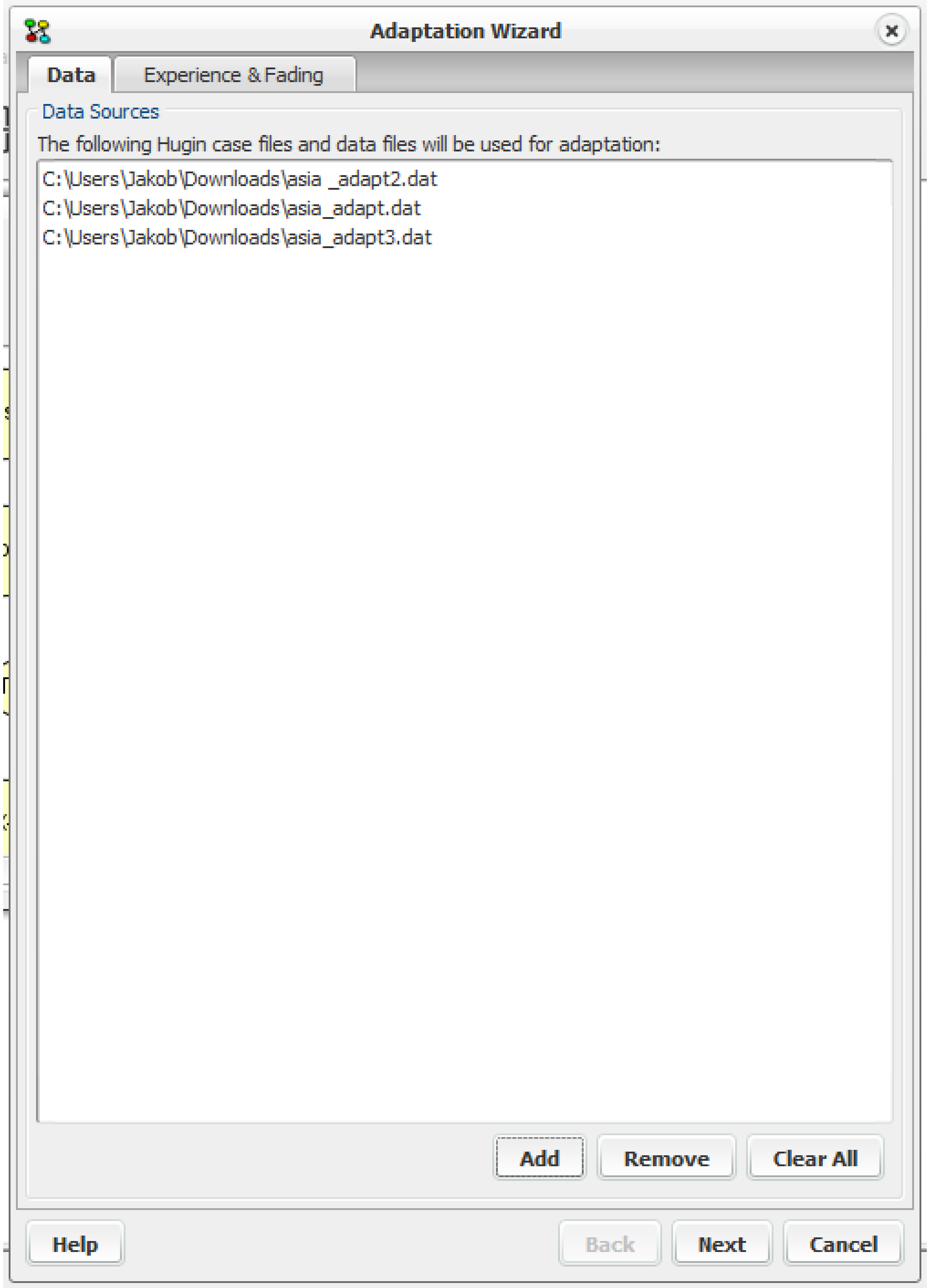
For more detailed information on the Learning Wizard, please consult the online help pages provided with the Learning Wizard.

4.5.3 Adaptation Wizard

The Adaptation Wizard is a tool for performing batch *adaptation* (page 244) on a network using any number of *case and data files* (page 85). The Adaptation Wizard can be activated by selecting “Adaptation Wizard” in the Wizards Menu. *Note that the wizard is only available in run mode, and for flat networks with discrete chance nodes.*

Step 1: Select data sources and configure tables

On the *Data* pane a set of case and/or data files can be added. The data sources will be used for adaptation in the order that they are added. The Data pane is displayed in figure 1.

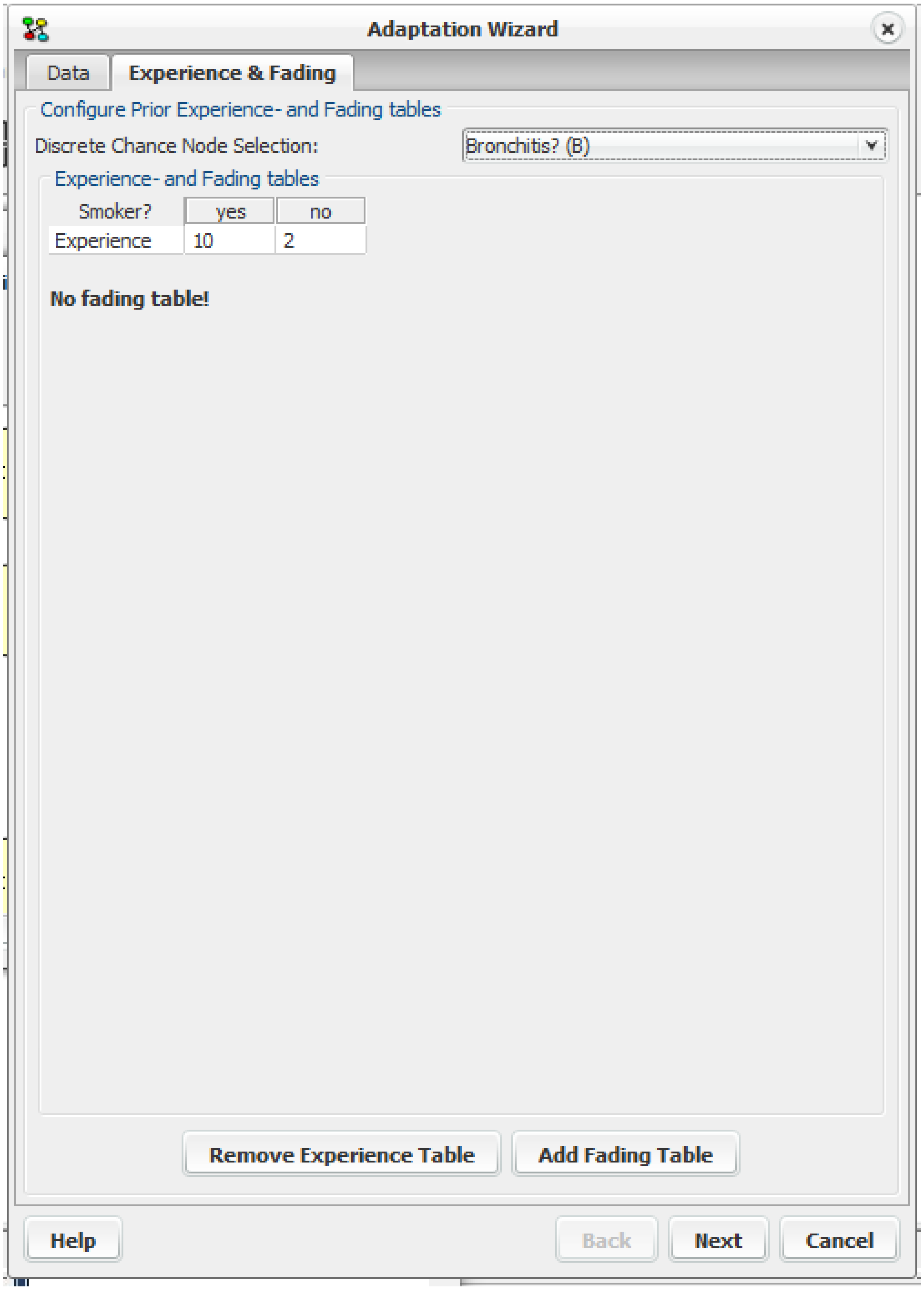


The following Hugin case files and data files will be used for adaptation:

C:\Users\Jakob\Downloads\asia _adapt2.dat
C:\Users\Jakob\Downloads\asia_adapt.dat
C:\Users\Jakob\Downloads\asia_adapt3.dat

Figure 1: Selecting data sources.

On the Experience and Fading pane, *experience* (page 231) and *fading tables* (page 233) may be configured for each node. The experience count and fading factors are configured in these tables. Adaptation will only be performed for conditional distributions of discrete chance nodes corresponding to parent configurations with a positive experience count.



Discrete Chance Node Selection: Bronchitis? (B)

Experience- and Fading tables

Smoker?	yes	no
Experience	10	2

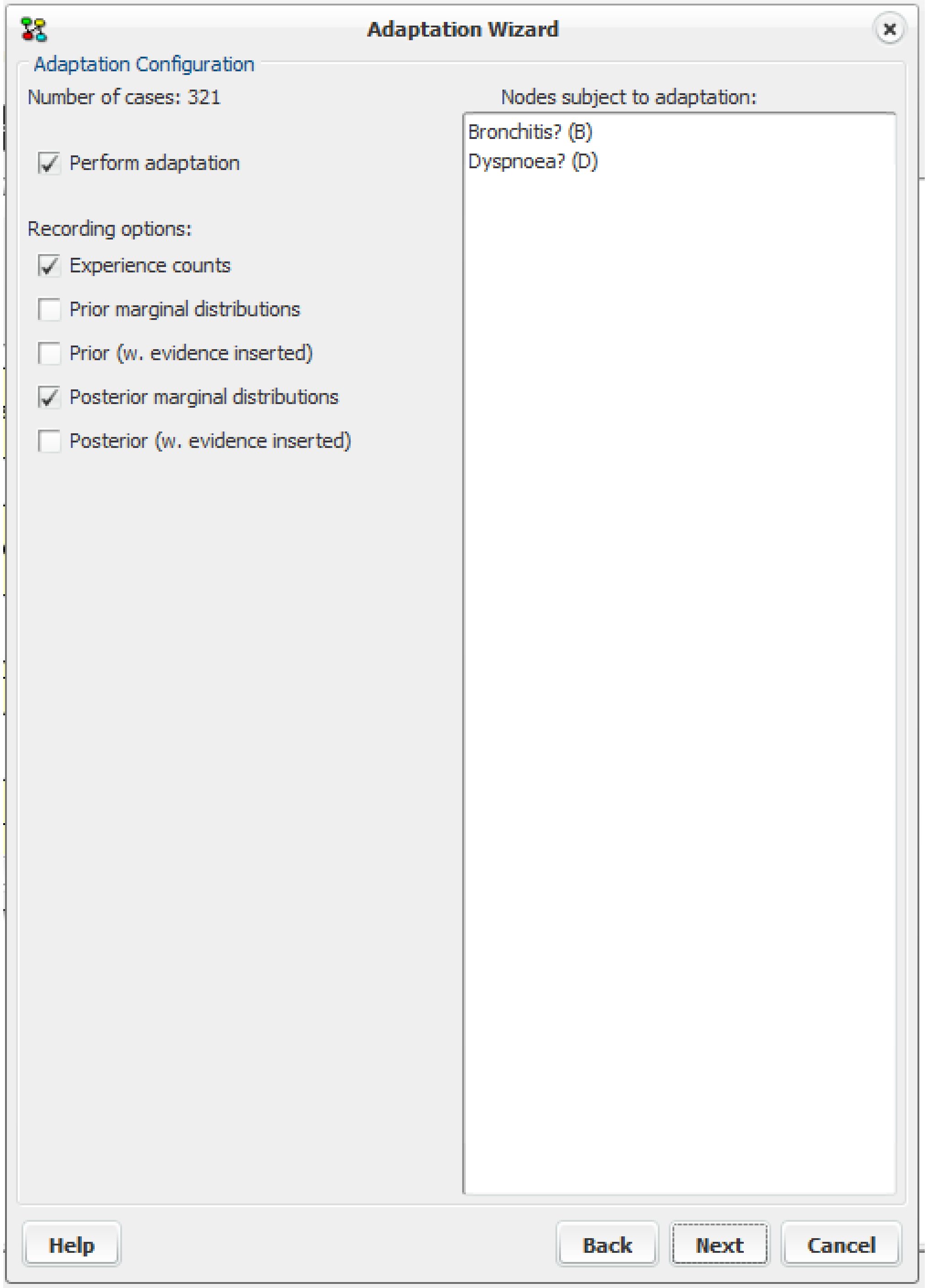
No fading table!

Figure 2: Configuring experience- and fading tables.

When the data sources, experience tables and fading tables have been configured, press the ‘next’ button.

Step 2: Review configuration

The dialog (figure 3) will let you review the configuration of the wizard. The recording options configures any information that the wizard should record and plot during adaptation. The dialog also displays a list with the nodes that will be subject to adaptation. When you are done press the ‘next’ button.



Adaptation Wizard

Adaptation Configuration

Number of cases: 321

☑ Perform adaptation

Recording options:

☑ Experience counts

□ Prior marginal distributions

□ Prior (w. evidence inserted)

☑ Posterior marginal distributions

□ Posterior (w. evidence inserted)

Nodes subject to adaptation:

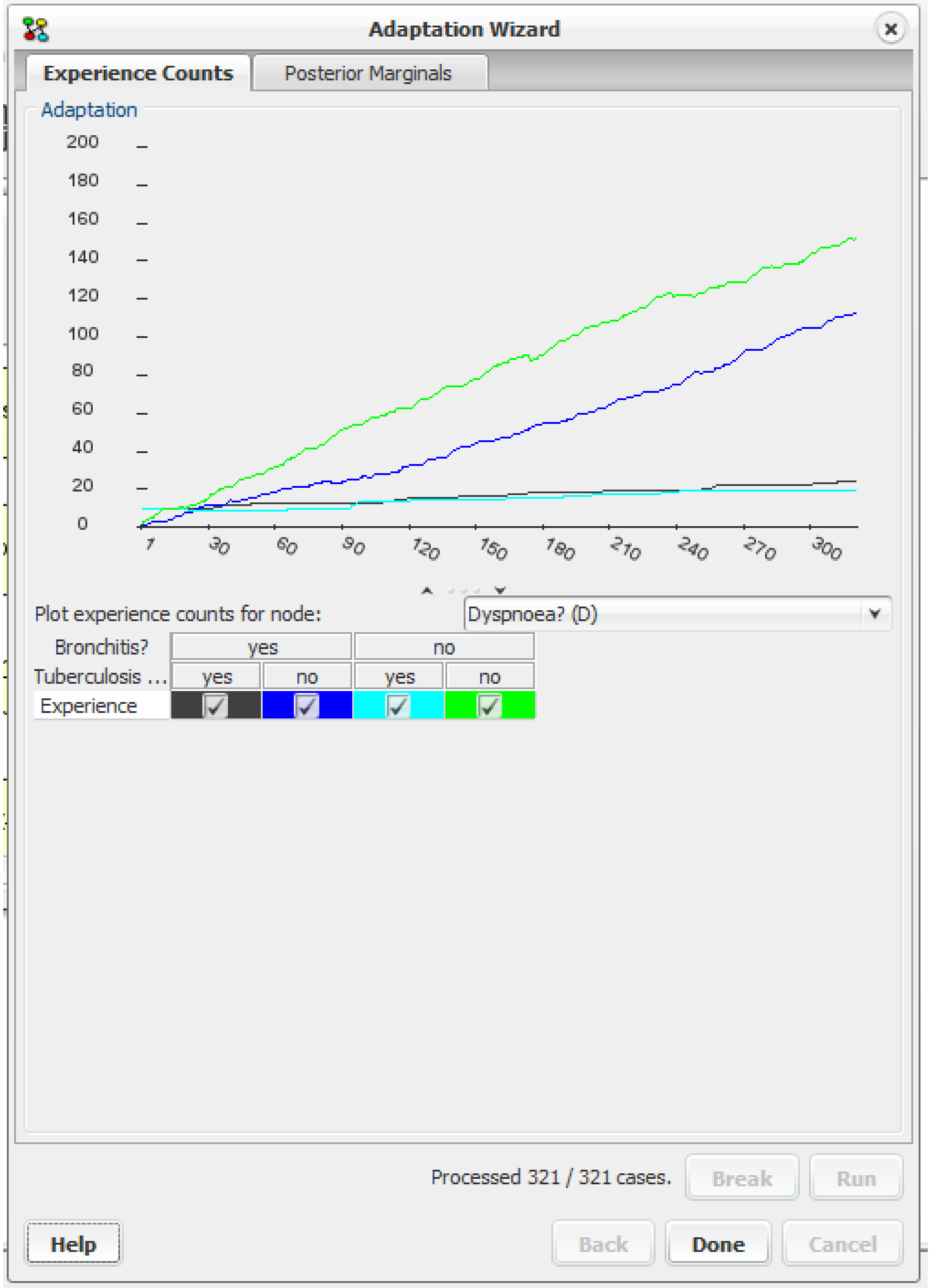
Bronchitis? (B)

Dyspnoea? (D)

Figure 3: Reviewing configuration, selection of information to be recorded.

Step 3: Perform adaptation

Figure 4 + 5 show the final dialog of the adaptation wizard. The batch adaptation process can be controlled using the buttons. Adaptation is started by pressing the button ‘Adapt’, and may be paused with ‘Break’. The dialog may have a number of panes (Experience Counts, figure 4; Marginals, figure 5) depending on the configuration specified for the recording options. In both panes the X-axis represent the cases, and the Y-axis represent experience count or posterior marginals.



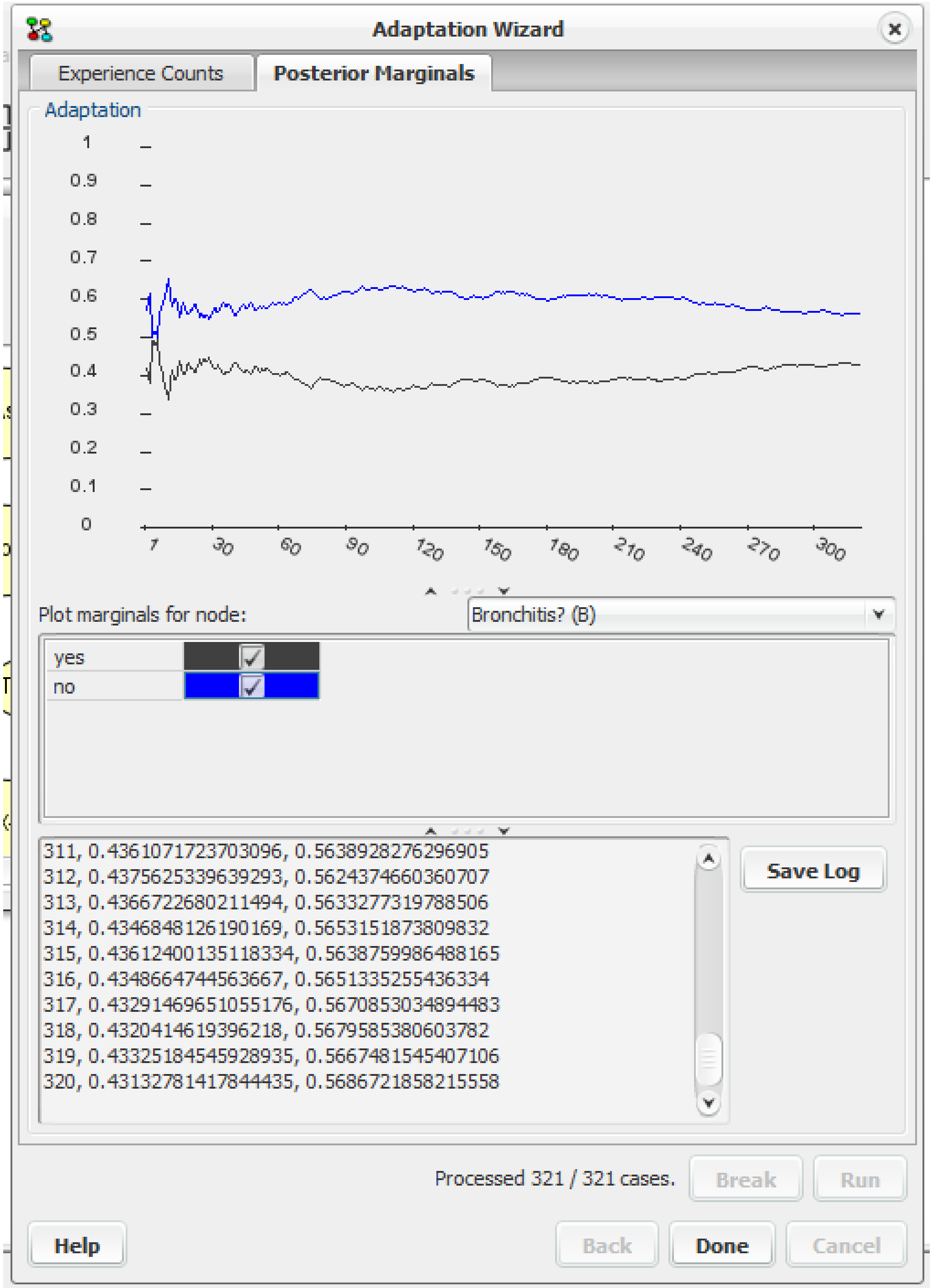
Plot experience counts for node: Dyspnoea? (D)

Bronchitis?	yes		no	
Tuberculosis ...	yes	no	yes	no
Experience	☑	☑	☑	☑

Processed 321 / 321 cases.

Figure 4: Experience counts.

From the Experience Counts pane (figure 4), it is possible to inspect the development of the experience counts for a node. The node can be selected from the dropdown list, and the exact configuration can be selected by clicking in the 'Experience' checkboxes. To make it easier to distinguish the different configurations, each is plotted in distinct colors.



Plot marginals for node: Bronchitis? (B)

yes ☑
no ☑

```
311, 0.4361071723703096, 0.5638928276296905
312, 0.4375625339639293, 0.5624374660360707
313, 0.4366722680211494, 0.5633277319788506
314, 0.4346848126190169, 0.5653151873809832
315, 0.43612400135118334, 0.5638759986488165
316, 0.4348664744563667, 0.5651335255436334
317, 0.43291469651055176, 0.5670853034894483
318, 0.4320414619396218, 0.5679585380603782
319, 0.43325184545928935, 0.5667481545407106
320, 0.43132781417844435, 0.5686721858215558
```

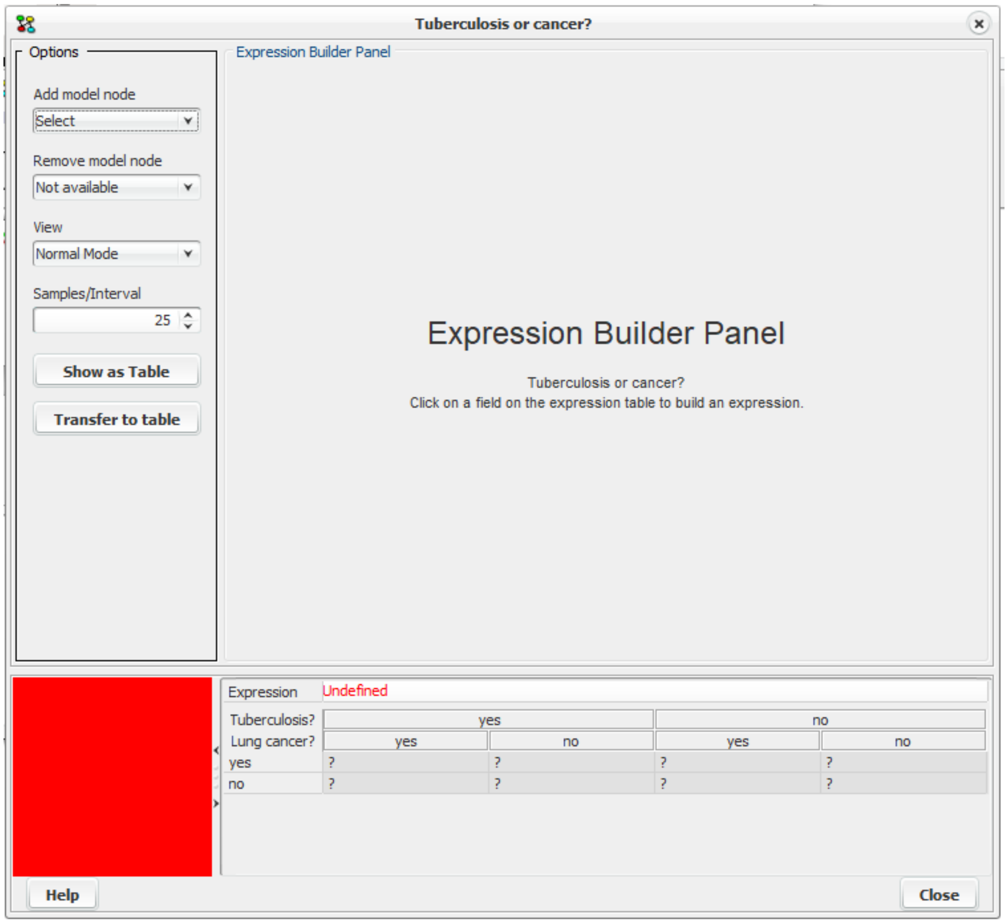
Processed 321 / 321 cases.

Figure 5: Posterior marginals.

From the Posterior Marginals pane (figure 5), the development of the posterior marginals of a node may be inspected. The posterior marginals for the exact states to plot can be selected by clicking in the checkboxes next to the states. The logged data can be saved to file by clicking on the *Save Log* button.

4.5.4 Expression Builder Wizard

The Expression Builder Wizard (Figure 1) gathers many functions related to adding *expressions* (page 150) on a *Node Table* (page 241). It consists of the *Options panel* (page 275), The *Expression Builder Panel* (page 277) and the Table Panel. The main purpose of this wizard is to make it easier to build an expression by assisting the user in selecting the correct type of expression the current node supports, selecting the correct type of parent nodes to be used in an expression and dividing the whole process into steps that allows him to concentrate on building one simple expression at a time without losing the overview of the complete expression.



Tuberculosis or cancer?

Options

Add model node: Select

Remove model node: Not available

View: Normal Mode

Samples/Interval: 25

Expression Builder Panel

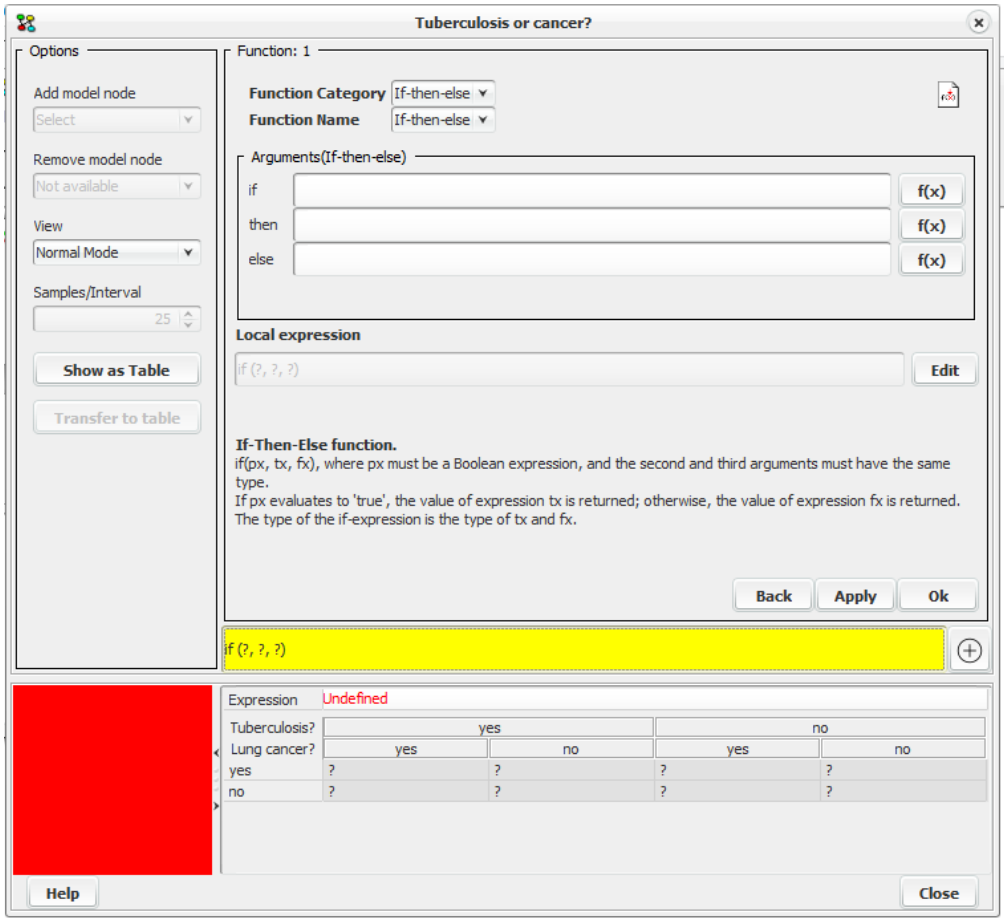
Tuberculosis or cancer?
Click on a field on the expression table to build an expression.

Expression: Undefined

Tuberculosis?	yes		no	
Lung cancer?	yes	no	yes	no
yes	?	?	?	?
no	?	?	?	?

Figure 1. Expression Builder Wizard

To build an expression, select one of the felts in the expression table where it shows “Undefined” (Figure 1). The Expression Builder Panel (Figure 1a) that is relevant for the type of the current node is now visible. Select the desired function category and then a function from the Function Name combo box to view the appropriate argument fields. Type expressions in the argument fields or press the “f(x)” button next to them to build an expression for the corresponding field. To insert a parent, right click and select from the list that appears.



Add model node: Select

Remove model node: Not available

View: Normal Mode

Samples/Interval: 25

Function Category: If-then-else

Function Name: If-then-else

Local expression: if (?, ?, ?)

If-Then-Else function.
if(px, tx, fx), where px must be a Boolean expression, and the second and third arguments must have the same type.
If px evaluates to 'true', the value of expression tx is returned; otherwise, the value of expression fx is returned.
The type of the if-expression is the type of tx and fx.

if (?, ?, ?)

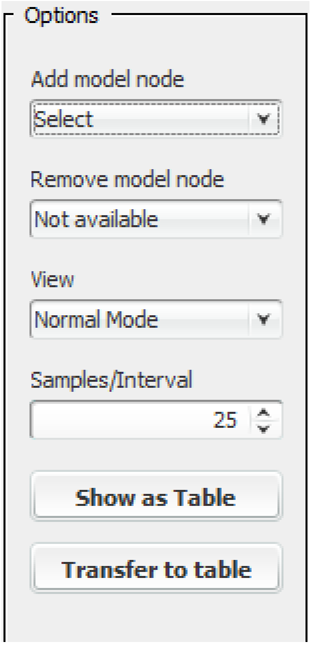
Expression	Undefined			
Tuberculosis?	yes		no	
Lung cancer?	yes	no	yes	no
yes	?	?	?	?
no	?	?	?	?

Figure 1a. The Expression Builder Panel is displayed for the node “Tuberculosis or Cancer”.

After building the expression press the “Apply” button and then the “Show as Table” button to confirm that the expression is built correctly. For more information about building an expression, see *Expression Builder Panel* (page 277).

Options Panel

The Option Panel (Figure 2) offers functionalities that are also available in the Node Table Frame menu.



Options

Add model node: Select

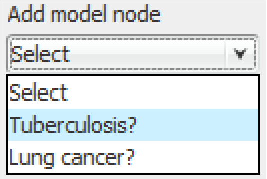
Remove model node: Not available

View: Normal Mode

Samples/Interval: 25

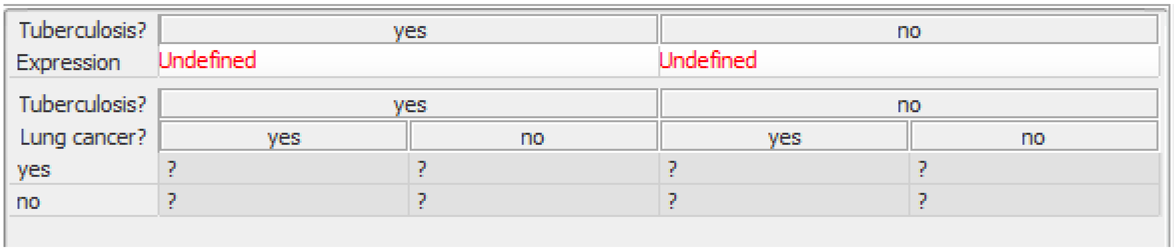
Figure 2. The Options Panel.

The Add/Remove model node combo boxes as implied, allows the user to add or remove parent nodes to the expression table .



Add model node: Select

Figure 2a. Selecting "Has tuberculosis".



Tuberculosis?	yes		no	
Expression	Undefined		Undefined	
Tuberculosis?	yes		no	
Lung cancer?	yes	no	yes	no
yes	?	?	?	?
no	?	?	?	?

Figure 2b. The table after adding a model node.

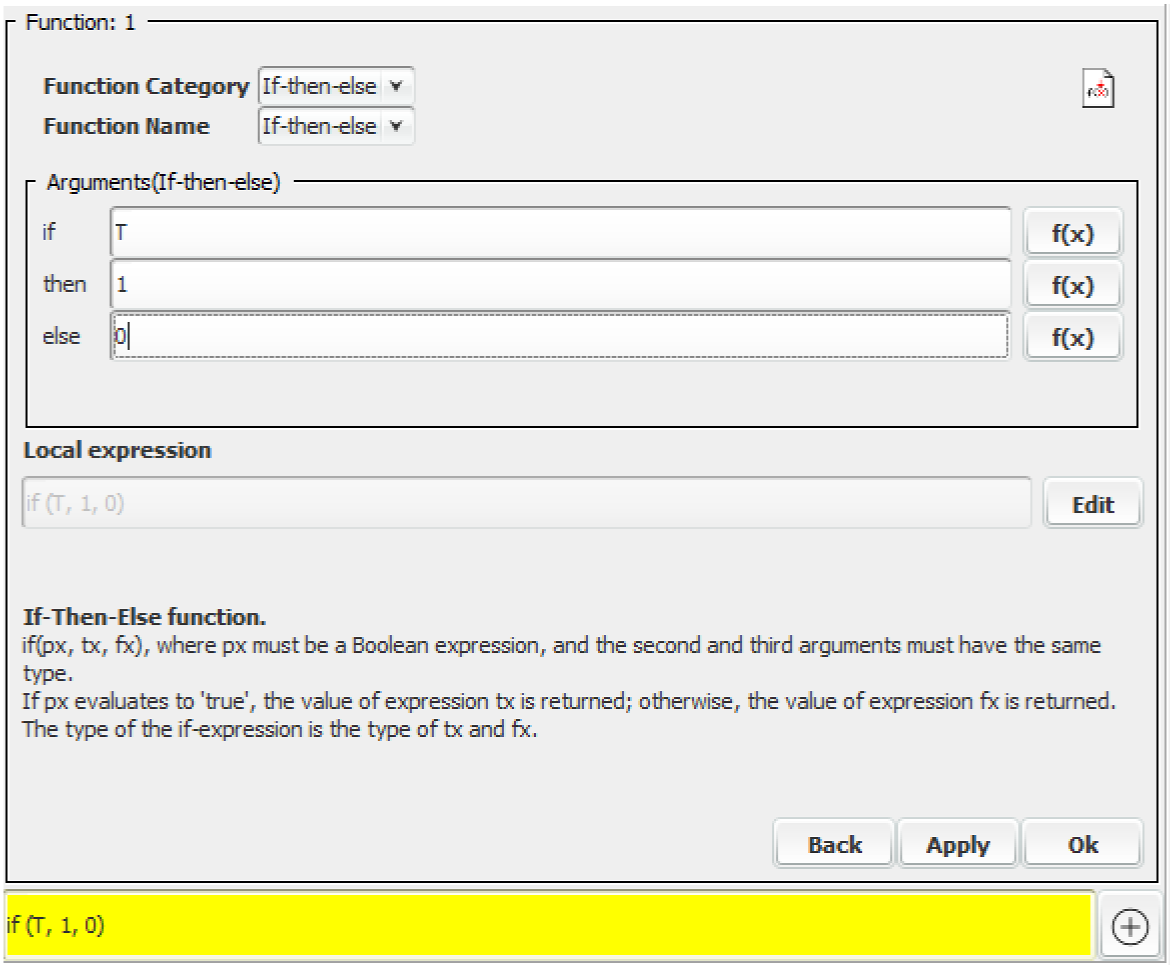
Figures 2a and 2b show how the Node table looks like after adding the parent node "Has tuberculosis" on the expression table model. This allows the specification of two different expressions, one for each state of the added parent node. To remove the node from the model, select it in the "Remove model node" combo box. The view combo box supports three different views of the values in the node table.

- Normal Mode: Only digits are displayed in every cpt index.
- Bar Mode: Both digits and a bar representing the value are displayed.

- Pure Bar Mode: Only a bar representing the value is displayed.

Expression Builder Panel

The Expression Builder Panel is where the actual building of the expressions takes place after selecting a cell in the expression table. It guides the user through the construction of Expressions, which are the cornerstones of the Table Generator, using a series of steps. The current step is displayed on the top left corner. In the example shown in Figure 3 the current step is 1 *(Function 1)*.



Function: 1

Function Category: If-then-else

Function Name: If-then-else

Arguments(If-then-else)

if: T

then: 1

else: 0

Local expression: if (T, 1, 0)

If-Then-Else function.
if(px, tx, fx), where px must be a Boolean expression, and the second and third arguments must have the same type.
If px evaluates to 'true', the value of expression tx is returned; otherwise, the value of expression fx is returned.
The type of the if-expression is the type of tx and fx.

if (T, 1, 0)

Figure 3: The Expression Builder Panel showing an *if - then - else* expression

The Expression Builder Panel assists the user in building expressions by providing a simple user interface for selecting a function or an operator and specifying its arguments.

The combo boxes at the top of the panel (*"Function category"* and *"Function Name"*), show the functions available (the actual set of functions depends on the type of the node associated with the current node table). Selecting a *"Function Category"* populates the *"Function Name"* combo box with the corresponding functions allowing the user to select. Figure 3a shows the function categories associated with a *"Numbered"* node. The functions are organized in a number of categories to make it faster to find a particular function.

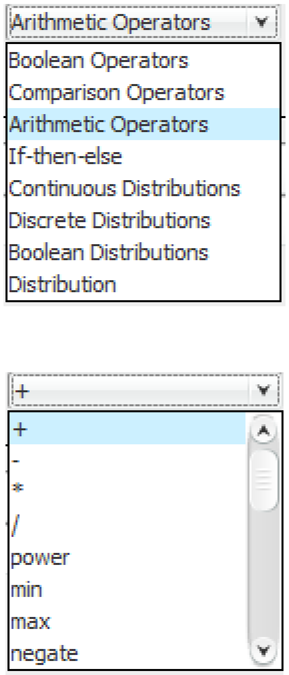


Figure 3a. Selecting Arithmetic Operators in the *“Function Category”* combo box will allow the selection of arithmetic operators in the *“Function Name”* combo box.

The arguments panel (Figure 3b) is where the arguments of the expression can be specified. Each argument is itself an expression and can therefore be built in the same way as the current expression. Pressing the *“f(x)”* button next to the argument text field, will start “step 2” allowing the user to build an expression for that argument.

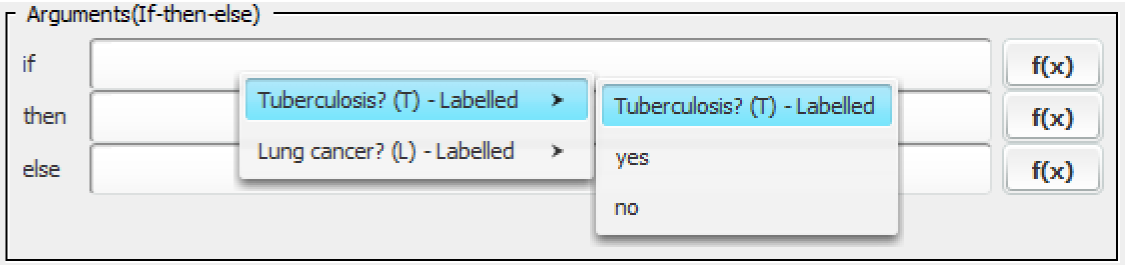
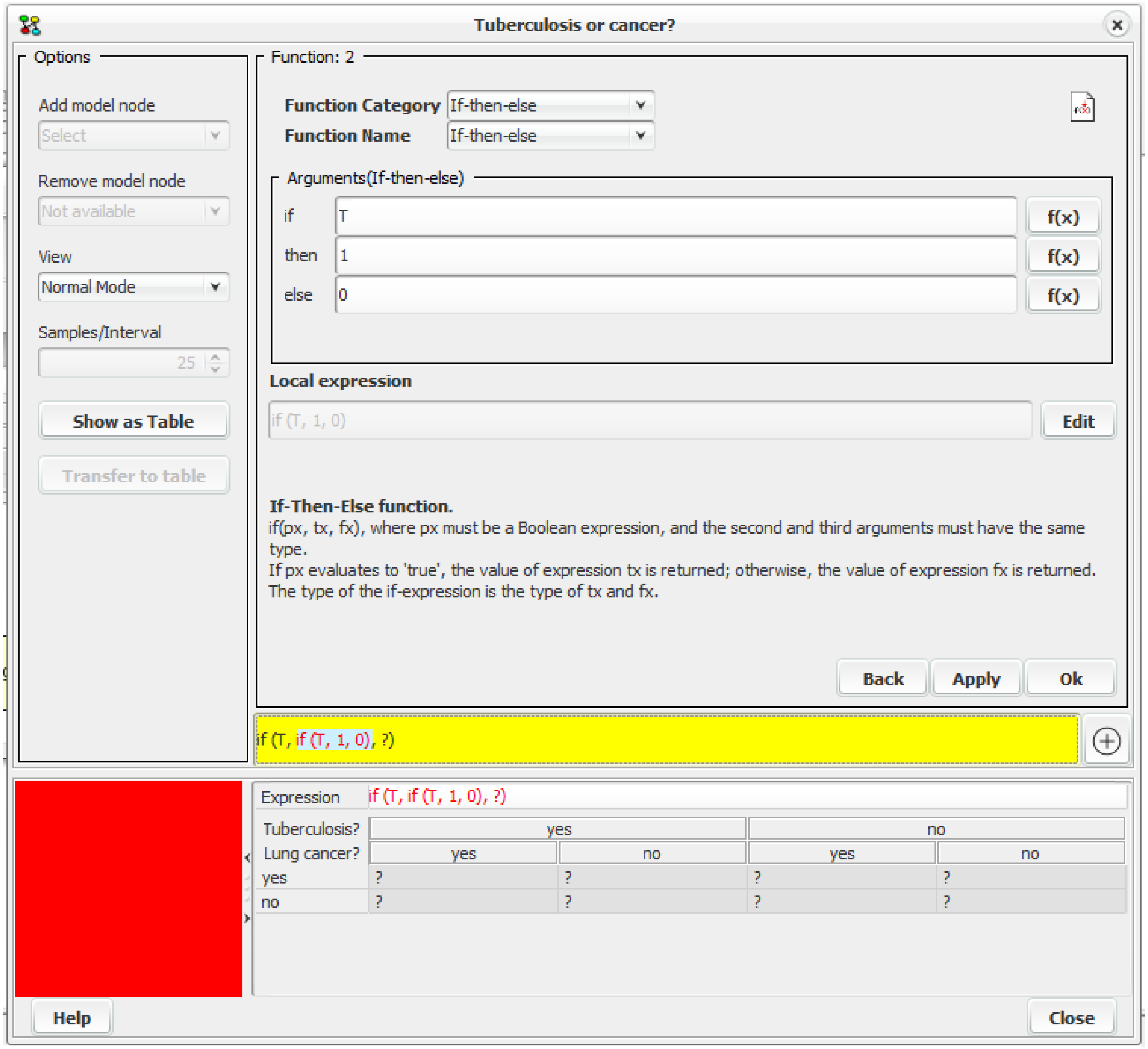

Arguments(If-then-else)
if
then
else
Tuberculosis? (T) - Labelled
Lung cancer? (L) - Labelled
Tuberculosis? (T) - Labelled
yes
no
f(x)
f(x)
f(x)

Figure 3b. The arguments panel. Selecting the parent node “Has tuberculosis” of type “Labelled”, to insert in the if argument field.

The use of a parent *node* (page 209) in an expression is possible by using its unique name. Figure 3b shows how to insert a parent node in an expression. By right clicking on an argument field a popup appears showing a list of the parent nodes. Pointing on a node displays its *type* (page 216), to help the user decide if it can be used in the current expression, because not all types can be used on all expressions. A short description of the currently editing expression is displayed under the local expression field (Figure 3) and among other things it describes what type of nodes can be used. When selecting a parent node from the list it is inserted into the argument field. Only discrete chance nodes and decision nodes can appear in expressions.

When editing in the argument fields, the changes made are also shown in the “Local expression” field and the colored field at the bottom. These fields display the expressions in their proper form replacing arguments that are not yet specified with the “?” mark.



Tuberculosis or cancer?

Add model node: Select

Remove model node: Not available

View: Normal Mode

Samples/Interval: 25

Function: 2

Function Category If-then-else

Function Name If-then-else

Arguments(If-then-else)

if: T

then: 1

else: 0

Local expression

if (T, 1, 0)

If-Then-Else function.
if(px, tx, fx), where px must be a Boolean expression, and the second and third arguments must have the same type.
If px evaluates to 'true', the value of expression tx is returned; otherwise, the value of expression fx is returned.
The type of the if-expression is the type of tx and fx.

if (T, if (T, 1, 0), ?)

Expression: if (T, if (T, 1, 0), ?)

Tuberculosis?	yes		no	
Lung cancer?	yes	no	yes	no
yes	?	?	?	?
no	?	?	?	?

Figure 3c. The local expression represents the current editing expression in the arguments panel and the colored field at the bottom shows the global expression.

The “Current expression” represents the sub expression that is being edited in the current step and the colored field displays the whole expression with the currently editing expression highlighted to help the user keep a perspective of the process. The “plus” button next to the colored field, initializes the “expression viewer” that allows the user to view a very long expression that might not be possible to display in the standard text field.

Figure 3c shows that we are currently in step two (Function 2. Upper left corner) and we are building an *if - then - else* expression with arguments T, 1 and 0. The *“Current expression”* shows the written form of the expression (if (T, 1, 0)). The colored field shows that the current expression is used as the second argument in an *if - then - else* expression, with the node “T” as the first argument and the last argument not specified yet.

The expression builder can be switched to **edit mode** by pressing the “Edit” button allowing the user to manually type in the “Current expression” field, just like it can be done in the *node table frame* (page 63). The insertion of a parent node is done in the same way as described above.

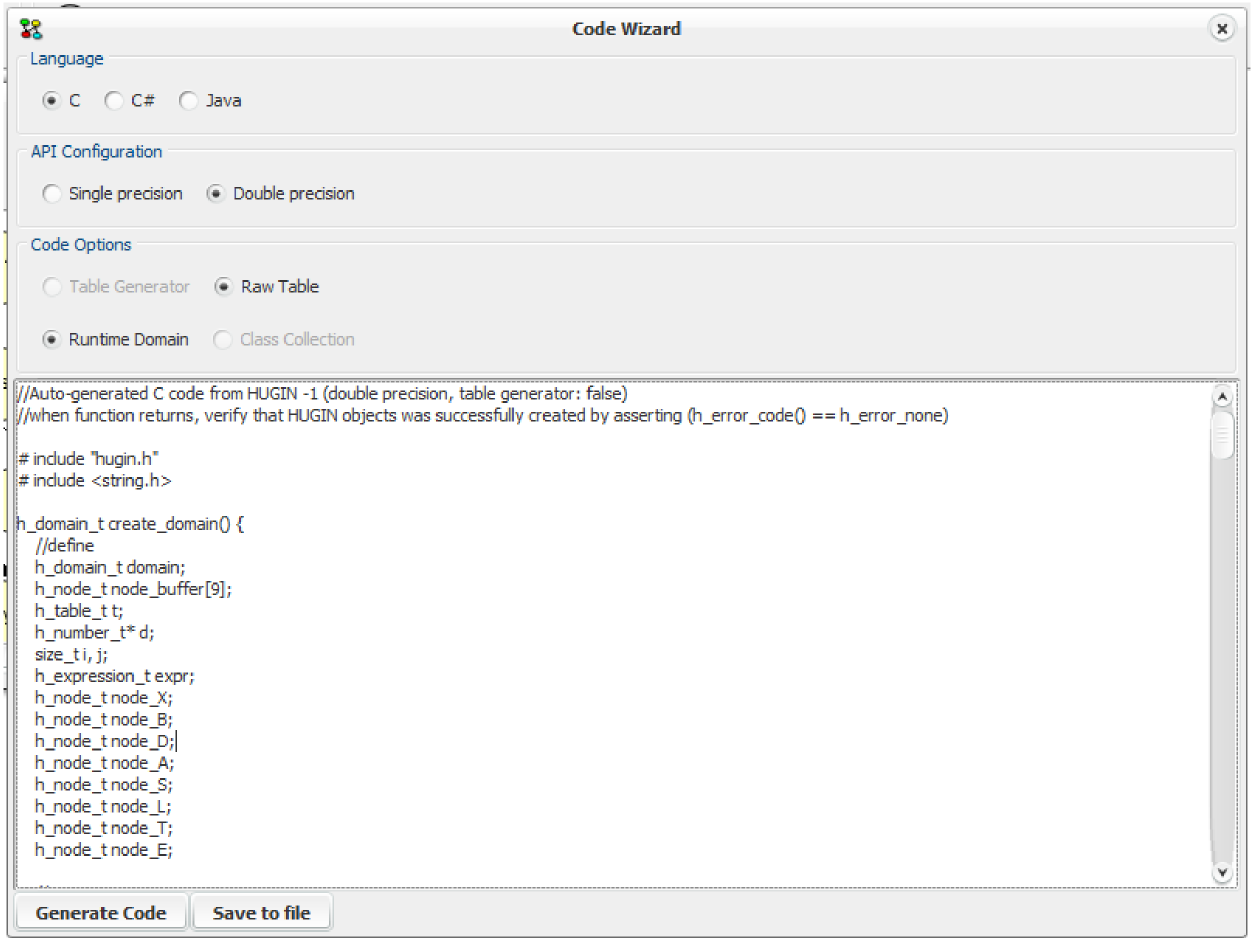
It is possible to return to build mode again by pressing again the same button (now displaying “Build”). If the expression typed in edit mode *(local expression field)* can be parsed correctly, the appropriate arguments panel will be displayed and the argument fields will be populated.

When pressing the *"Back"* button the Expression Builder returns to the previous step if there is one or to its initial state, ignoring eventual changes.

The *"Apply"* button inserts the expression in the expression table without parcing for errors. The *"Ok"* button closes the current arguments panel, and returns the expression created to the target arguments field of the previous step or to the expression table if currently in step 1. **After inserting the expression in the expression table press the "Show As Table" button (Options Panel) to parse for errors.**

4.5.5 Code Wizard

The Code Wizard is activated by selecting from the menu **Wizards -> Code Wizard**.



Language

● C ○ C# ○ Java

API Configuration

○ Single precision ● Double precision

Code Options

○ Table Generator ● Raw Table

● Runtime Domain ○ Class Collection

```
//Auto-generated C code from HUGIN -1 (double precision, table generator: false)
//when function returns, verify that HUGIN objects was successfully created by asserting (h_error_code() == h_error_none)

# include "hugin.h"
# include <string.h>

h_domain_t create_domain() {
   //define
   h_domain_t domain;
   h_node_t node_buffer[9];
   h_table_t t;
   h_number_t* d;
   size_t i, j;
   h_expression_t expr;
   h_node_t node_X;
   h_node_t node_B;
   h_node_t node_D;
   h_node_t node_A;
   h_node_t node_S;
   h_node_t node_L;
   h_node_t node_T;
   h_node_t node_E;
```

Figure 1: The Code Wizard

The Code Wizard may be usefull for programmers using any of the HUGIN APIs for application development. In some scenarios loading a HUGIN model from a file is impractical and embedding the model in code may the only solution. The Wizard generates a code snippet for constructing a runtime model.

The code produced has the following properties:

- Constructs a DOMAIN (For OOBNs this means the DOMAIN resulting from unfolding the OOBN model)

- DOMAIN is triangulated using the triangulation available at code generation time
- No error checking or error handling

4.5.6 EM Learning Wizard

To get assistance in the process of learning the CPT's of a Bayesian network from data, the EM Learning Wizard can be quite useful. The EM Learning Wizard can be invoked from the "Wizards" item of the menu bar of the *Main Window* (page 186).

Please note that the EM Learning Wizard is intended in situations where a network structure is already available. If a network structure is not available, one should consider using the complete *Learning Wizard* (page 267), which provides a superset of the functionalities provided by the EM Learning Wizard.

The EM Learning Wizard read data from a plain text file, provides a *data matrix* (page 267) for editing and *preprocessing of the data* (page 408) in different ways (e.g., variable exclusion, value replacements, and discretization). The wizard also enables you to provide prior knowledge about the conditional probability distributions and densities.

For more detailed information on the EM Learning Wizard, please consult the online help pages provided with the EM Learning Wizard.

4.5.7 Hidden Node Analyzer

The Hidden Node Analyzer is a wizard for determining the best number of states of a (hidden) discrete chance node in a Bayesian network from a dataset. The Hidden Node Analyzer has three steps:

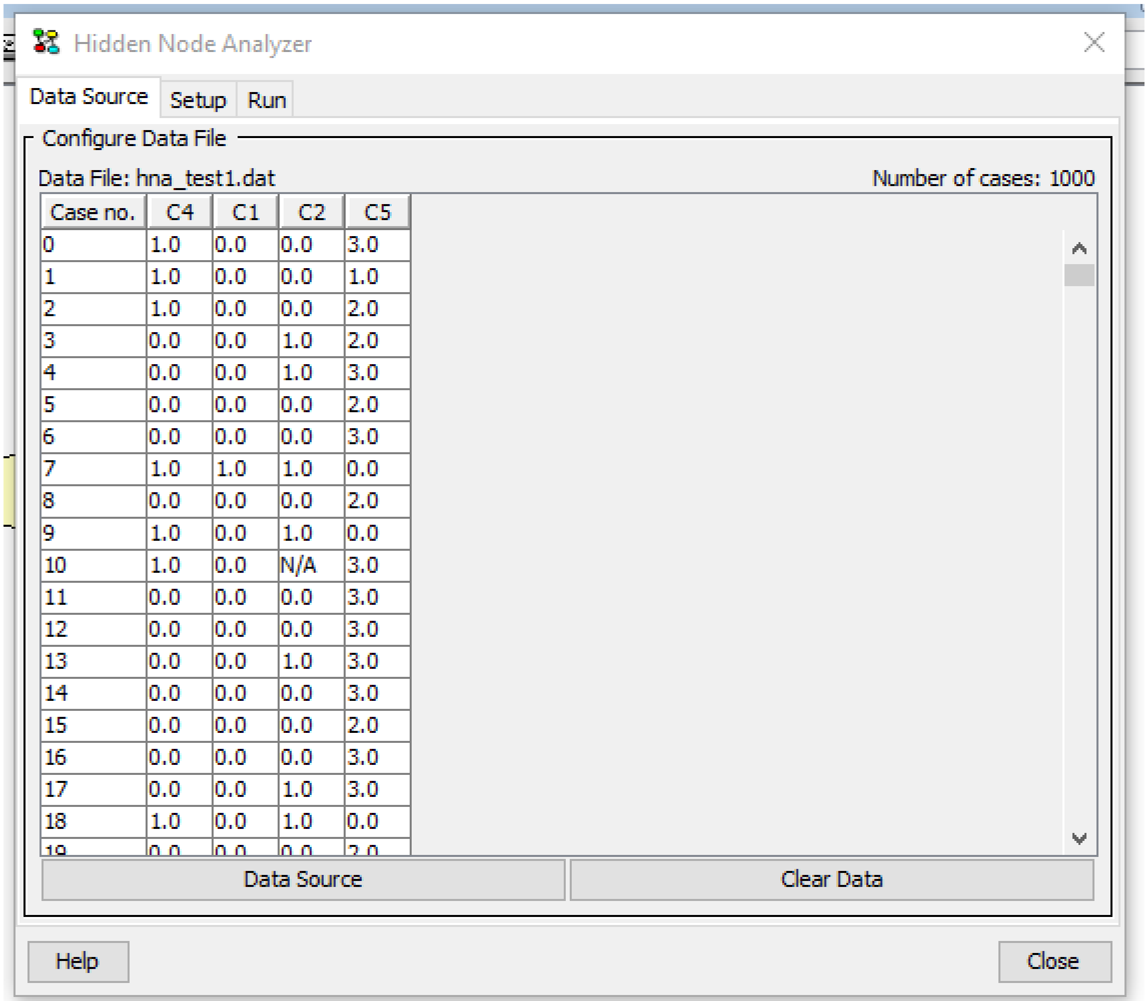
- loading a data file
- selecting the hidden node and setting the parameters for identification of the number of states of the hidden node
- the analysis

Each step is described in the following sections.

Load Data File

To perform any analysis the wizard will first need a set of cases. Cases can be imported by loading a HUGIN data file or connecting to a database service and import. Click on button "Data Source" to import data.

After importing a set of cases, the Data Source pane will display a table with the imported cases, see Figure 1. To discard all imported cases click on the button "Clear Data".



Data File: hna_test1.dat

Number of cases: 1000

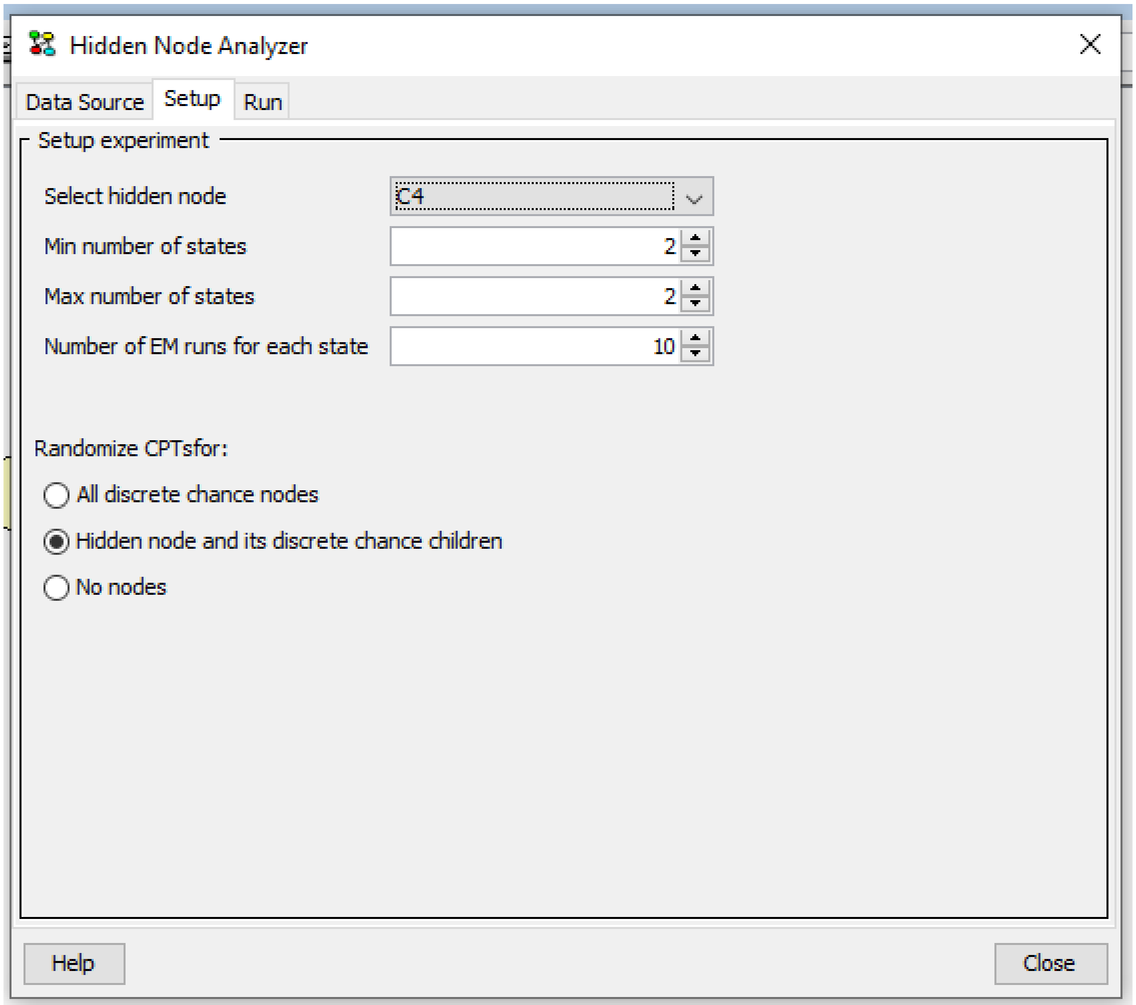
Case no.	C4	C1	C2	C5
0	1.0	0.0	0.0	3.0
1	1.0	0.0	0.0	1.0
2	1.0	0.0	0.0	2.0
3	0.0	0.0	1.0	2.0
4	0.0	0.0	1.0	3.0
5	0.0	0.0	0.0	2.0
6	0.0	0.0	0.0	3.0
7	1.0	1.0	1.0	0.0
8	0.0	0.0	0.0	2.0
9	1.0	0.0	1.0	0.0
10	1.0	0.0	N/A	3.0
11	0.0	0.0	0.0	3.0
12	0.0	0.0	0.0	3.0
13	0.0	0.0	1.0	3.0
14	0.0	0.0	0.0	3.0
15	0.0	0.0	0.0	2.0
16	0.0	0.0	0.0	3.0
17	0.0	0.0	1.0	3.0
18	1.0	0.0	1.0	0.0
19	0.0	0.0	0.0	2.0

Figure 1: Data Source pane. A set of cases has been imported.

Setting Parameters

The Hidden Node Analyzer can be used to identify the number of states of a hidden node. A node is hidden if no case in the dataset contains an observation on the node.

To identify the number of states of a hidden node, the user should specify the hidden node, the minimum number of states, the maximum number of states, and whether the conditional probability table (CPT) of the hidden node should be randomized (default option) as illustrated in Figure 2.



Setup experiment

Select hidden node: C4

Min number of states: 2

Max number of states: 2

Number of EM runs for each state: 10

Randomize CPTsfor:

○ All discrete chance nodes

● Hidden node and its discrete chance children

○ No nodes

Figure 2: Setup pane.

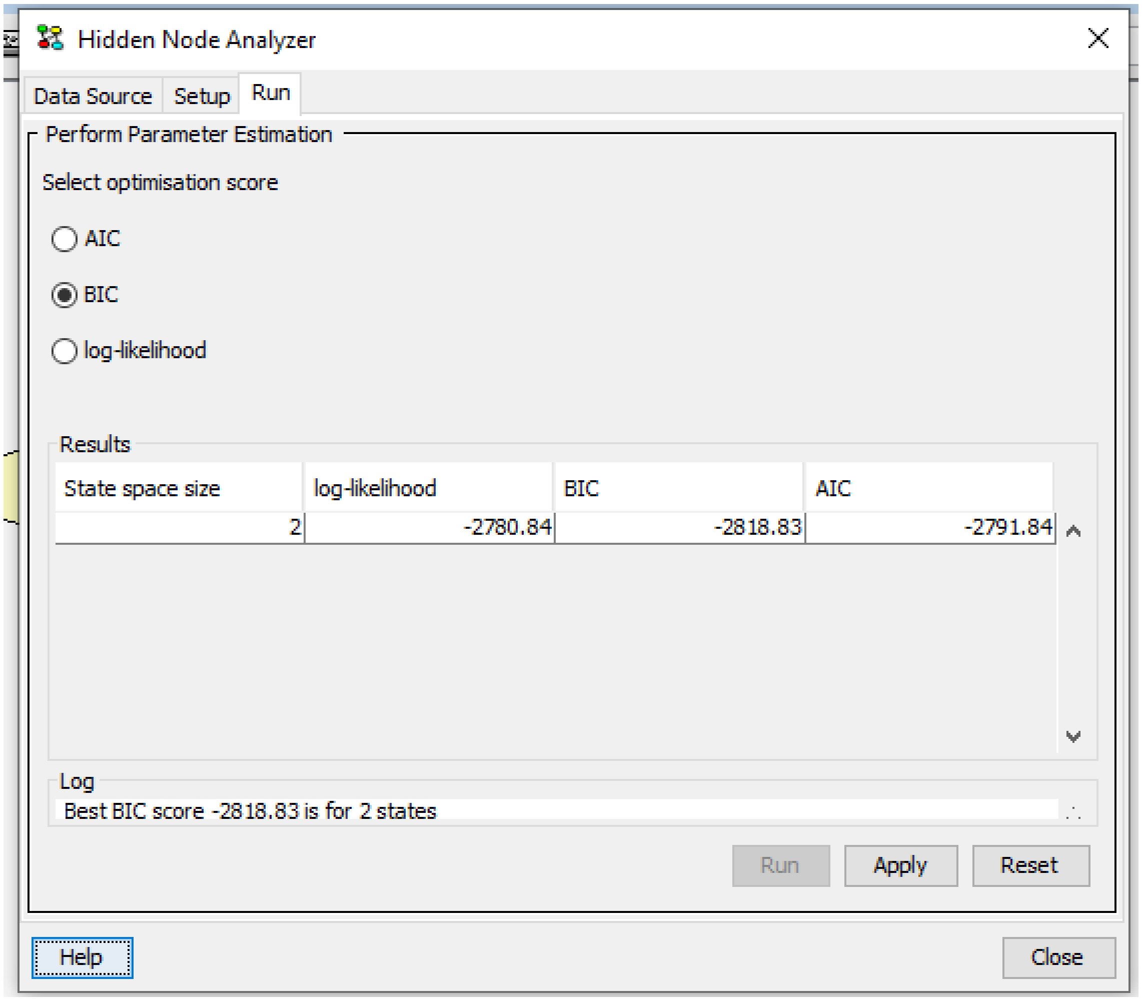
The Hidden Node Analyzer supports discrete chance nodes only and only if there are atleast one non-boolean node.

Analysis

The Hidden Node Analyzer iterates over the possible state space sizes of the hidden from minimum to maximum as selected by the user. In each iteration an EM learning operation is performed from the initial network, i.e., the state of the network when the wizard was opened. The conditional probability table of the hidden node is randomized, if the user has selected this option (default). For each iteration the AIC, BIC and log-likelihood are displayed to the user.

Notice that to perform the analysis, the network must be compiled and one EM learning operation is performed for each state considered. This may result in long running times.

Figure 3 shows the result of running the Hidden Node Analyzer on an example network and dataset.



Perform Parameter Estimation

Select optimisation score

○ AIC

● BIC

○ log-likelihood

Results

State space size	log-likelihood	BIC	AIC
2	-2780.84	-2818.83	-2791.84

Log

```
Best BIC score -2818.83 is for 2 states
```

Figure 3: Run pane.

For each possible state space size of the hidden node, the Hidden Node Analyzer reports the value of each optimisation score. For the selected optimisation score, the highest scoring state size space is shown in the last line. This is a help for the user to identify the number of states producing the highest score.

The user can choose to *apply* the settings associated with the highest scoring state space size. This will update the underlying model including setting the number of states of the hidden node and updating the conditional probability tables et cetera of the nodes in the network from the result of the EM learning iteration producing the highest score. Only conditional probability tables for nodes with experience tables are updated.

4.5.8 Feature Selection Analyzer

Feature selection is an important task in the development of Bayesian networks for, for instance, classification. The HUGIN Graphical User Interface has support for feature selection through the Feature Selection Analyzer. The Feature Selection Analyzer has three steps:

- Open a network in the network pane
- loading a data file for feature selection
- selection of the target variable (e.g., the classification variable) feature selection

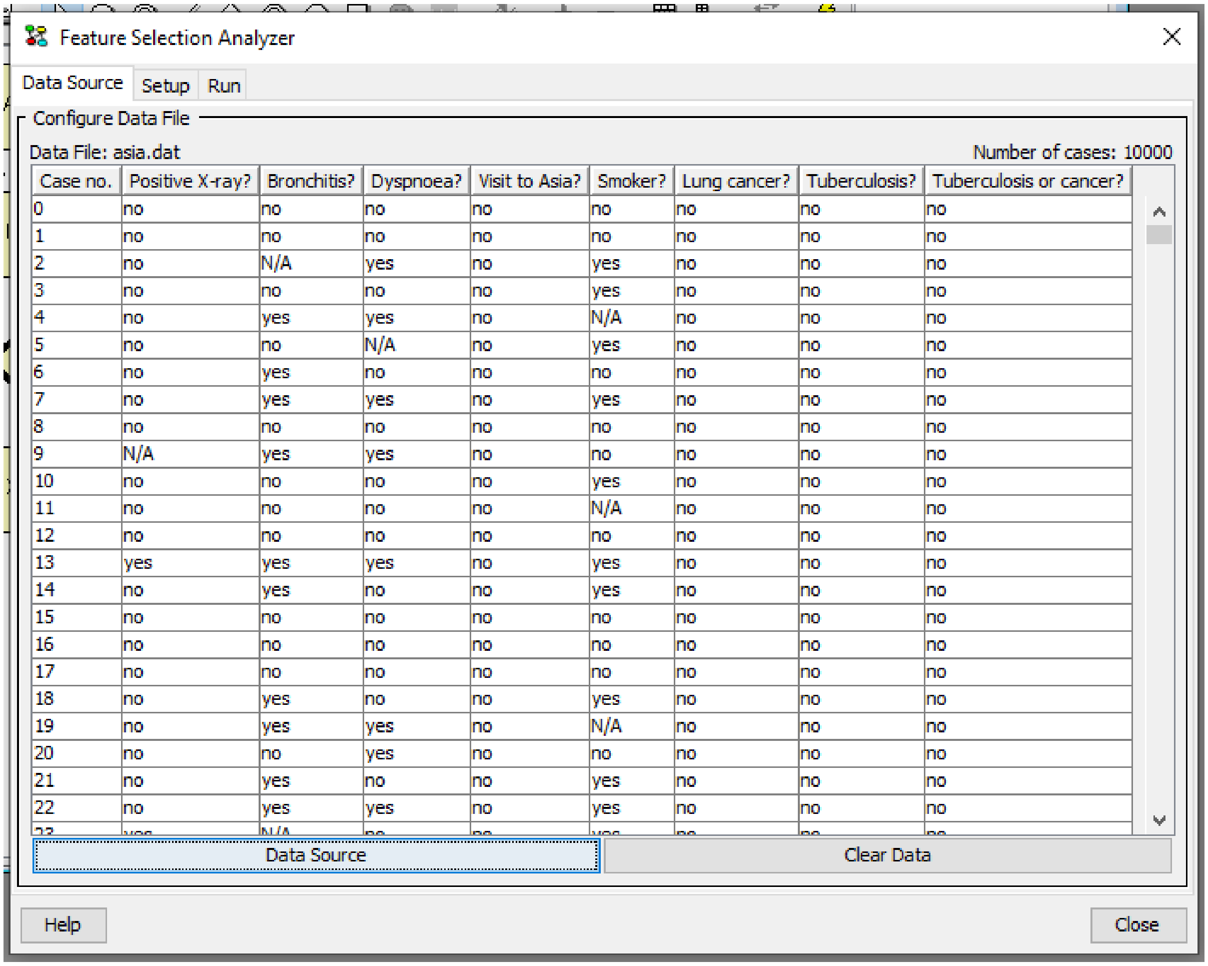
- the feature selection

Each step is described in the following sections.

Load Data File

To perform any analysis the wizard will first need a set of cases. Cases can be imported by loading a HUGIN data file or connecting to a database service and import. Click on button “Data Source” to import data. The column names in the data file must correspond to the names of the nodes in the model. Columns not matching the nodes in the model will be filtered out.

After importing a set of cases, the Data Source pane will display a table with the imported cases, see Figure 1. To discard all imported cases click on the button “Clear Data”.



Data File: asia.dat

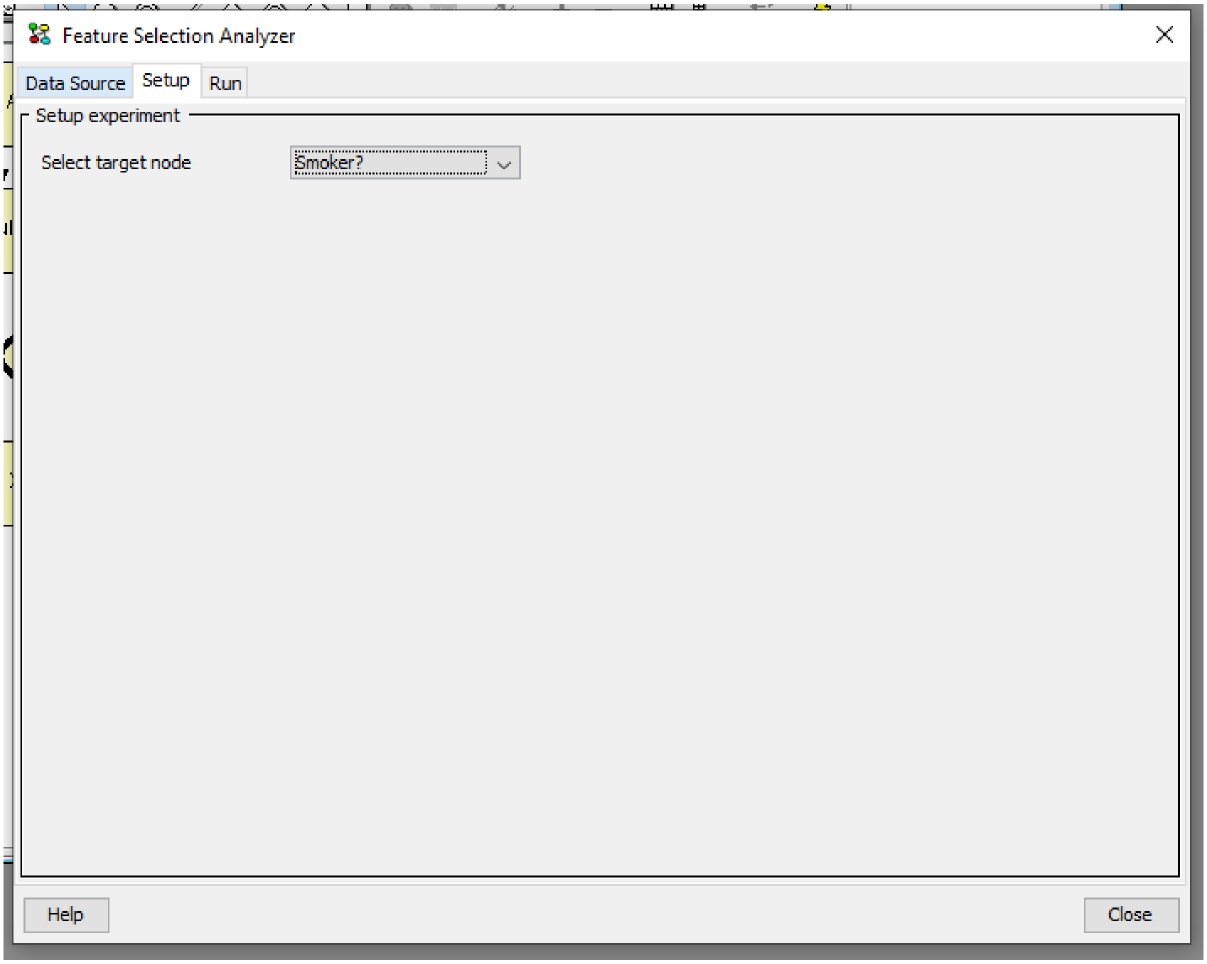
Number of cases: 10000

Case no.	Positive X-ray?	Bronchitis?	Dyspnoea?	Visit to Asia?	Smoker?	Lung cancer?	Tuberculosis?	Tuberculosis or cancer?
0	no	no	no	no	no	no	no	no
1	no	no	no	no	no	no	no	no
2	no	N/A	yes	no	yes	no	no	no
3	no	no	no	no	yes	no	no	no
4	no	yes	yes	no	N/A	no	no	no
5	no	no	N/A	no	yes	no	no	no
6	no	yes	no	no	no	no	no	no
7	no	yes	yes	no	yes	no	no	no
8	no	no	no	no	no	no	no	no
9	N/A	yes	yes	no	no	no	no	no
10	no	no	no	no	yes	no	no	no
11	no	no	no	no	N/A	no	no	no
12	no	no	no	no	no	no	no	no
13	yes	yes	yes	no	yes	no	no	no
14	no	yes	no	no	yes	no	no	no
15	no	no	no	no	no	no	no	no
16	no	no	no	no	no	no	no	no
17	no	no	no	no	no	no	no	no
18	no	yes	no	no	yes	no	no	no
19	no	yes	yes	no	N/A	no	no	no
20	no	no	yes	no	no	no	no	no
21	no	yes	no	no	yes	no	no	no
22	no	yes	yes	no	yes	no	no	no
23	yes	N/A	no	no	yes	no	no	no

Figure 1: Data Source pane. A set of cases has been imported.

Select Target Node

Feature selection is performed relative to a *target* node. The target node should be selected in this pane.



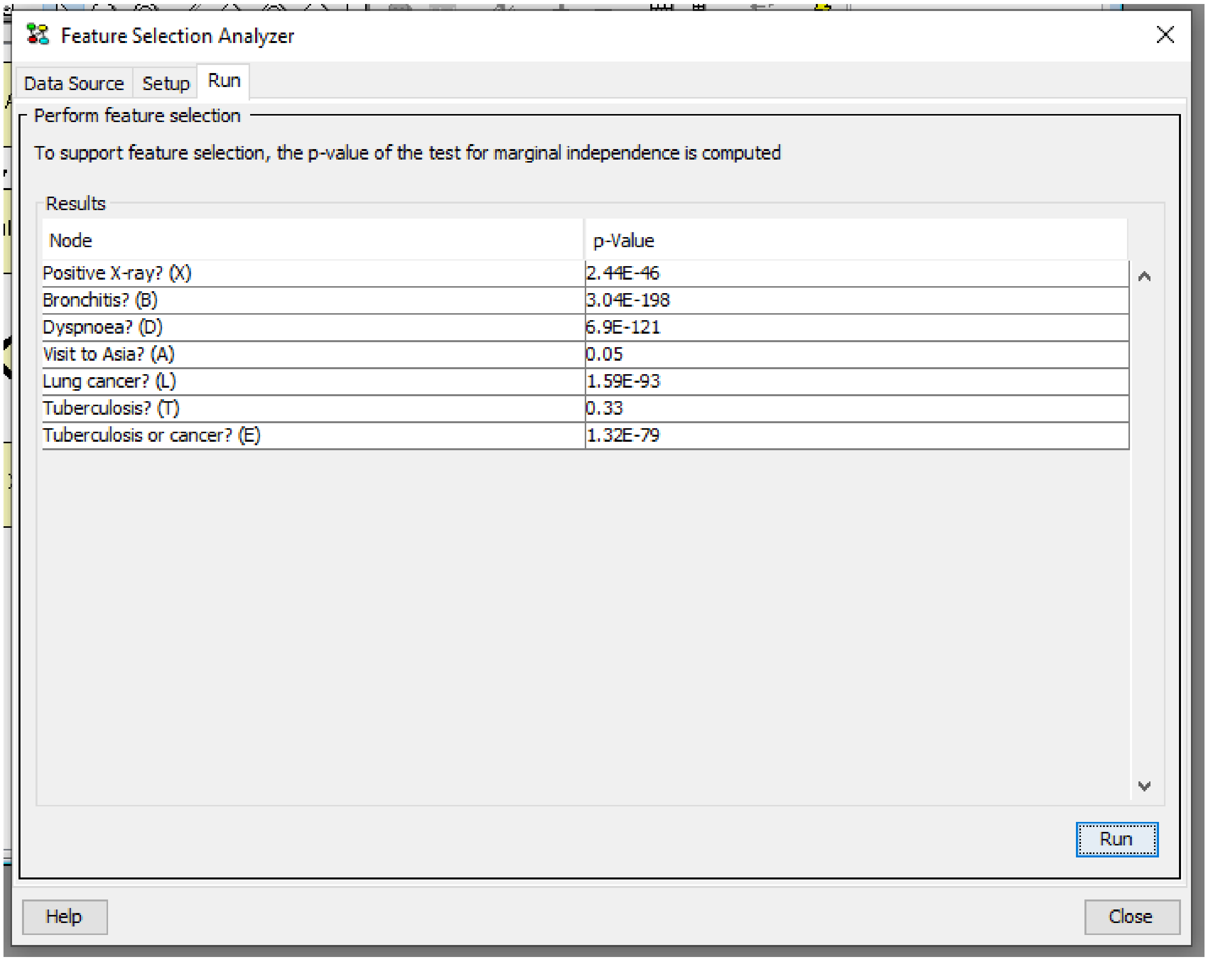
Select target node: Smoker?

Figure 2: Setup Pane.

Feature selection is supported for discrete chance nodes only, i.e., the target node and the feature nodes should all be discrete chance nodes.

Feature Selection

Feature selection is based on computing the p-value of the test for marginal independence between the selected target node and each possible feature node. Figure 3 shows the result of feature selection on an example network and dataset.



To support feature selection, the p-value of the test for marginal independence is computed

Results

Node	p-Value
Positive X-ray? (X)	2.44E-46
Bronchitis? (B)	3.04E-198
Dyspnoea? (D)	6.9E-121
Visit to Asia? (A)	0.05
Lung cancer? (L)	1.59E-93
Tuberculosis? (T)	0.33
Tuberculosis or cancer? (E)	1.32E-79

Figure 3: Run Pane.

To support feature selection, the p-value of the test for marginal independence is computed. The p-value is the tail probability under the independence assumption. The higher the value the more likely the nodes are to be independent. (p is the probability of obtaining a Q value as large or larger than the Q value computed from the data under the null-hypothesis - which is the independence assumption. The Q value is a measure of the distance between the joint distribution of the feature and the target and the product of the marginal distributions of the feature and the target).

Thus, for a small value (for instance, less than a significance level alpha), the null-hypothesis that the nodes are not related is rejected and, hence, we assume the feature to be relevant for the target node.

If the user selects a set of nodes in the list, then these nodes will remain selected when the wizard is closed. This is useful for selecting the highest scoring features.

4.5.9 Parameter Sensitivity Wizard

The Parameter Sensitivity Wizard is a tool that provides support for analysis based on individual parameters. The current version allows the user to constrain parameters (*Constraints panel* (page 134)) and to perform *"Parameter Sensitivity Analysis"* (page 335) and *Two Way Parameter Sensitivity Analysis* (page 340).

The Parameter Sensitivity Wizard can be activated through the menu bar (Wizards - Parameter Sensitivity Wizard).

Note that the wizard is only available in run mode, and for flat networks with discrete nodes.

4.6 Methods Of Analysis

4.6.1 Methods of Analysis

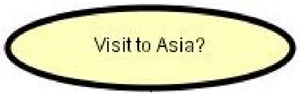
The HUGIN Graphical User interface supports a number of different methods of analysis on a model. These methods include d-separation analysis, data conflict analysis, sensitivity to evidence analysis, and value of information analysis. The Analysis Menu in figure 1 can be found under analysis in the *Network Menu* (page 185).

- *d-separation* (page 288)
- *Show requisite parents for decision* (page 291)
- *Show requisite ancestors for decision* (page 291)
- *Data Conflict* (page 292)
- *Evidence Sensitivity* (page 299)
- *Utility Sensitivity* (page 309)
- *Value of Information* (page 312)
- *CVDT Function* (page 320)
- *Convolution* (page 321)
- *XOI Loss Model* (page 323)
- *Joint Configurations* (page 326)
- *Correlation Analysis* (page 329)
- *Distance Analysis* (page 332)
- *Parameter Sensitivity* (page 335)

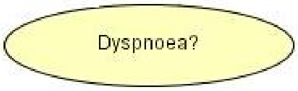
4.6.2 D-Separation (Operation)

Determining the *d-relations* (page 289) in a network can be done in Hugin when in run-mode. This is done using the operations available in the D-Separation submenu of the *Network menu* (page 185).

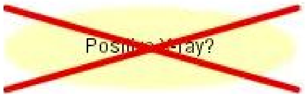
When a d-separation operation has been performed, the visual representation of the network changes to show the result of the operation. A node can be represented in five different ways depending on its role in the operation:



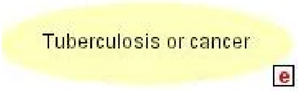
The source node. That is, the node whose relation to all the target nodes will be determined.



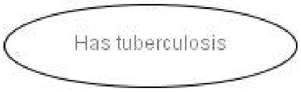
A connected target node. That is, the source node is d-connected to this node.



A separated target node. That is, the source node is d-separated from this node.



An instantiated node. The d-relation to the source node is not determined.



An "anonymous" node. That is, a node which is none of the above.

- **Show D-Separation**

This operation will display all nodes d-seperated from the currently selected node.

- **Show Markov Blanket**

This operation will display the *Markov Blanket* (page 122). for the currently selected node.

4.6.3 D-Separation (Theory)

The rules of d-separation, due to *Pearl (1988)* (page 541), can be used to determine the dependences and independences among the variables of a Bayesian network. This is very useful for e.g. (structural) *model verification* (page 348) and explanation. It is possible to perform *d-separation* (page 288) in HUGIN when the network is in run-mode.

To use the rules of d-separation, one must be familiar with the three fundamental kinds of connections

- X → Y → Z (*serial connection*): Information may be transmitted through the connection unless the state of Y is known. Example: If we observe the rain falling (Y), any knowledge that there is a dark cloud (X) is irrelevant to any hypothesis (or belief) about wet lawn (Z). On the other hand, if we don't known whether it rains or not, observing a dark cloud will increase our belief about rain, which in turn will increase our belief about wet lawn.
- X ← Y → Z (*diverging connection*): Information may be transmitted through the connection unless the state of Y is known. Example: If we observe the rain falling (Y) and then that the lawn is wet (X), the added knowledge that the lawn is wet (X) will tell us nothing more about the type of weather report to expect from the radio (Z) than the information gained from observing the rain alone. On the other hand, if we don't known whether it rains or not, a rain report in the radio will increase our belief that it is raining, which in turn will increase our belief about wet lawn.
- X → Y ← Z (*converging connection*): Information may be transmitted through the connection only if information about the state of Y or one of its descendants is available. Example: If we know the lawn is wet (Y) and that the sprinkler is on (X), then this will effect our belief about whether it has been raining or not (Z), as the wet lawn leads us to believe that this was because of the sprinkler rather than the rain. On the other hand, if we have no knowledge about the state of the lawn, then observing that it rains will not affect our belief about whether the sprinkler has been on or not.

Two variables X and Z are *d-separated* if for all paths between X and Z there is an intermediate variable Y such that either

- the connection is serial or diverging and Y is instantiated (i.e., its value is known), or
- the connection is converging and neither Y nor any of its descendants have received evidence.

If X and Z are not d-separated, they are *d-connected*. Note that dependence and independence depends on what you know (and do not know). That is, the evidence available plays a significant role when determining the dependence and independence relations.

Let us consider the DAG in Figure 1. Using the above rules, we find e.g. that

- C and L are d-separated, as each path between C and L includes at least one converging connection.
- C and L are d-connected given K (i.e., the value of K is known), as there are now several paths between C and L where the converging connections contain an instantiated variable (e.g., C → G → K ← H → L) or a descendant (K) of the node in the converging connection has been instantiated (e.g., C ← A → D ← B → E → H → L).

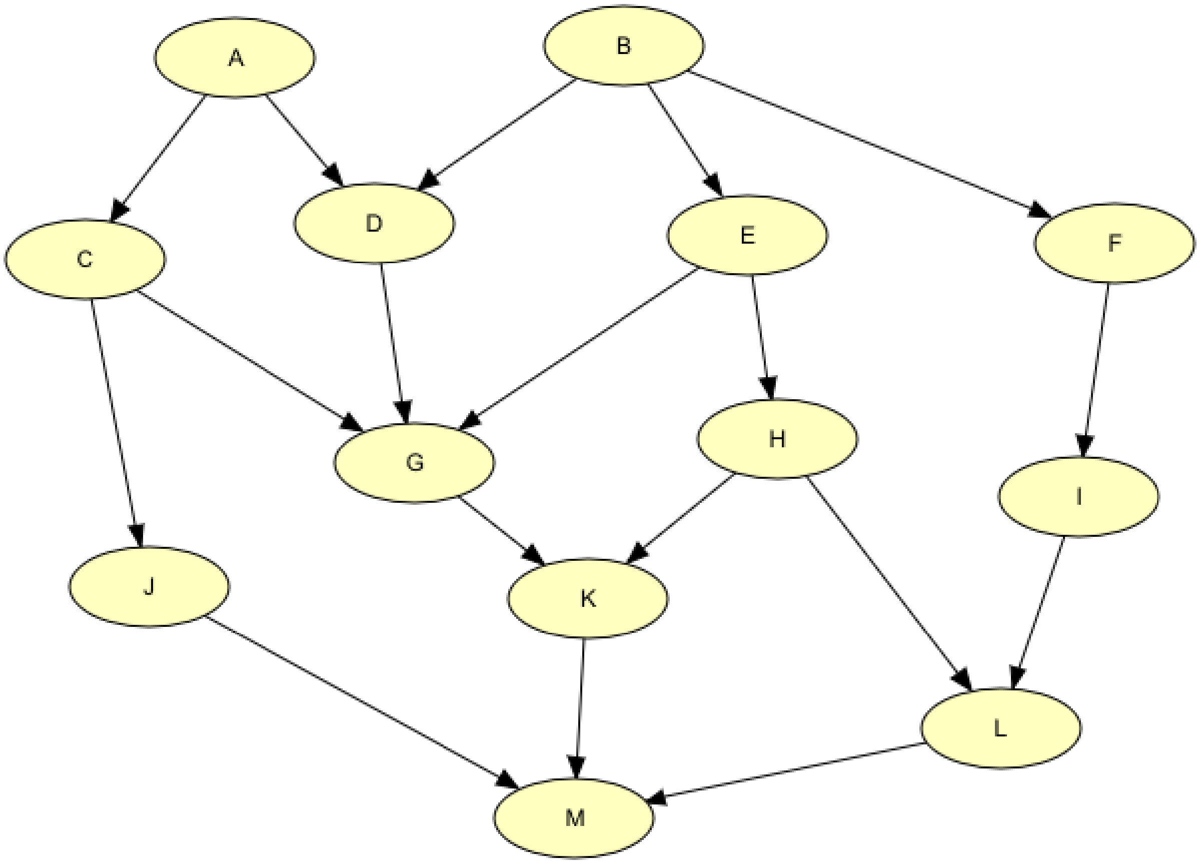


Figure 1: A sample network.

As mentioned in *Introduction to Bayesian Networks* (page 19), an alternative to using the d-separation rules is to use an equivalent criterion due to *Lauritzen et al. (1990):* (page 541): Let A, B, and C be disjoint sets of variables. Then

- identify the smallest sub-graph that contains A B C and their ancestors,
- add undirected links between nodes having a common child, and
- drop directions on all directed links.

Now, if every path from a variable in A to a variable in B contains a variable in C, then A is conditionally independent of B given C.

4.6.4 Requisite Parents

Not all available observations matter when a decision must be made. Intuitively, a parent of a decision node is said to be requisite if the value of the parent may affect the optimal choice of the decision.

The performance of inference in a LIMID can be improved, if the network is simplified by removing the nonrequisite parents of all decision nodes.

Lauritzen and Nilsson (2001) (page 541) present an algorithm for removing the nonrequisite parents of the decision nodes in a LIMID. The result of this algorithm is a LIMID, where no parent of a decision node is nonrequisite. This network is known as the minimal reduction of the LIMID.

Our definition of requisiteness is slightly different than the one given by Lauritzen and Nilsson: **We define a parent of a decision node to be requisite if and only if it is also a parent of the decision node in the minimal reduction of the LIMID.**

Inspect the requisite parents of a decision node

Select a decision node. Click **Network -> Analysis -> Show requisite parents for decision.**

Requisite parents will be marked with a green V (see Figure 1) and non-requisite parents will be marked with a red X (see Figure 2).

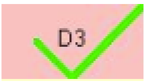


Figure 1: Requisite parents are marked with a green V.

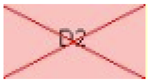


Figure 2: Non-requisite parents are marked with a red X.

Note: If the model is a class, this function is only available in run-mode. If the model is a flat net, the function is available in both edit-mode and run-mode.

4.6.5 Requisite Ancestors

Inspecting requisite ancestors for a decision node may be useful for solving a network with decisions and utlity functions as a traditional influence diagram where all observations and all decisions of the past must be remembered.

Not all available observations matter when a decision must be made. In order to get the optimal solution of an influence diagram, relevant past observations and decisions must be added as parents of each decision node in the LIMID network.

Ancestors of a decision node is said to be requisite if the value of the ancestor may affect the optimal choice of the decision.

Inspect the requisite ancestors of a decision node

Select a decision node. Click Network -> Analysis -> Show requisite ancestors for decision.

Requisite ancestors will be marked with a green V (see Figure 1).

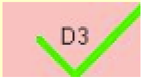


Figure 1: Requisite ancestors are marked with a green V.

Note: If the model is a class, this function is only available in run-mode. If the model is a flat net, the function is available in both edit-mode and run-mode.

4.6.6 Data Conflict Analysis

Conflict analysis is the activity of detecting, tracing, and explaining possible conflicts among observations of variable values (i.e., evidence or data). Inconsistencies among observations are easily detected (P(evidence) = 0), but also flawed findings should be detected and traced. For example, in a diagnostic situation a single flawed test result may take the investigation in a completely wrong direction.

Data conflicts are indicated in the right-hand side of the status bar of the *Main Window* (page 186) and in the *Junction Tree Window* (page 254) (see Figures 1 and 2). Please note that a positive conflict measure indicates negatively correlated observations (i.e., a possible conflict) and that a negative conflict measure indicates positively correlated observations.

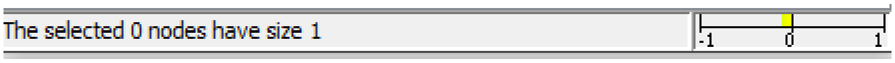
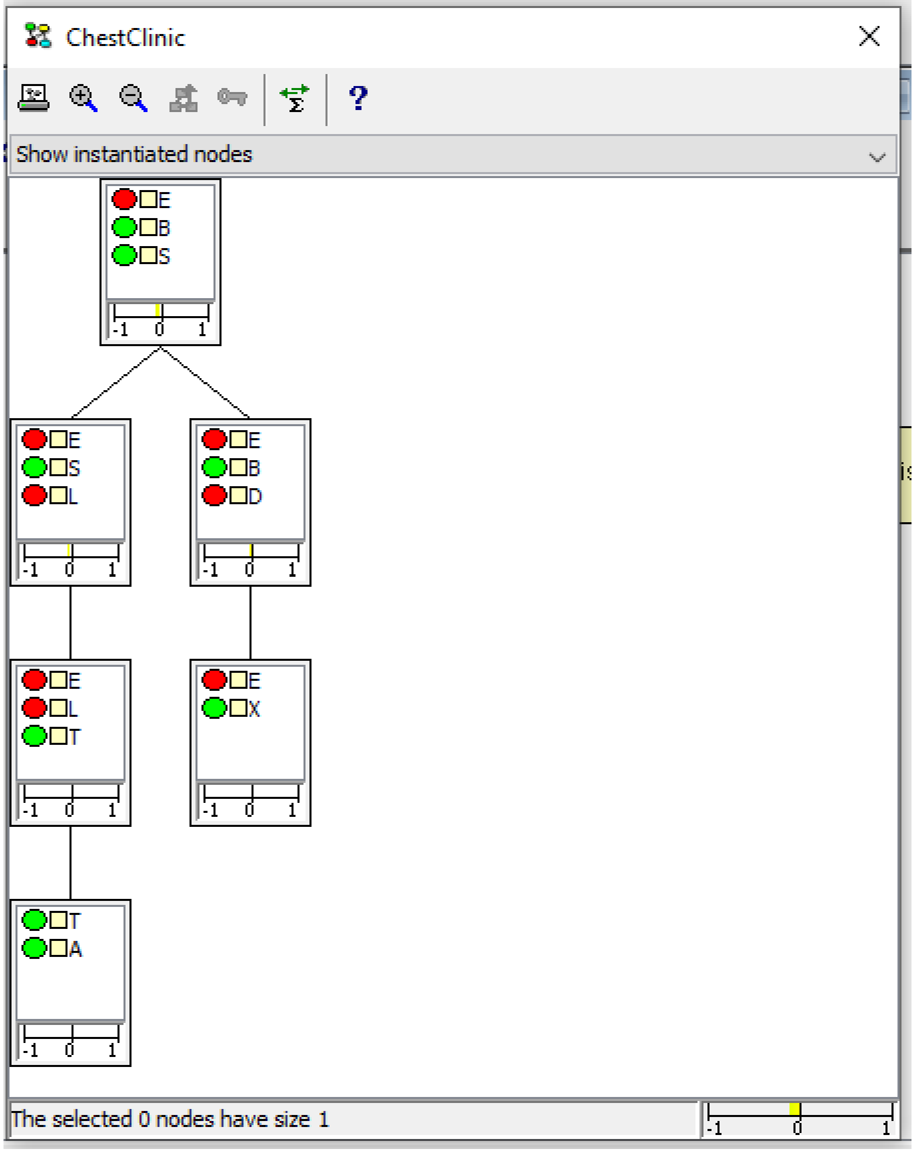

The selected 0 nodes have size 1

Figure 1: The conflict measure computed for the entered evidence is shown in the right-hand side of the status bar of the Main Window.

© Copyright 2023, HUGIN EXPERT A/S



ChestClinic

Show instantiated nodes

The selected 0 nodes have size 1

Figure 2: The overall conflict measure computed for the entered evidence is shown in the root clique of the junction tree and in the right-hand side of the status bar of the Junction Tree Window. Local conflicts pertaining to evidence entered in a sub-trees are indicated in the root cliques of these sub-trees.

To understand what conflict analysis is and how it can be used, there are several issues of interest:

- *Definition of Data Conflict* (page 294): How do we define data conflict?
- *Conflict Measure* (page 294): How do we define an appropriate measure of data conflict?
- *Conflict Resolution* (page 296): How do we distinguish between a conflict and a rare case?
- *Tracing Conflicts* (page 297): How do we identify the pieces of evidence that contribute to a data conflict?

Definition of Data Conflict

We define two sets of observations e_1 and e_2 to be in a possible conflict with one another if they are negatively correlated.

For positively correlated findings we expect that $P(e_{1\text{`|e:sub:`}2})$ > $P(e_1)$ and vice versa (i.e., observing e_2 makes it more likely to also observe e_1 (and vice versa)). In other words, we expect that

$$P(e_1, e_2) > P(e_1)P(e_2)$$

if e_1 and e_2 are positively correlated,

$$P(e_1, e_2) < P(e_1)P(e_2)$$

if e_1 and e_2 are negatively correlated, and

$$P(e_1, e_2) = P(e_1)P(e_2)$$

if e_1 and e_2 are independent.

Conflict Measure

Therefore, given a set of observations (evidence), e = $\{e_1,\ldots,e_n\}$, we define the conflict measure for e as

$$conf(e) = log\frac{\prod_{i=1}^{n} P(e_i)}{P(e)}$$

If conf(e) is positive, $e_1,\ldots,e_n$ are negatively correlated, indicating a possible conflict among these pieces of evidence. (The choice of base for the log function is immaterial.)

Notice, that if conf(e) is negative (i.e., no apparent conflict among $e_1,\ldots,e_n$), then this gives you no guarantee that all of $e_1,\ldots,e_n$ are positively correlated. It may well happen that there is a local conflict (i.e., that conf(e') > 0 for a proper subset e' of e) although conf(e) < 0.

As an example, consider the junction tree in Figure 3, where there are five pieces of evidence (marked by the red nodes). The overall conflict measure is -0.54, indicating no conflict. This measure is obtained from the root clique.

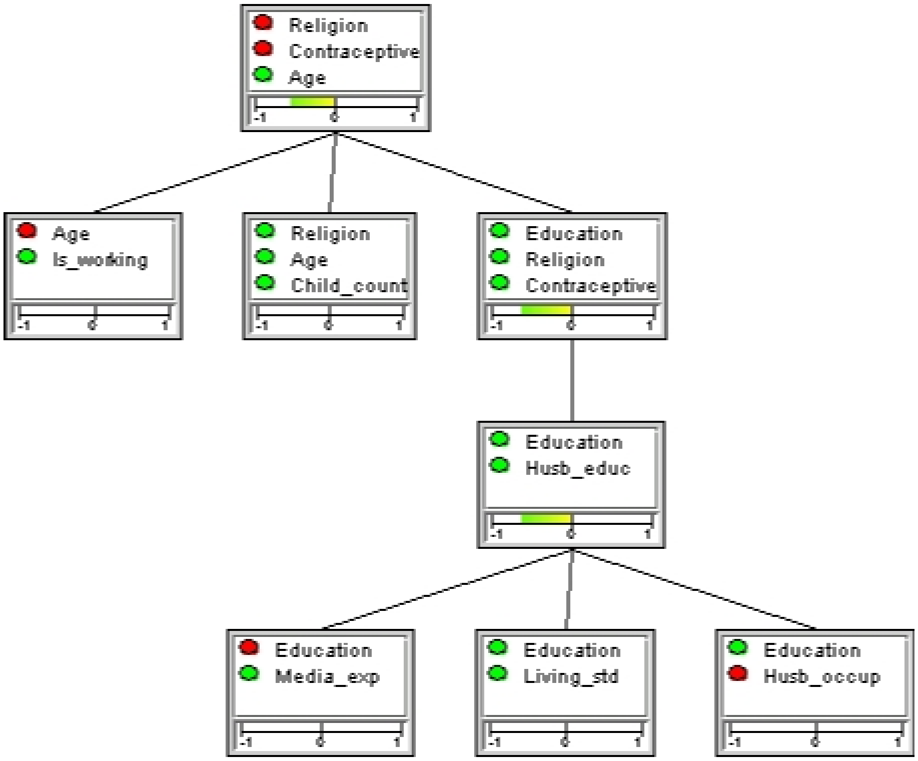


Figure 3: The overall conflict measure for the five pieces of evidence entered (marked by the red nodes) is -0.54, indicating no conflict. Note that the local conflict measure for the two pieces of evidence on Education and Husb_occup ("Husband Occupation") is less than the overall conflict.

The measure indicated in each clique is the conflict measure pertaining to the evidence entered in the sub-tree with that clique as root. For example, the conflict measure of the clique containing nodes "Education" and "Husb_educ" is -0.6 and pertains to the two pieces of evidence associated with "Education" and "Husb_occup".

To compute the local conflict measure for the three pieces of evidence associated with "Age", "Religion" and "Contraceptive" one can select, for example, the clique { "Education", "Religion", "Contraceptive" } as root (see Figure 4). (For details on selecting a root of a junction tree, see the *Junction Tree section* (page 254).) Now, it turns out that there is a slight conflict among these three pieces of evidence (conflict measure is 0.39).

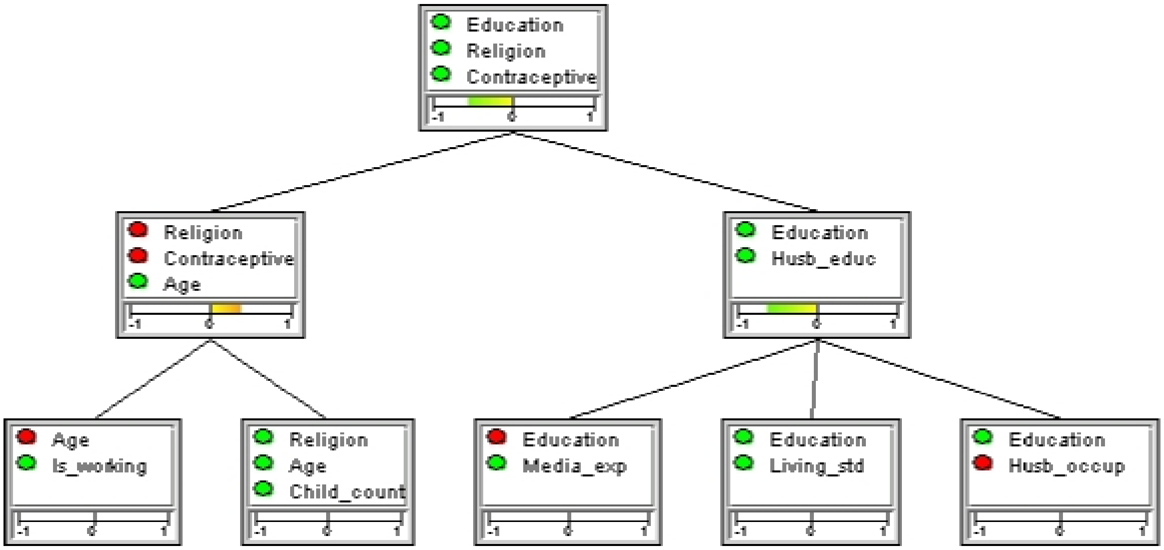


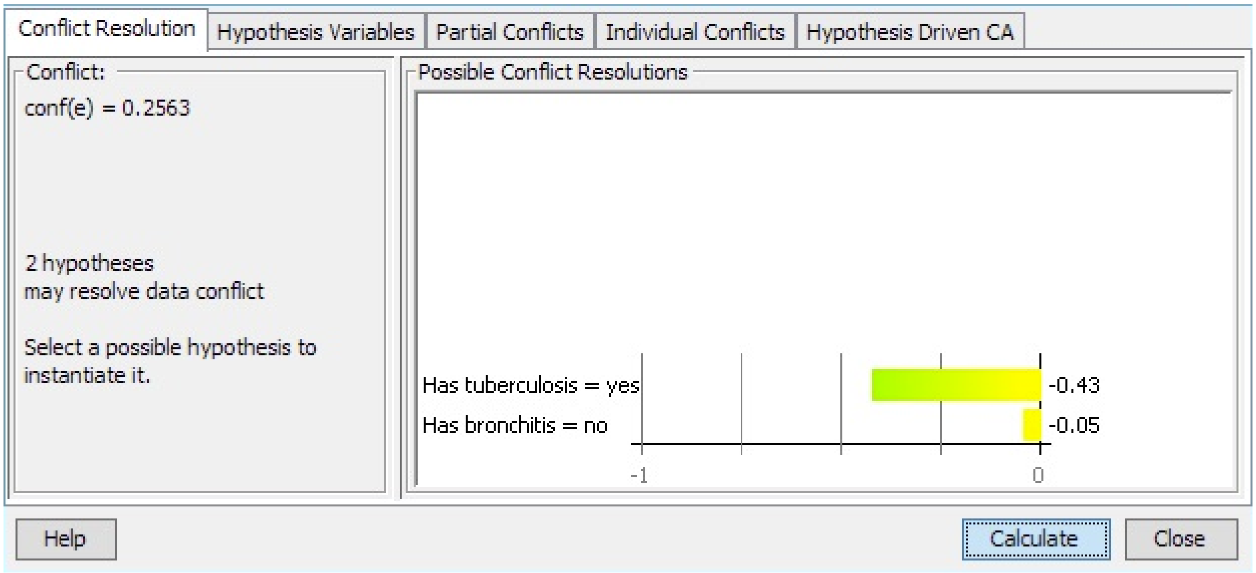
Figure 4: There is a slight conflict among the values observed for the three variables “Age”, “Religion” and “Contraceptive” (conflict measure is 0.39), although there is no overall conflict.

Conflict Resolution

There are situations in which a positive conflict measure is computed, where there is no real conflict. These include:

- *Rare case*: Typical data from a very rare case may indicate a possible conflict. If conf(e_1,…,e_n) > 0 and there is a hypothesis H=h such that conf(e_1,…,e_n,h) < 0, then h explains away the conflict. That is, if H=h is the correct hypothesis (e.g., a diagnosis) in the current situation, then there is no conflict.
- *Missing observation*: Basically the same situation, where conf(e_1,…,e_n) > 0 but conf(e_1,…,e_n,I=i) < 0, where I=i is a missing piece of information. That is, there is a local conflict among e_1,…,e_n, but the observation I=i explains the conflict.

By activating the button with the 🔑 symbol, one can obtain a list of possible instantiations of currently uninstantiated variables that can eliminate the current conflict. The hypotheses are chosen from ones that are set up in the “Hypothesis Variables” tab. An example of the dialog box that appears when this button is activated is shown in Figure 5.



conf(e) = 0.2563

2 hypotheses
may resolve data conflict

Select a possible hypothesis to
instantiate it.

Figure 5: Conflict resolutions dialog box. The box contains a list of possible instantiations of currently uninstantiated variables that can eliminate the current conflict.

The dialog box contains a list of possible instantiations in the form

::

```
<CM>:<variable>=<value>
```

where <CM> is the new conflict measure obtained if <variable> is instantiated to <value>. Only instantiations (if any) with a resulting conflict measure less than or equal to 0 get displayed.

The Instantiate button enters the currently selected instantiation (if any) as evidence.

Tracing Conflicts

Whenever a positive conflict has been observed that cannot be explained as a rare case, it is important to pinpoint the piece (or pieces) of evidence that is in conflict with the majority of the pieces of evidence.

Basically, this involves computation of conflict measures for subsets of the evidence. As mentioned and illustrated above, the *junction tree* (page 254) is useful for this purpose. As an example, consider the junction tree in Figure 6, where four pieces of evidence have been entered (marked by the red nodes).

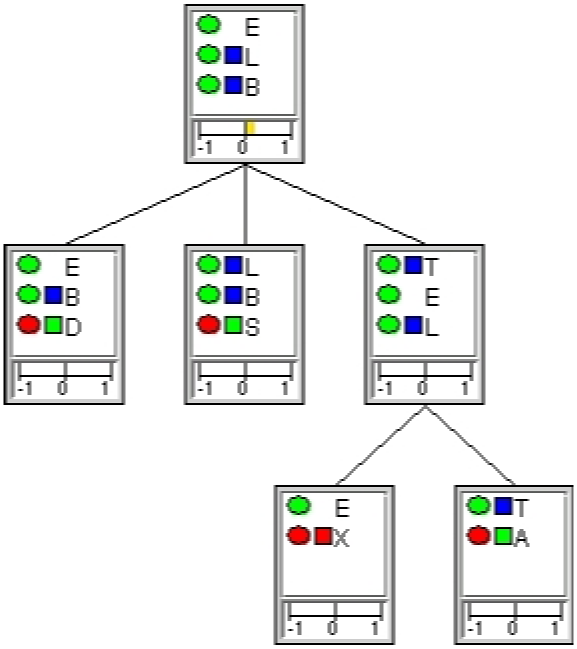


Figure 6: A slight conflict among the pieces of evidence entered (red nodes).

The conflict measure of 0.21 indicates a slight conflict among the these pieces of evidence. To trace the source of the conflict, we can investigate local conflict measures. The junction tree shown above does not reveal any definitive clues as to which piece could be the one causing the conflict. It does show, however, that the local conflict measure is zero for the sub-tree rooted at clique { T, E, L }, indicating that the observations on X and A are not in conflict with one another.

To investigate further, we may select another clique as root of the junction tree, so as to compute some different local conflict measures. If we select clique { T, E, L } as the root of the tree (see Figure 7), we find that there is a slight local conflict among the observed value for D and the observed value for S.

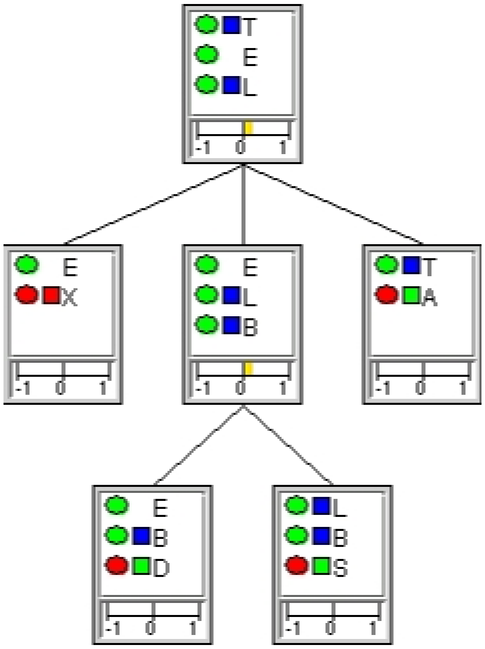


Figure 7: A slight local conflict among the observed values for D and S.

4.6.7 Evidence Sensitivity Analysis

What is Evidence Sensitivity Analysis?

Evidence sensitivity analysis (SE analysis) is the analysis of how sensitive the results of a belief update (propagation of evidence) is to variations in the set of evidence (observations, likelihood, etc.).

Consider the situation where a decision maker has to make a decision based on the probability distribution of a hypothesis variable. It could, for instance, be a physician deciding on a treatment of a patient given the probability distribution of a disease variable. Prior to deciding on a treatment the physician may have the option to investigate the impact of the collected information on the posterior distribution of the hypothesis variable. Given a set of findings and a hypothesis, which sets of findings are in favor, against, or irrelevant for the hypothesis, which sets of findings discriminate the hypothesis from an alternative hypothesis, what if a variable had been observed to a different value than the one observed, etc. These questions can be answered by SE analysis.

Given a Bayesian network model and a hypothesis variable, the task is to determine how sensitive the belief in the hypothesis variable is to variations in the evidence. We consider myopic hypothesis driven SE analysis on discrete random variables.

Evidence sensitivity analysis is available for discrete chance nodes only if there are no unpropagated evidence. The functionality is enabled when a discrete chance node is selected or when pressing the right mouse button on a discrete chance node in the *Node List Pane* (page 243) (and evidence has been entered and propagated).

Distance Measures

The main purpose of hypothesis driven SE analysis is to investigate how changes in the set of evidence impact the probability of a hypothesis. In order to perform this investigation two distance measures are required. Each distance measure will provide a numerical value specifying the distance between either two probabilities or two probability distributions.

Let X be a hypothesis variable with state space $x_1,\ldots,x_n$ and let ε = { $\varepsilon_1, \ldots, \varepsilon_m$} be a set of evidence (findings). We let ε_Y in ε denote the finding on variable Y in X(ε).

The distance d(p,q) between two probabilities p and q is defined, for p different from zero, as:

$$d(p, q) = |\frac{q}{p} - 1|$$

This measure is, for instance, useful for measuring the distance between the probability P(x| ε) of hypothesis x given evidence ε and the probability P(x| ε{ ε_i}) of hypothesis x given evidence ε{ ε_i}, i.e., the set of evidence where ε_i is excluded from ε.

A pair of probabilities p and q are said to be *almost equal* when their distance d(p,q) is below a predefined threshold δ, i.e., d(p,q) < δ.

The *cost-of-omission* c(P(X| ε), P(X| ε{ ε_i})) of ε_i is defined as:

$$c(P|X|\varepsilon), P(X|\varepsilon \setminus \{\varepsilon_i\}) = \sum_{x \in dom(x)} P(X|\varepsilon) log(\frac{P(x|\varepsilon)}{P(x|\varepsilon \setminus \{\varepsilon_i\})})$$

This formula is undefined for values P(x| ε)=0 and P(x| ε{ ε_i})=0. For these two cases, we define cost-of-omission to be 0 and infinity, respectively. Notice the difference between the distance measure and the cost-of-omission measure. The distance measure evaluates the distance between probability values whereas the cost-of-omission measure evaluates the distance between two posterior probability distributions relative to omitting a certain finding ε_i from the evidence ε. The cost-of-omission measure is a special case of the more general cross entropy distance (or Kullback-Leibler distance) measure between a probability distribution P and an approximation P' of P:

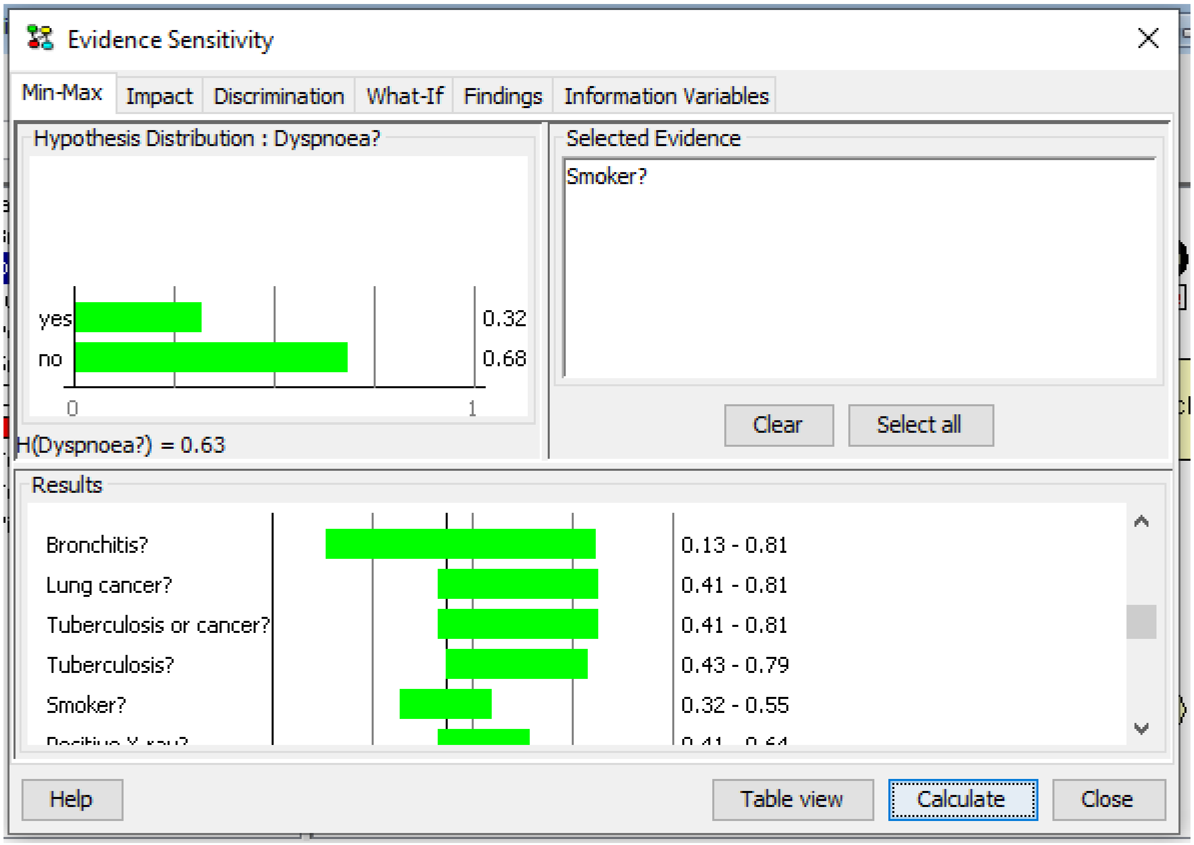
$$H(P, P') = \sum_{x \in dom(x)} P(x) log(\frac{P(x)}{P'(x)})$$

The cost-of-omission measure is, for instance, useful for measuring the distance between the posterior probability distribution P(X| ε) of hypothesis variable X given evidence ε and the posterior probability distribution P(X| $\varepsilon\{\varepsilon_i\}$) of hypothesis variable X given evidence $\varepsilon\{\varepsilon_i\}$, i.e., the set of evidence where ε_i is excluded from ε.

Min-Max

As part of performing a SE analysis we may be interested in knowing the minimum and maximum values of the posterior belief in each possible state of the hypothesis variable given all possible observations on a given variable.

Using the dialog shown in Figure 1 we may for each *information variable* (page 307) determine the minimum, current, and maximum value of the posterior probability of each state of the hypothesis variable. The minimum and maximum values are determined by entering and propagating each state of the information variable. This analysis requires one propagation for each state of each possible information variable.



Selected Evidence: Smoker?

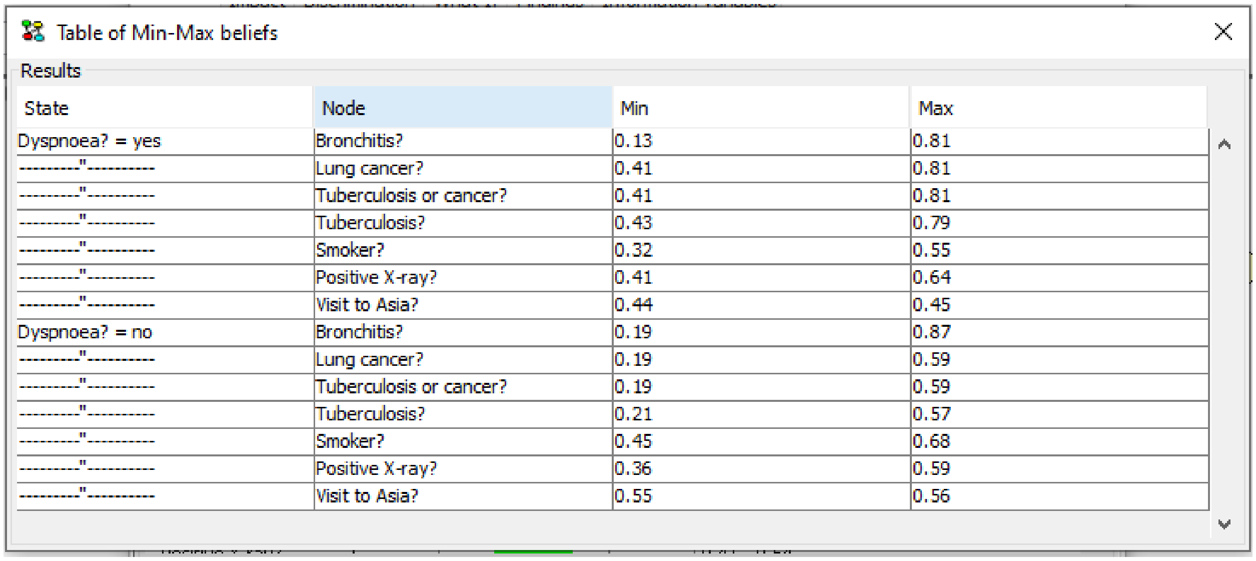
H(Dyspnoea?) = 0.63

Bronchitis?	0.13 - 0.81
Lung cancer?	0.41 - 0.81
Tuberculosis or cancer?	0.41 - 0.81
Tuberculosis?	0.43 - 0.79
Smoker?	0.32 - 0.55
Positive X-ray?	0.41 - 0.64

Figure 1: The entropy of the hypothesis variable B is H(B)=0.63.

Figure 1 shows the posterior probability distribution of *Has bronchitis* given the entire set of evidence, the entropy H(Has bronchitis)=0.63 of *Dyspnoea?* as well as the minimum, current, and maximum values for the posterior probability of *Dyspnoea* equal to yes given variations over each selected information variable (i.e., *Has bronchitis*, *Tuberculosis* or *cancer*, etc).

Pressing the *Table View* button in the dialog shown in Figure 1 produces the table shown in Figure 2.



Results

State	Node	Min	Max
Dyspnoea? = yes	Bronchitis?	0.13	0.81
---------"----------	Lung cancer?	0.41	0.81
---------"----------	Tuberculosis or cancer?	0.41	0.81
---------"----------	Tuberculosis?	0.43	0.79
---------"----------	Smoker?	0.32	0.55
---------"----------	Positive X-ray?	0.41	0.64
---------"----------	Visit to Asia?	0.44	0.45
Dyspnoea? = no	Bronchitis?	0.19	0.87
---------"----------	Lung cancer?	0.19	0.59
---------"----------	Tuberculosis or cancer?	0.19	0.59
---------"----------	Tuberculosis?	0.21	0.57
---------"----------	Smoker?	0.45	0.68
---------"----------	Positive X-ray?	0.36	0.59
---------"----------	Visit to Asia?	0.55	0.56

Figure 2: Table view of the results in Figure 1.

Impact

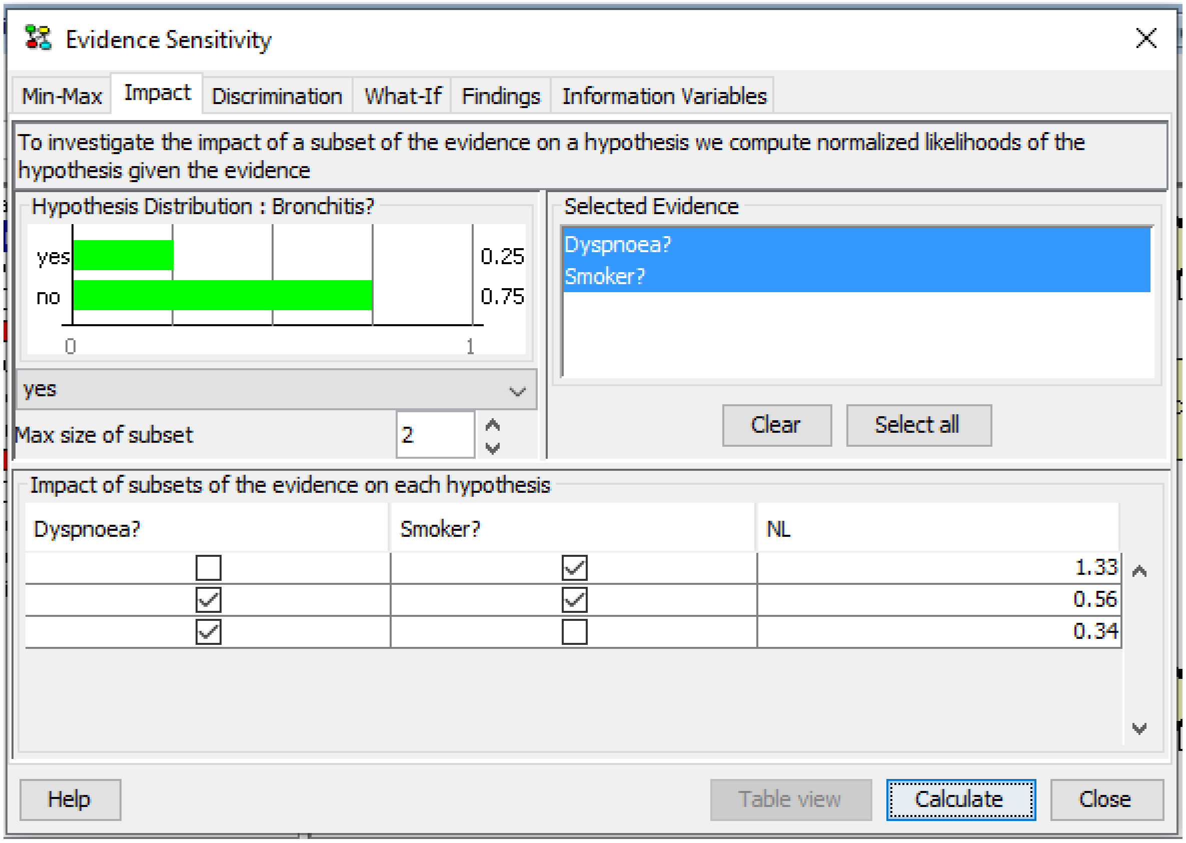
Investigation of the impact of different subsets of the evidence on each state of the hypothesis variable is a useful part of SE analysis. Investigating the impact of different subsets of the evidence on states of the hypothesis may help to determine subsets of the evidence acting in favor of or against each possible hypothesis.

The impact of a subset of the evidence on a state of the hypothesis variable is determined by computing the normalized likelihood of the evidence given the hypothesis.

$$NL = \frac{P(\varepsilon'|x)}{P(\varepsilon')} = \frac{P(\varepsilon', x)/P(x)}{P(\varepsilon')} = \frac{P(x|\varepsilon')P(\varepsilon')/P(x)}{P(\varepsilon')} = \frac{P(x|\varepsilon')}{P(x)}$$

This analysis requires one propagation for each subset of the selected evidence.

Using the dialog shown in Figure 3 we may investigate the impact of all subsets of the selected evidence on each state of the hypothesis variable.



To investigate the impact of a subset of the evidence on a hypothesis we compute normalized likelihoods of the hypothesis given the evidence

Selected Evidence: Dyspnoea?, Smoker?

Hypothesis state: yes

Max size of subset: 2

Dyspnoea?	Smoker?	NL
□	☑	1.33
☑	☑	0.56
☑	□	0.34

Figure 3: The normalized likelihood (NL) of different subsets of the evidence for the selected state of the hypothesis variable.

Figure 3 shows that the normalized likelihood (NL) of the hypothesis *Has bronchitis* = *yes* given on *Smoker*? is 1.33. This suggests that evidence supports the hypothesis that the patient is suffering from bronchitis (since the normalized likelihood is above one). The evidence on *Dyspnoea?* is in favor of the alternative hypothesis.

Discrimination

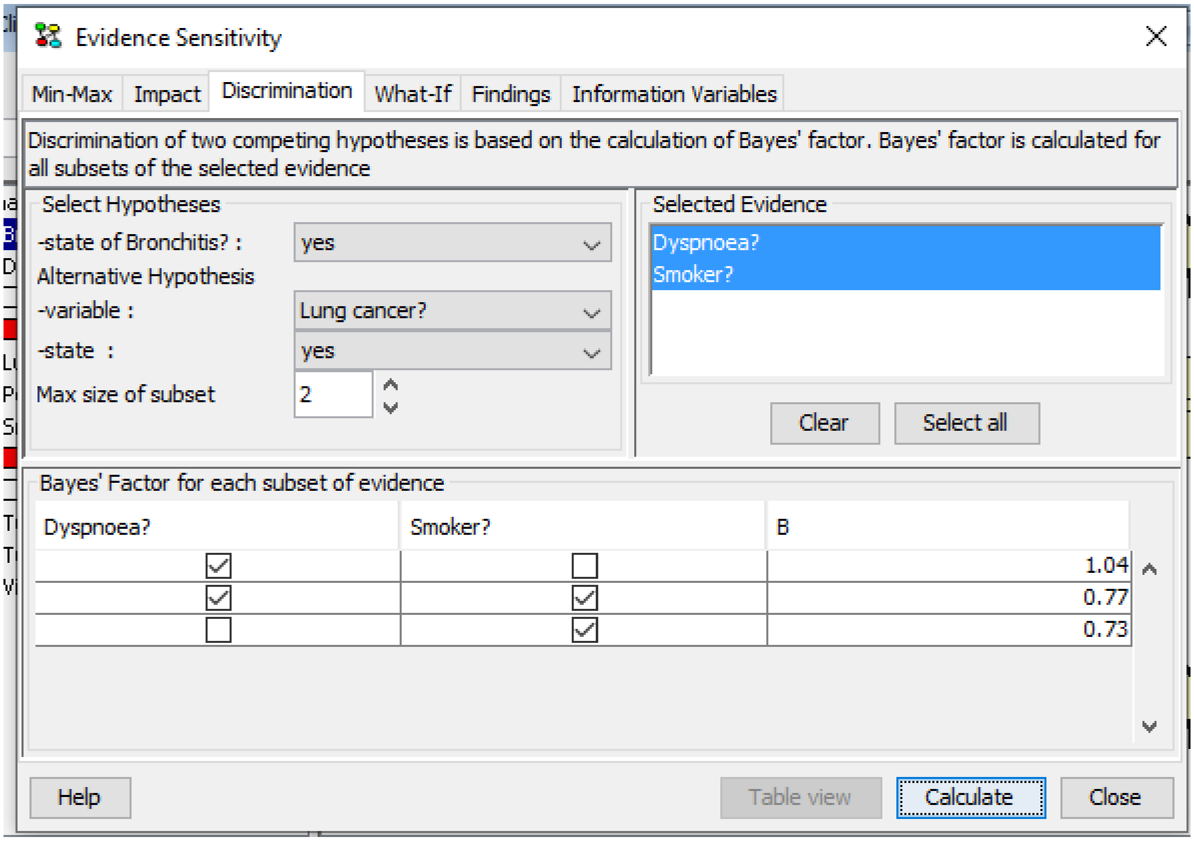
A central question considered by SE analysis is the question of how different subsets of the evidence discriminates between competing hypotheses. The challenge is to compare the impact of subsets of the evidence on competing hypotheses.

We consider the discrimination between two different hypotheses represented as states of two different variables. The discrimination of a pair of competing hypotheses is based on the calculation of Bayes' factor for all subsets of a selected set of evidence.

$$B = \frac{\text{posterior odds ratio}}{\text{prior odds ratio}} = \frac{P(x|\varepsilon')/P(y|\varepsilon')}{P(x)/P(y)} = \frac{P(\varepsilon'|x)}{P(\varepsilon'|y)} = \frac{L(x|\varepsilon')}{L(y|\varepsilon')}$$

This requires one propagation for each subset.

Using the dialog shown in Figure 4 we may investigate how each subsets of the selected evidence discriminates between two competing hypotheses.



Discrimination of two competing hypotheses is based on the calculation of Bayes' factor. Bayes' factor is calculated for all subsets of the selected evidence

Select Hypotheses

-state of Bronchitis? : yes

Alternative Hypothesis

-variable : Lung cancer?

-state : yes

Max size of subset 2

Selected Evidence

Dyspnoea?
Smoker?

Bayes' Factor for each subset of evidence

Dyspnoea?	Smoker?	B
☑	□	1.04
☑	☑	0.77
□	☑	0.73

Figure 4: Discrimination between the hypothesis Has bronchitis = true and the alternative hypothesis Has lung cancer = true.

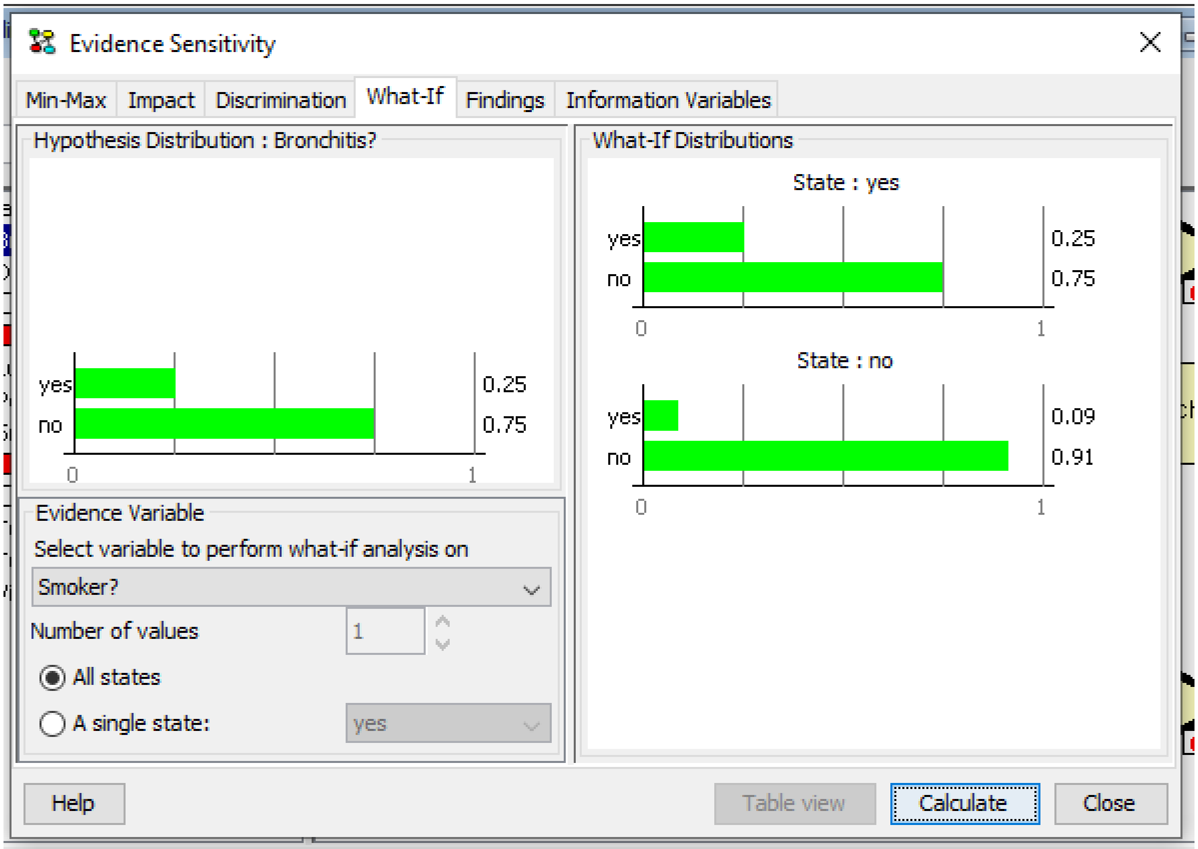
Figure 4 shows Bayes' factor between the hypothesis *Has bronchitis = true* and the alternative hypothesis *Has lung cancer = true* for each subset of the selected evidence. All subsets (except the empty set) support the alternative hypothesis (i.e. that the patient is suffering from lung cancer)

What-If

In *what-if* SE analysis the type of question considered is the following. What if an observed discrete or continuous random variable had been observed to a value different from the value actually observed. For an observed continuous chance node we consider a limited number of possible values computed based on the mean and standard deviation of the node. The number of values considered for a continuous node is specified by the user as *Number of values*. This the number of values above and below the mean value, i.e., twice this number of values will be consider plus the mean value.

We consider a hypothesis driven approach to what-if analysis. Hypothesis driven what-if analysis is performed by computing the posterior distribution of the hypothesis variable for each possible state of the observed variable.

Using the dialog shown in Figure 5 we may investigate the consequences of changing the observed state of an evidence variable.



Select variable to perform what-if analysis on: Smoker?

Number of values: 1

● All states

○ A single state: yes

Figure 5: The posterior distribution of the hypothesis variable Has bronchitis has a function of the observed state of Smoker?.

Figure 5 shows how the posterior distribution of the hypothesis variable *Has bronchitis* changes as a function of the observed state of *Smoker?*.

Findings

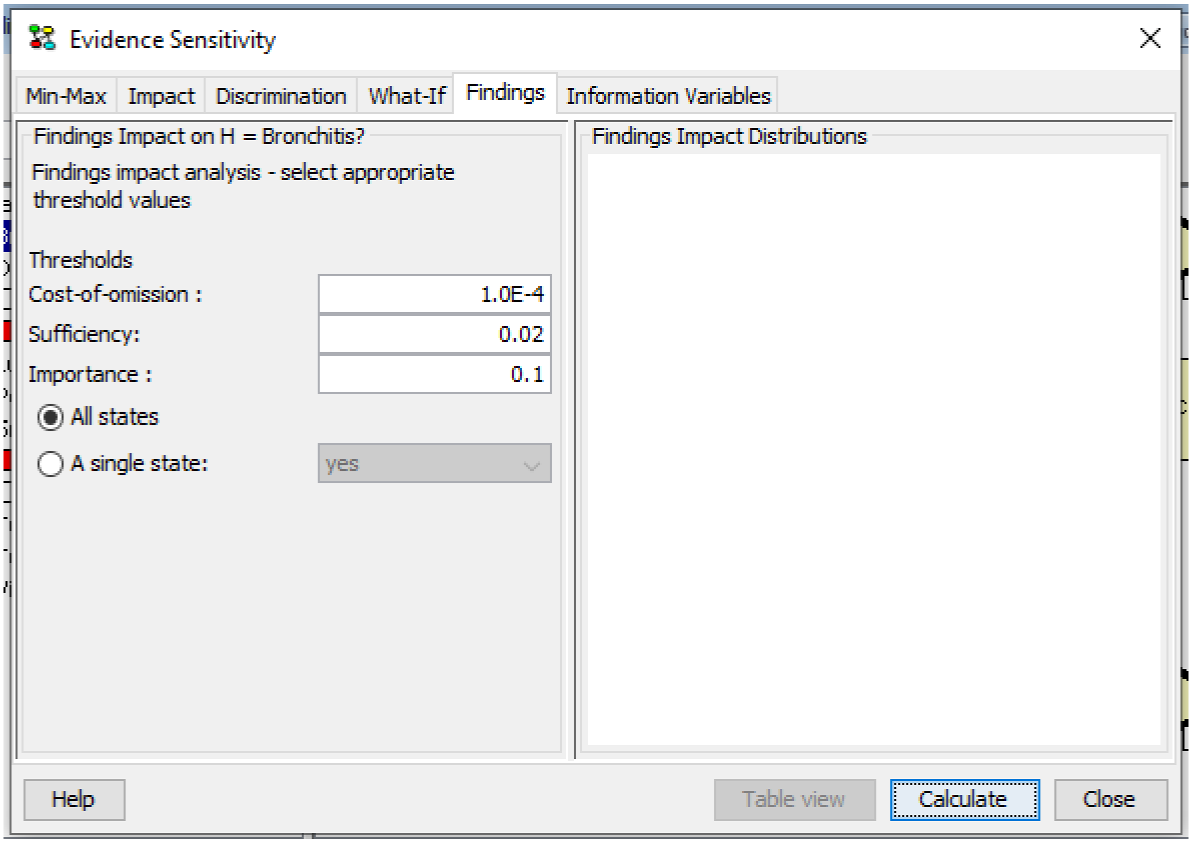
The impact of each finding on the probability of the hypothesis is determined by computing and comparing the prior probability of the hypothesis, the posterior probability of the hypothesis given the entire set of evidence and the posterior probability of the hypothesis given the entire set of evidence except the finding.

The notions of an important finding, a sufficient set of evidence, and a redudant finding are central to findings SE analysis:

- A finding is *important* when the difference between the probability of the hypothesis given the entire set of evidence except the finding and the probability of the hypothesis given the entire set of evidence is too large.
- A subset of evidence, e.g. the entire set of evidence except a certain finding, is *sufficient* when the probability of the hypothesis given the subset of evidence is *almost equal* to the probability of the hypothesis given the entire set of evidence.
- A finding is *redundant* when the probability of the hypothesis given the entire set of evidence except the finding is *almost equal* to the probability of the hypothesis given the entire set of evidence.

The impact of each finding may be considered for each state or a certain state of the hypothese variable. Sufficiency and importance thresholds are specified by the user.

Using the dialog shown in Figure 6 and 7 we may investigate the of a finding on the posterior distribution of the hypothesis variable.



Findings Impact on H = Bronchitis?

Findings impact analysis - select appropriate threshold values

Thresholds

Cost-of-omission : 1.0E-4

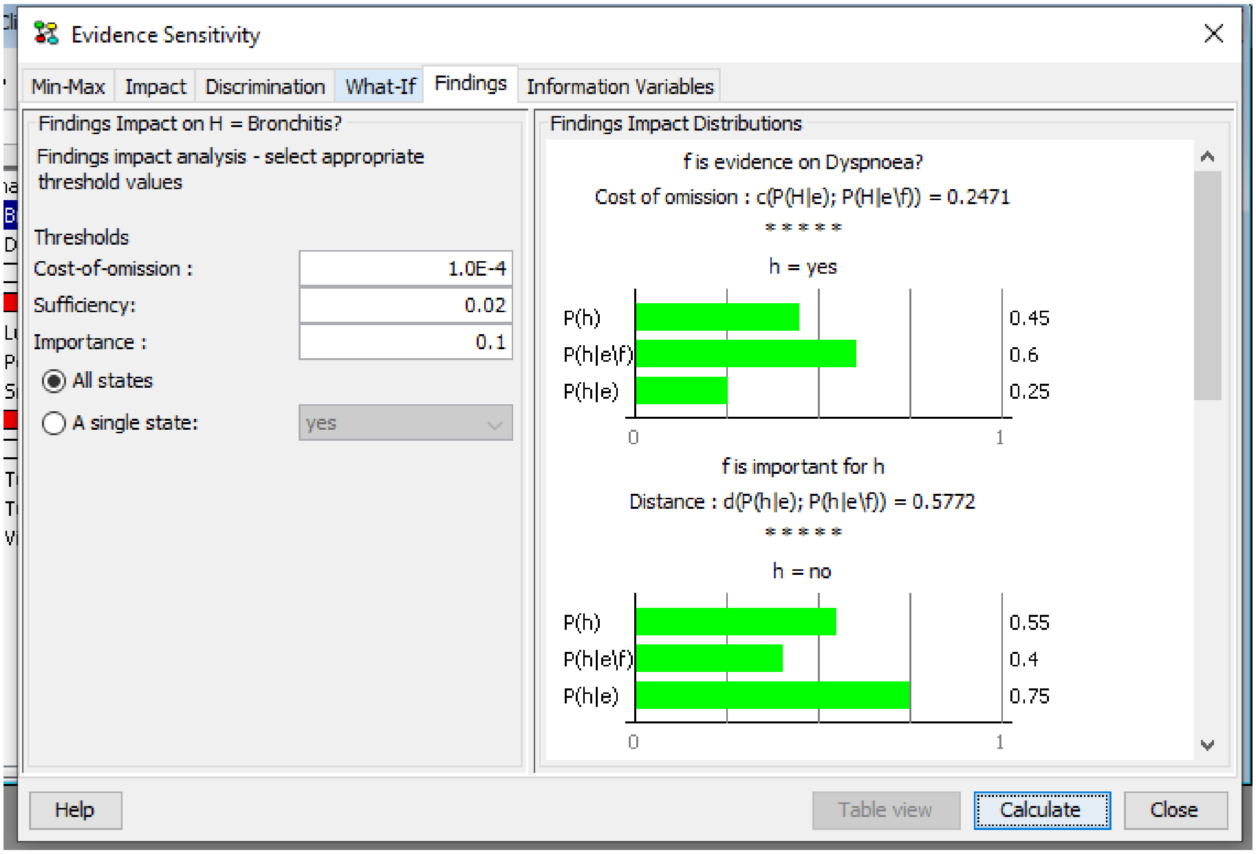
Sufficiency: 0.02

Importance : 0.1

● All states

○ A single state: yes

Figure 6: Findings



Findings Impact on H = Bronchitis?

Findings impact analysis - select appropriate threshold values

Thresholds

Cost-of-omission : 1.0E-4

Sufficiency: 0.02

Importance : 0.1

● All states

○ A single state: yes

Findings Impact Distributions

f is evidence on Dyspnoea?

Cost of omission : c(P(H|e); P(H|e\f)) = 0.2471

*****

h = yes

f is important for h

Distance : d(P(h|e); P(h|e\f)) = 0.5772

*****

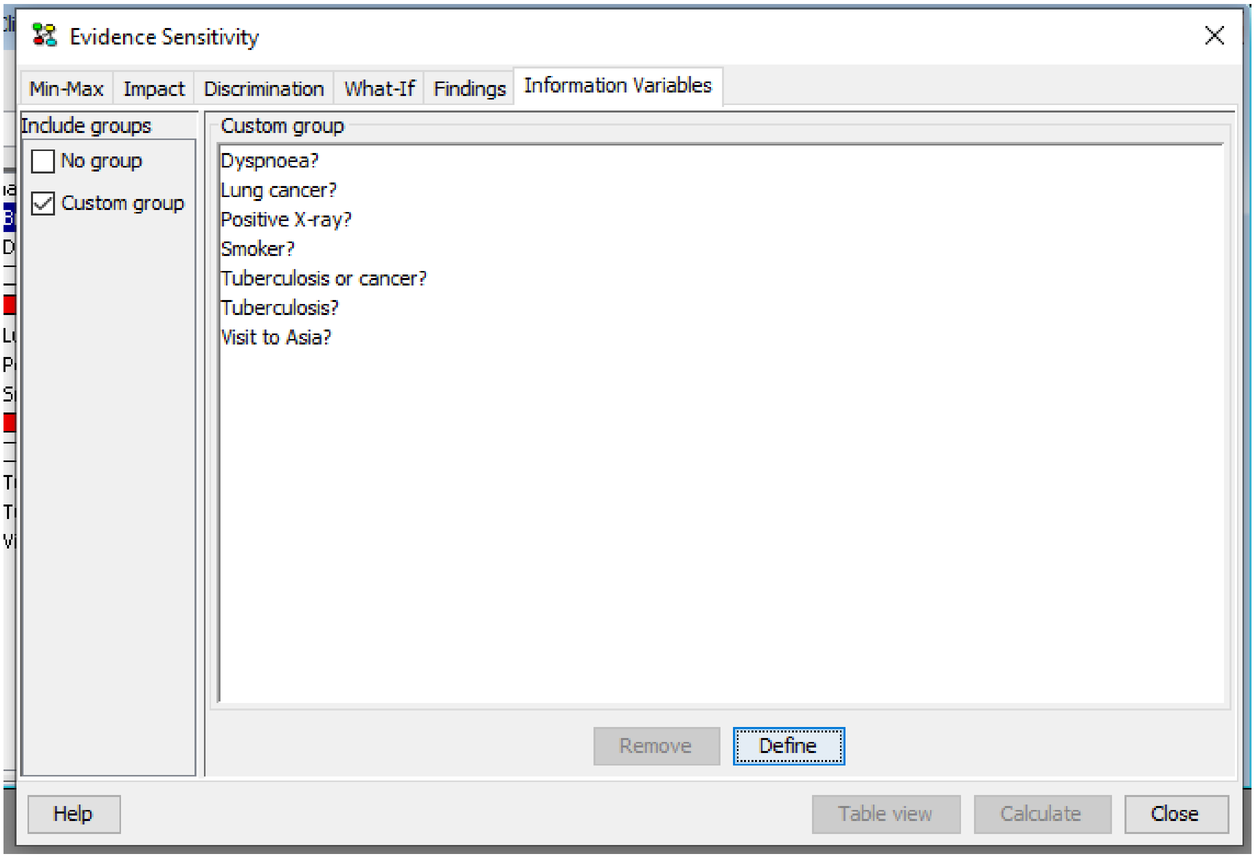
h = no

Figure 7: The impact of findings on B.

Figure 6 and 7 illustrates the impact of the finding on *Smoker?* on the hypothesis *Has bronchitis = false* (corresponding to H in state false, i.e. h=false).

Information Variables

Min-max SE analysis on a hypothesis variable is performed relative to a set of information variables. The set of information variables can be selected as indicated in Figure 8.



Include groups

□ No group

☑ Custom group

Custom group

Dyspnoea?
Lung cancer?
Positive X-ray?
Smoker?
Tuberculosis or cancer?
Tuberculosis?
Visit to Asia?

Figure 8: Selecting the set of information variables.

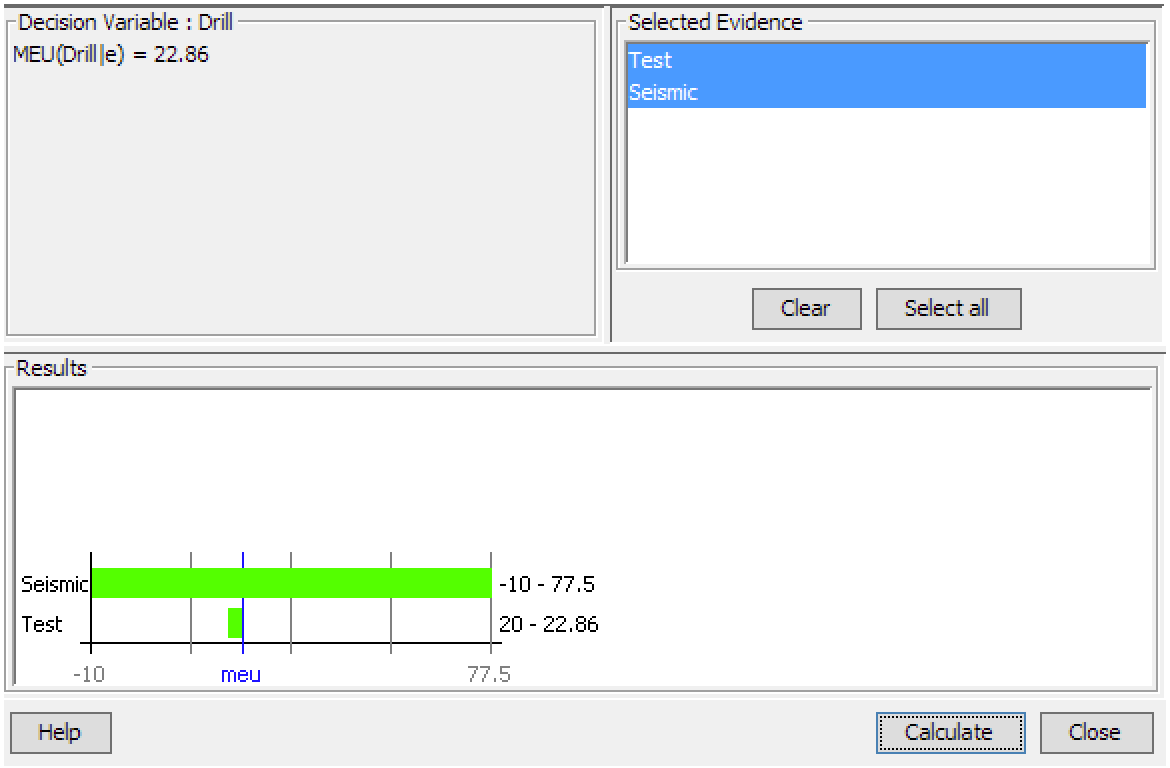
Selecting information variables proceeds in the same way as selecting Target(s) of Instantiations in the d-Separation pane.

4.6.8 Evidence Sensitivity Analysis (Decision)

Evidence sensitivity analysis (SE analysis) with respect to a decision node is the analysis of how sensitive the expected utility of a decision node is to variations in the set of observed evidence (i.e., observations on chance nodes made prior to the decision). The analysis aims to answer the questions such as, for instance, *how does the expected utility of the decision change as the value observed for a single chance node is changed keeping the remaining observed nodes fixed?*. The analysis is performed for each selected evidence node in turn. This can be useful for identifying the most and least important observation.

Given an influence diagram model and a decision node, the task is to computed the expected utility of the decision options for each possible value of each observed node. This is solved by iterating through the states of each observed node and computing the expected utility given that state (and the remaining set of evidence).

Using the dialog shown in Figure 1 we determine for each selected evidence variable the minimum, current, and maximum value of the expected utility of the decision.



Decision Variable : Drill

MEU(Drill|e) = 22.86

Selected Evidence

Test
Seismic

Results

Seismic -10 - 77.5
Test 20 - 22.86

Figure 1: The sensitivity to evidence dialog for a decision node.

Figure 1 shows the maximum expected utility (MEU) of the *Drill* decision given the evidence is 22.86 where the evidence is on *Test* (observed to yes) and *Seismic* (observed to open, which is an indication of some oil) in the upper left corner. For each selected evidence node, the minimum and maximum MEU are illustrated using a green bar. This shows how the MEU varies with the observed value for the selected node. The MEU as a function of the value of *Seismic* is between -10 and 77.5. It is -10, if we change the observed value to di (i.e., diffuse reflection pattern, which is an indication of almost no hope of oil) and 77.5, if we change the observed value to cl (i.e., closed reflection pattern, which is an indication of a lot of oil). It is clear that changes to the observed value for *Seismic* produce the most variation in the expected utility for the decision.

4.6.9 Utility Sensitivity Analysis

The utility sensitivity wizard performs n-way utility sensitivity analysis. Use the wizard to inspect the impact that changes in utility- and discrete chance node parameters has on decision utilities.

Analysis is performed with respect to a single target decision node. A number of tests can be performed by varying the parameters of a single parameter node (utility or discrete chance).

To launch the utility sensitivity wizard, switch to *Run Mode* (page 123) and select the target decision node. Then open the *network menu* (page 185) -> *analysis methods* (page 288) and select *Utility Sensitivity*.

The utility sensitivity wizard can be seen in figure 1.

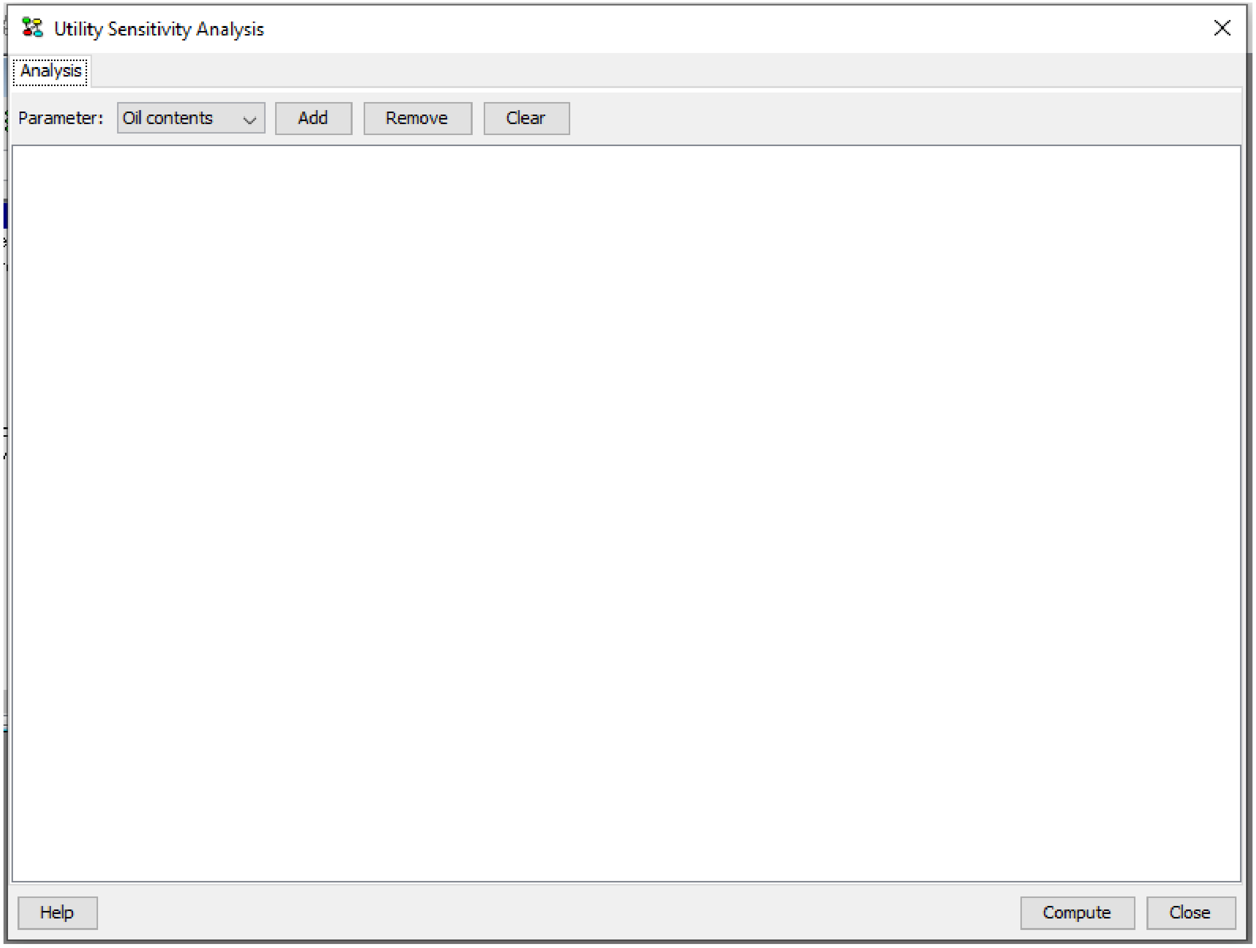


Figure 1: Utility sensitivity wizard window.

Initially the utility sensitivity wizard contains a blank sheet, to which a number of tests can be added. Add a test using the controls at the top of the frame:

- Parameter - the node wich parameters we will modify
- Add - add a test using selected paramter node
- Remove - remove last test from sheet
- Clear - clear the entire sheet

Test Components

Each test added to the sheet contains a pair of tables side by side. The *Target* table reports the decision utilities for the target decision node and also the overal expected utility given the updated parameters and policies. Each cell in the *Target* table contains the expected utility if the given configuration was entered as evidence. The *Parameter* table represents the utility table or CPT of the parameter node.

For each test, the *Target* table contains the results, and the *Parameter* table is used for configuring input to the test.

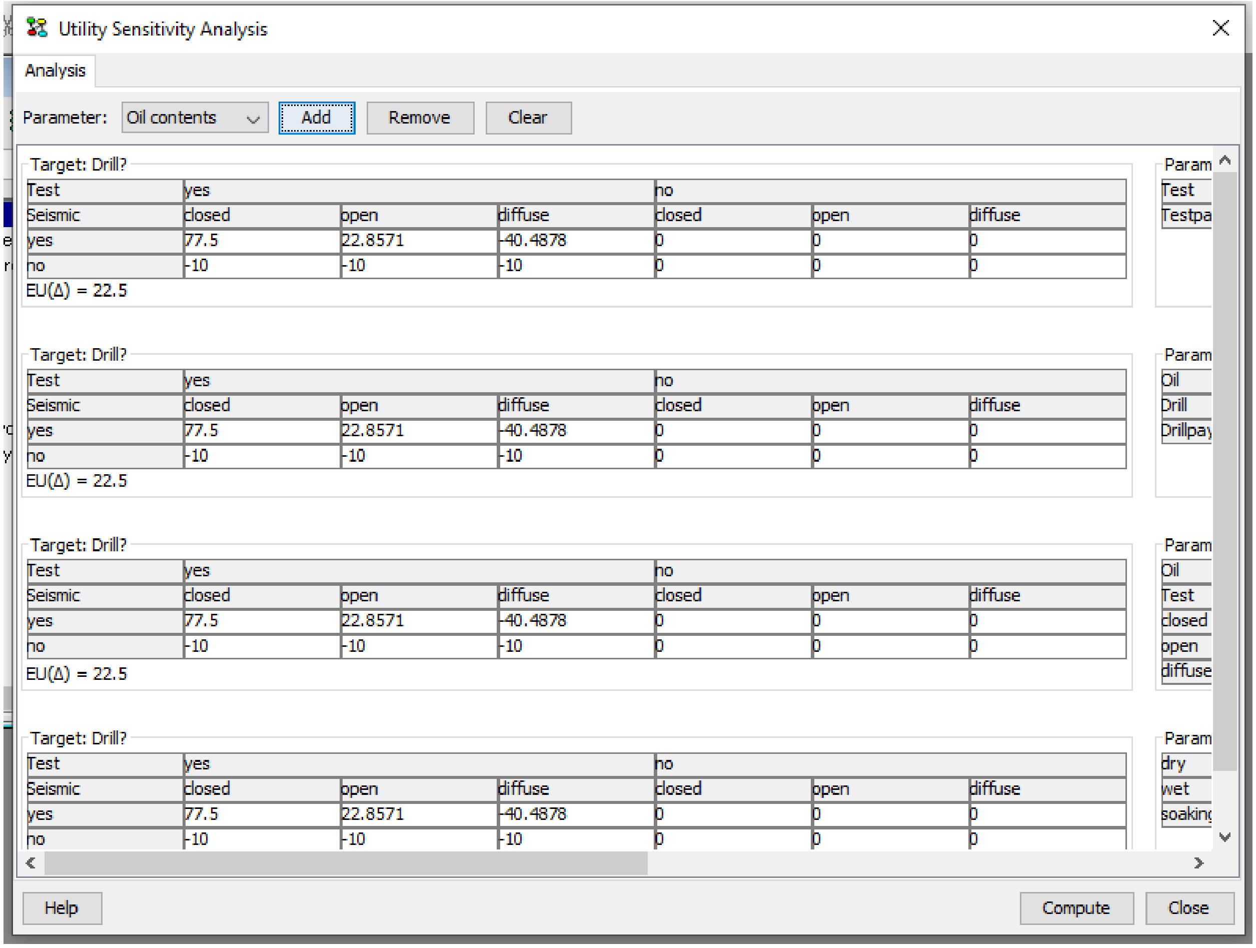
The basis for the computations in each test is the current table, policies and evidence entered in the network (at the time the utility sensitivity wizard was launched). Computations for each test is performed following these steps:

1. The specified parameters (utility or CPT) replace original parameters of parameter node

2. A propagation is performed using the new parameter table
3. Policies are updated using Single Policy Update
4. Over all expected utility is computed (reported as EU under Target)
5. The decision utilities are computed (and polupated in Target table)

To re-compute decision utilities after modifying *Parameter* table values, click the *Compute* button (located in bottom next to *close* button).

Figure 2 show the utility sensitivity analysis wizard with a number of tests. The *Target* tables show the impact of changes in the *Parameters*.



Target: Drill?

Test	yes			no		
Seismic	closed	open	diffuse	closed	open	diffuse
yes	77.5	22.8571	-40.4878	0	0	0
no	-10	-10	-10	0	0	0

EU(Δ) = 22.5

Param

Test
Testpa

Target: Drill?

Test	yes			no		
Seismic	closed	open	diffuse	closed	open	diffuse
yes	77.5	22.8571	-40.4878	0	0	0
no	-10	-10	-10	0	0	0

EU(Δ) = 22.5

Param

Oil
Drill
Drillpay

Target: Drill?

Test	yes			no		
Seismic	closed	open	diffuse	closed	open	diffuse
yes	77.5	22.8571	-40.4878	0	0	0
no	-10	-10	-10	0	0	0

EU(Δ) = 22.5

Param

Oil
Test
closed
open
diffuse

Target: Drill?

Test	yes			no		
Seismic	closed	open	diffuse	closed	open	diffuse
yes	77.5	22.8571	-40.4878	0	0	0
no	-10	-10	-10	0	0	0

Param

dry
wet
soaking

Figure 2: Utility sensitivity wizard window, with a number of tests on the sheet.

4.6.10 Value of Information Analysis

Value of information analysis may be performed in both (limited memory) influence diagrams and Bayesian networks. The approach to value of information analysis in (limited memory) influence diagrams is, however, slightly different from the approach to value of information analysis in Bayesian networks. Thus, in the following two sections we consider in turn value of information analysis in (limited memory) influence diagrams and Bayesian networks, respectively. Each of the following two sections is self contained. This implies that each section may be read separately, but it also implies that some overlap may be present.

Limited Memory Influence Diagrams

Consider the situation where a decision maker has to make a decision given a set of observations. It could, for instance, be a physician deciding on a treatment of a patient given observations on symptoms and risk factors related to a specific disease. The decision could, for instance, be whether or not to operate the patient immediately. Prior to making the decision, the physician may have the option to gather additional information about the patient such as waiting for the result of a test or asking further questions. Given a range of options, which option should the physician choose next? That is, which of the given options will produce the most information? These questions can be answered by a value of information analysis.

Given a (limited memory) influence diagram model, a decision variable, and observations on the informational parents of the decision, the task is to identify the variable, which is most informative with respect to the decision variable.

We consider one step lookahead hypothesis driven value of information analysis on discrete random variables relative to discrete decision variables. The functionality is enabled when a discrete decision node is selected or when pressing the right mouse button on a discrete decision node in the *Node List Pane* (page 243).

Maximum Expected Utility

The main reason for acquiring additional information is to increase the maximum expected utility of the decision under consideration. The selection of the variable to observe next (e.g. the question to ask next) is based on the principle of maximizing expected utility. Expected utility is a measure of the usefulness of the decision options associated with the decision variable.

Value of Information Analysis

The value of information is a core element of decision analysis. We perform decision analysis using limited memory influence diagram representations of decision problems. The structure of an influence diagram specify a partial order on observations relative to a partial order on decisions. Value of information analysis in limited memory influence diagrams considers the impact of changing the partial order of observations relative to a decision. Assume D is the next decision to be considered and let ε be the set of observations and decisions made up to decision D. Initially, the basis for making decision D is the expected utility function EU(D| ε) over the options encoded by D.

Assume X_i = x is observed prior to making decision D. The revised basis for making decision D is the now expected utility function EU(D| ε,x):

$$EU(D|X_i = x, \varepsilon)$$

Prior to observing the state of X_i the probability distribution of X_i is P(X_i | ε). Thus, we can compute the expected utility of the optimal decision at D after Xi is observed EUO(X_i, D| ε) to be:

$$EUO(X_i, D|e) = \sum_{X_i} P(X_i|\varepsilon) \max_D EU(D|X_i, \varepsilon)$$

This number should be compared with the maximum expected utility of the optimal decision at D without the observation on Xi. The value VOI(Xi,D| ε) of observing X_i before decision D is:

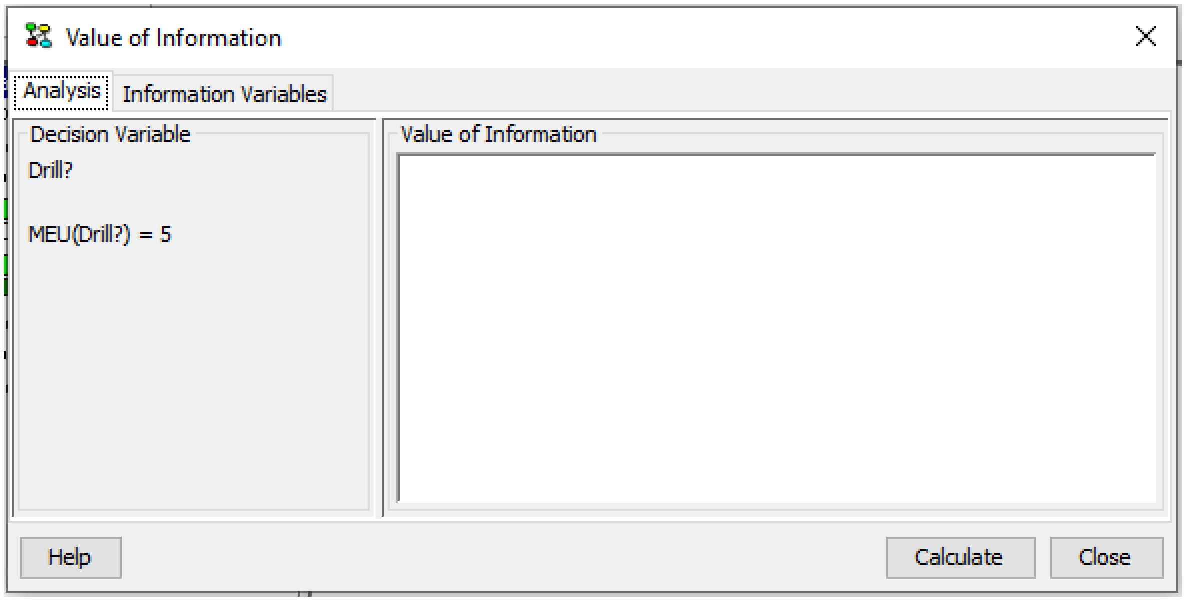
$$VOI(X_i, D|\varepsilon) = EUO(X_i|\varepsilon) \max_D EU(D|X_i, \varepsilon)$$

Performing Value of Information Analysis

Value of information analysis on a selected decision variable can be activated from either the *Network Menu* (page 185), *Node List Pane* (page 243), or by clicking the right mouse button. Notice that the decision variable must be selected in order to activate value of information analysis.

Value of information analysis is supported for discrete chance variables relative to discrete chance variables.

Figure 1 shows the value of information pane appearing after activating value of information analysis on *Drill*. In this pane the results of the value information analysis is shown (Figure 3) and the set of information variables can be defined (Figure 2).



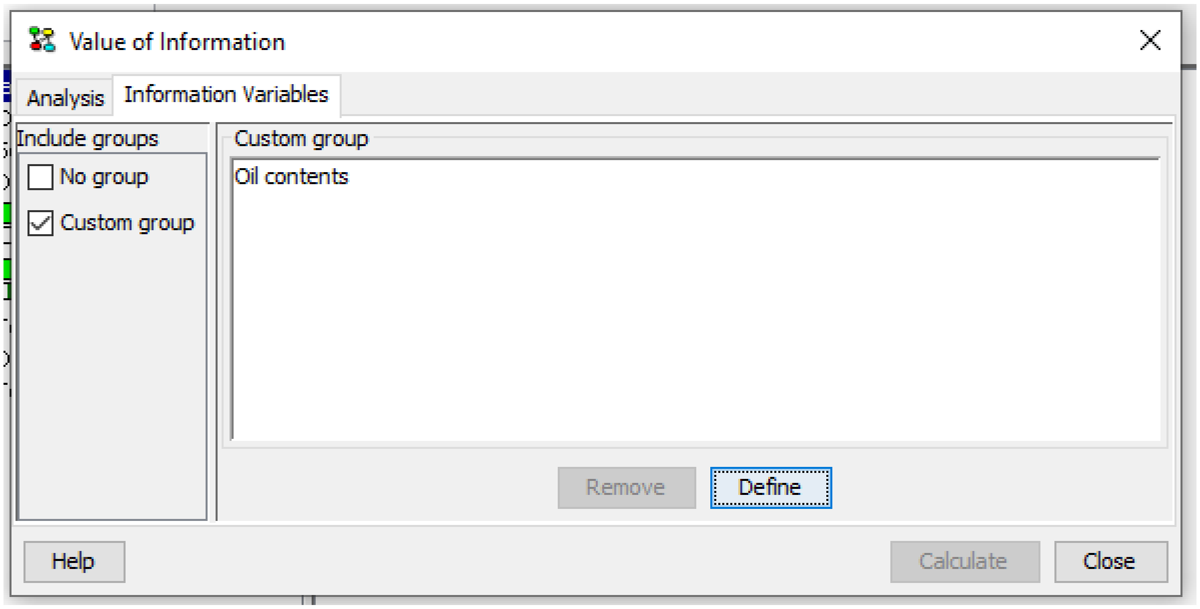
Drill?

MEU(Drill?) = 5

Figure 1: The maximum expected utility of the decision variable Drill is MEU(Drill)=5.

Figure 1 shows that the maximum expected utility of the decision variable Drill is MEU(Drill)=5.

Value of information analysis on a decision variable is performed relative to a set of information variables. The set of information variables can be selected as indicated in Figure 2.



Include groups

☐ No group

☑ Custom group

Custom group

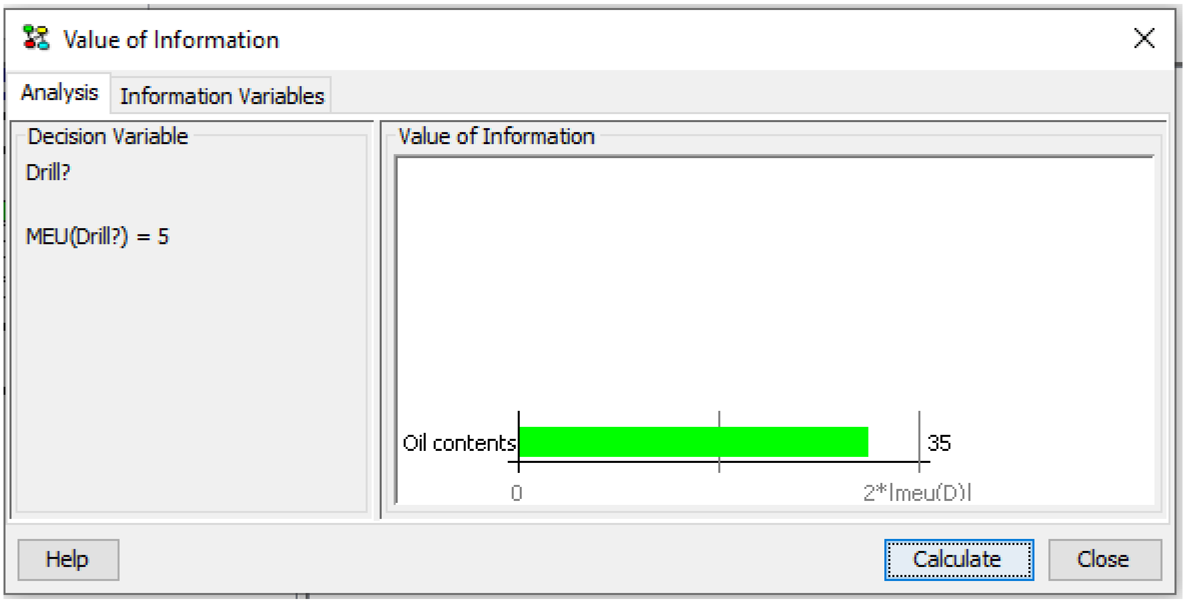
Oil contents

Figure 2: Selecting the set of information variables.

Selecting information variables proceeds in the same way as selecting *Target(s) of Instantiations* in the *d-Separation pane* (page 288).

For the custom group we restrict the set of possible information variables not to include descendants of the decision, descendants of subsequent decisions, informational parents of the decision, or informational parents of preceding decisions.

After selecting the set of information variables the value of information analysis can be performed by pressing Calculate. The results for the example are shown below in Figure 3.



Decision Variable

Drill?

MEU(Drill?) = 5

Figure 3: The results of value of information analysis on Drill relative to the selected set of information variables.

The results show the value of information of each of the selected information variables relative to the decision. There is one bar for each information variable. The name of the information variable and the value of information of the information variable relative to the decision is associated with each bar. The size of the bar is proportional to a multiplum of the maximum expected utility of the decision.

The value displayed for each observation node is the difference between the maximum expected utility of the decision node with and without the node observed before the decision.

Bayesian Networks

Consider the situation where a decision maker has to make a decision based on the probability distribution of a hypothesis variable. It could, for instance, be a physician deciding on a treatment of a patient given the probability distribution of a disease variable. For instance, if the probability of the patient suffering from the disease is above a certain threshold, then the patient should be treated immediately. Prior to deciding on a treatment the physician may have the option to gather additional information about the patient such as performing a test or asking a certain question. Given a range of options, which option should the physician choose next? That is, which of the given options will produce the most information? These questions can be answered by a value of information analysis.

Given a Bayesian network model and a hypothesis variable, the task is to identify the variable, which is most informative with respect to the hypothesis variable.

We consider myopic hypothesis driven value of information analysis on discrete random variables relative to discrete random variables. The functionality is enabled when a discrete chance node is selected or when pressing the right mouse button on a discrete chance node in the *Node List Pane* (page 243).

Entropy and Mutual Information

The main reason for acquiring additional information is to decrease the uncertainty about the hypothesis under consideration. The selection of the variable to observe next (e.g. the question to ask next) can be based on the notion of entropy. Entropy is a measure of how much probability mass is scattered around on the states of a variable (the degree of chaos in the distribution of the variable). Entropy is a measure of randomness. The more random a variable is, the higher its entropy will be.

Let X be a discrete random variable with n states x1,…,xn and probability distribution P(X), then the entropy of X is defined as:

$$H(X) = -\mathrm{E}_{P(X)}[logP(X)] = -\sum_{x} P(X)logP(X) \geq 0$$

The maximum entropy, log(n), is achieved when the probability distribution, P(X), is uniform while the minimum entropy, 0, is achieved when all the probability mass is located on a single state. Thus, the value of H(X) is in the range [0,log(n)].

Since entropy can be used as a measure of the uncertainty in the distribution of a variable, we can determine how the entropy of a variable changes as observations are made. In particular, we can identify the most informative observation.

If Y is a random variable, then the entropy of X given an observation on Y is:

$$H(X|Y) = -\mathrm{E}_{P(X,Y)}[logP(X|Y)] = -\sum_{Y} P(Y)\sum_{X} P(X|Y)logP(X|Y) = H(X) - I(X|Y)$$

where I(X,Y) is the mutual information (also known as cross entropy) of X and Y. The conditional entropy H(X| Y) is a measure of the uncertainty of X given an observation on Y, while the mutual information I(X,Y) is a measure of the information shared by X and Y (i.e. the reduction in entropy from observing Y). If X is the variable of interest, then I(X,Y) is a measure of the value of observing Y. The mutual information is computed as:

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = \sum_{Y} P(Y)\sum_{X} P(X|Y)log\frac{P(X,Y)}{P(X)P(Y)}$$

In principle, I(X,Y) is a measure of the distance between P(X)P(Y) and P(X,Y). The conditional mutual information given a set of evidence ε is computed by conditioning the probability distributions on the available evidence ε:

$$I(X,Y|\varepsilon) = \sum_{Y} P(Y|\varepsilon) \sum_{X} P(X|Y,\varepsilon) log \frac{P(X,Y|\varepsilon)}{P(X|\varepsilon)P(Y|\varepsilon)}$$

Thus, we compute I(X,Y) for each possible observation Y. The next variable to observe is selected as the variable Y, which has the highest non-zero mutual information with X, i.e. I(X,Y), if any.

Value of Information Analysis

Value of information analysis is the task of identifying the values of pieces of information. When considering hypothesis driven value of information analysis in Bayesian networks, we need to define a value function in order to determine the value of an information scenario. Entropy can be used as a value function.

In a hypothesis driven value of information analysis the value of an information scenario is defined in terms of the probability distribution of the hypothesis variable. If T is the hypothesis variable, then the value function is defined as:

$$V(T) = -H(T) = \sum_{T} P(T) log(P(T))$$

The reason for using the negation of the entropy is best illustrated using an example. Consider a binary hypothesis variable T with states false and true. Hence, the distribution of T is fully specified by a single parameter p, i.e. P(T)=(*false,true*)=(p,1-p). Figure 4 illustrates the entropy as a function of p while Figure 5 illustrates the negation of the entropy as a function of p.

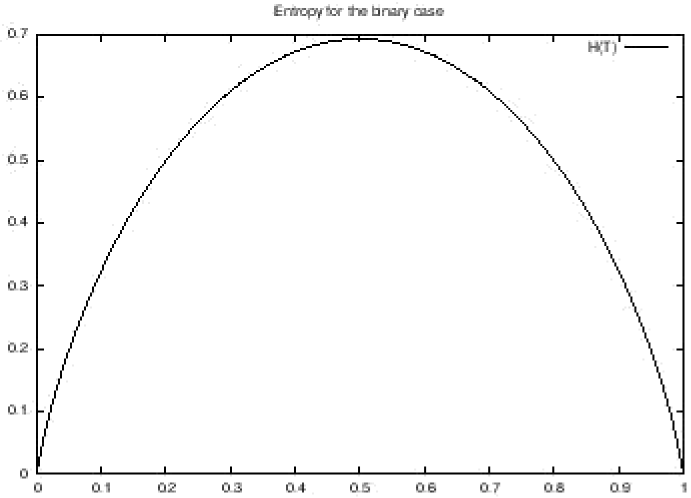


Figure 4: The entropy of T.

As can be seen from Figure 4 the entropy takes on its maximum value for the uniform distribution and its minimum value for the extreme cases (p=0 and p=1). Since the value function should take on its maximum value at the extreme cases and the minimum value in the uniform case, the negation of the entropy is used as the value function as illustrated in Figure 5.

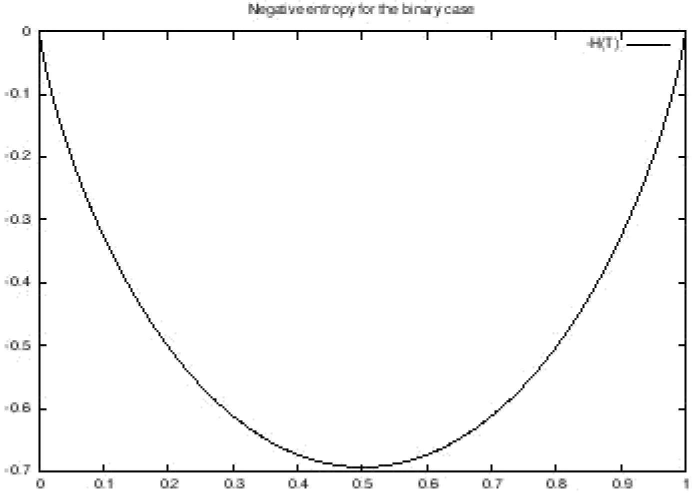


Figure 5: The negation of the entropy of T.

The value of the information scenario after observing a variable X is:

$$V(T|X) = -(H(T) - I(X,T))$$

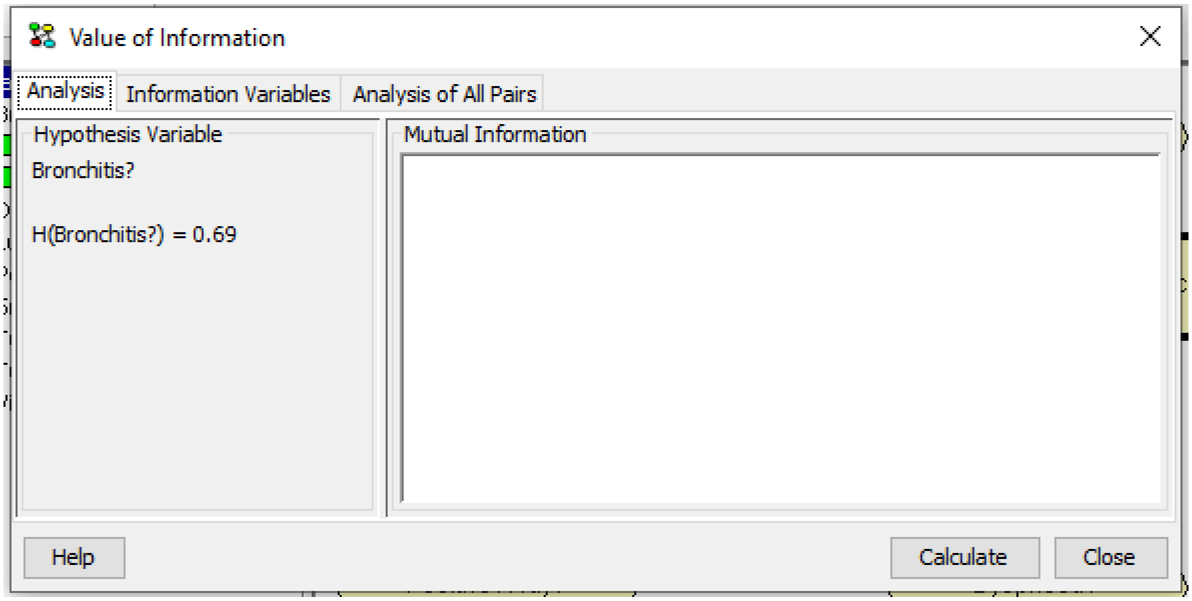
Thus, myopic hypothesis driven value of information analysis in Bayesian networks amounts to computing the value of the initial information scenario V(H) and the value of information scenarios where a variable X is observed, i.e. V(H| X). The task is to identify the variable, which increase the value of information the most. The most informative variable to observe is the variable with the highest mutual information with the hypothesis variable.

Performing Value of Information Analysis

Value of information analysis on a selected hypothesis variable can be activated from either the *Network Menu* (page 185), *Node List Pane* (page 243), or by clicking the right mouse button. Notice that the hypothesis variable must be selected in order to activate value of information analysis.

Value of information analysis is supported for discrete chance variables relative to discrete chance variables.

Figure 6 shows the value of information pane appearing after activating value of information analysis on B. In this pane the results of the value information analysis is shown (Figure 8) and the set of information variables can be defined (Figure 7).



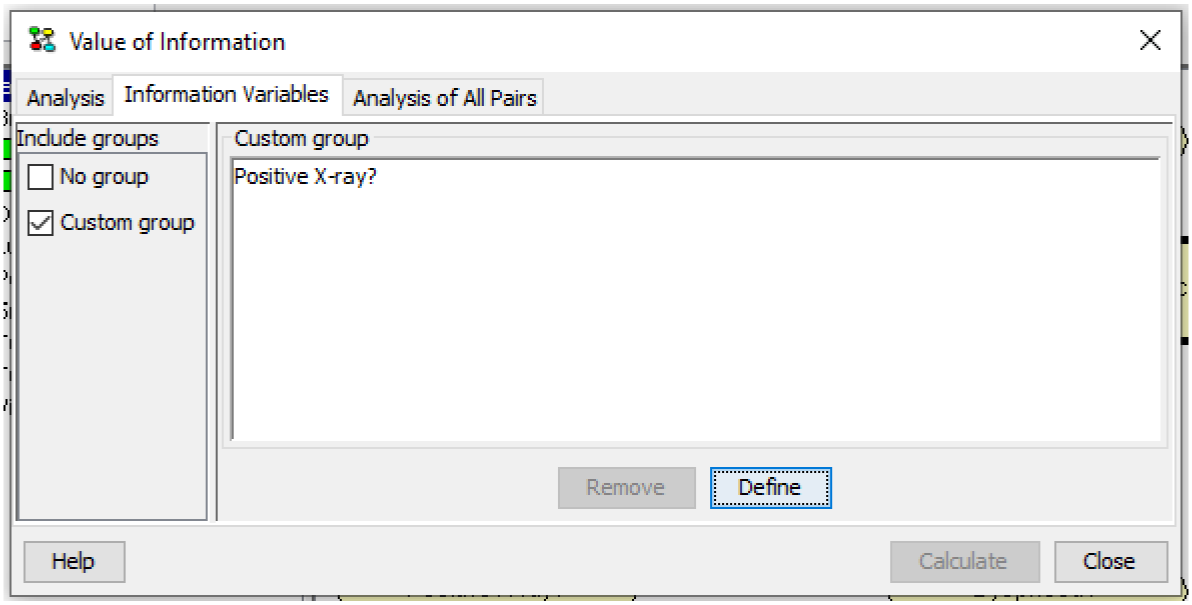
Hypothesis Variable: Bronchitis?

H(Bronchitis?) = 0.69

Figure 6: The entropy of the hypothesis variable B is H(B)=0.69.

Figure 6 shows that the entropy of B is H(B)=0.69.

Value of information analysis on a hypothesis variable is performed relative to a set of information variables. The set of information variables can be selected as indicated in Figure 7.



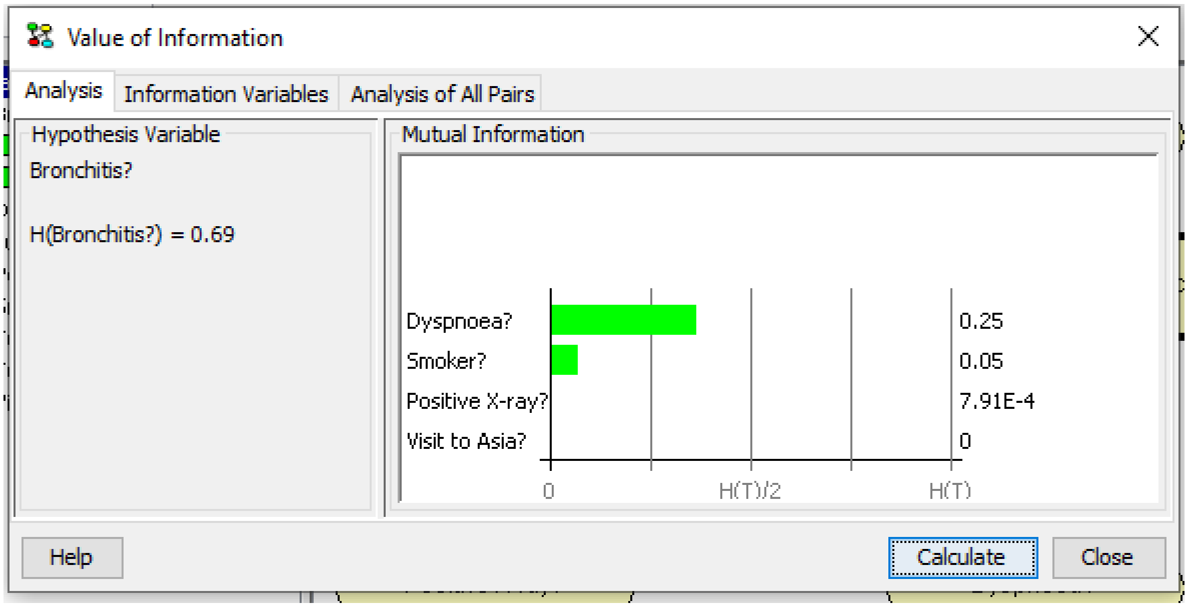
Include groups:
☐ No group
☑ Custom group

Custom group: Positive X-ray?

Figure 7: Selecting the set of information variables.

Selecting information variables proceeds in the same way as selecting Target(s) of Instantiations in the d-Separation pane.

After selecting the set of information variables the value of information analysis can be performed by pressing Calculate. The results for the example are shown below in Figure 8.



Hypothesis Variable
Bronchitis?

H(Bronchitis?) = 0.69

Figure 8: The results of value of information analysis on B relative to the selected set of information variables.

The results show the mutual information I(T,H) between the target node and each of the selected information variables. There is one bar for each information variable. The name of the information variable and the mutual information between the target node and the information variable are associated with each bar. The size of the bar is proportional to the ratio between the mutual information and the entropy of the target node.

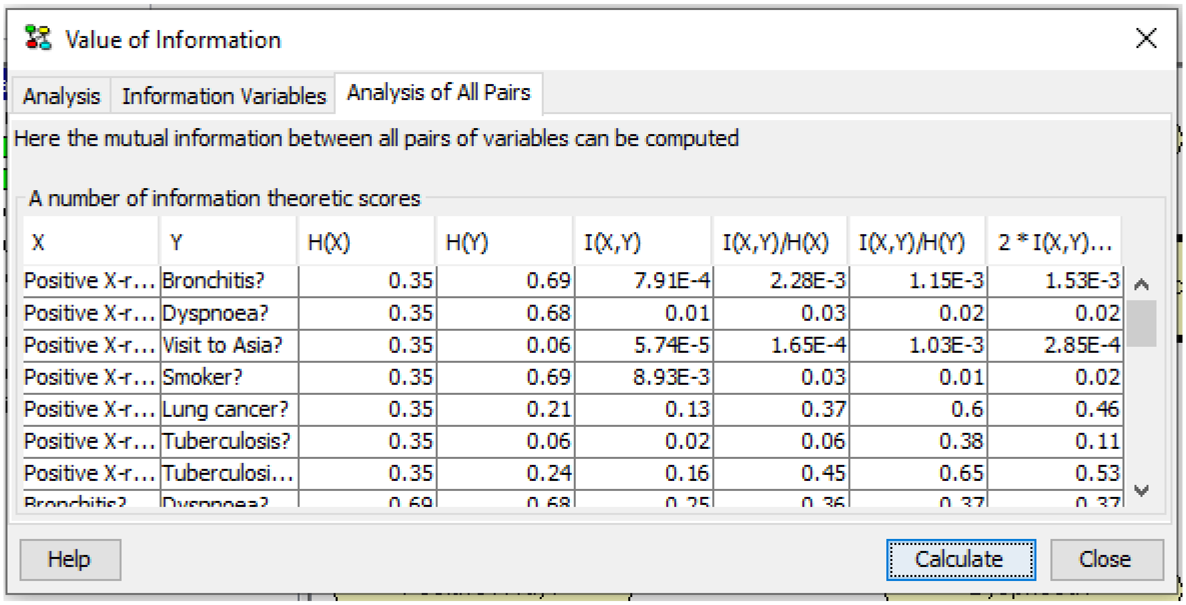
The examples above consider the case of discrete observation nodes. For possible observations on continuous chance nodes an approximation is used. Instead of using the true mixture of Normal distributions for a continuous node, a single Normal distribution with the same mean and variance is used as an approximation.

Mutual Information for All Pairs

In order to compare the strengths of dependency between different pairs of variables, a normalized variant of the mutual information can be used:

$$U(X|Y) = \frac{2l(X,Y)}{H(X)+H(Y)}$$

Notice that this measure is symmetric in X and Y. After selecting the set of information variables the value of information analysis can be performed by pressing Calculate. The results for the example are shown below in Figure 9.



Here the mutual information between all pairs of variables can be computed

A number of information theoretic scores

X	Y	H(X)	H(Y)	I(X,Y)	I(X,Y)/H(X)	I(X,Y)/H(Y)	2 * I(X,Y)...
Positive X-r...	Bronchitis?	0.35	0.69	7.91E-4	2.28E-3	1.15E-3	1.53E-3
Positive X-r...	Dyspnoea?	0.35	0.68	0.01	0.03	0.02	0.02
Positive X-r...	Visit to Asia?	0.35	0.06	5.74E-5	1.65E-4	1.03E-3	2.85E-4
Positive X-r...	Smoker?	0.35	0.69	8.93E-3	0.03	0.01	0.02
Positive X-r...	Lung cancer?	0.35	0.21	0.13	0.37	0.6	0.46
Positive X-r...	Tuberculosis?	0.35	0.06	0.02	0.06	0.38	0.11
Positive X-r...	Tuberculosi...	0.35	0.24	0.16	0.45	0.65	0.53
Bronchitis?	Dyspnoea?	0.69	0.68	0.25	0.36	0.37	0.37

Figure 9: the results of an Analysis of All Paris calculation.

4.6.11 Continuous-Discrete Node Value Transfer Function

The Continuous-Discrete node Value Transfer function (CDVT function) is available in Analysis Menu under the Network Menu. In essence the CDVT function implements a modelling technique to circumvent the limitation of no discrete children of continuous nodes in CLG Bayesian networks. A discrete interval node is linked to a continuous node such that the expression generating the distribution of the discrete node is equal to the posterior marginal of the parent continuous node. In this way the discrete node becomes a discretize representation of the continuous node. The discrete node has no table as the table is regenerated using the Table Generator functionality. The states of the discrete node are determined by the user prior to compiling the network.

Figure 1: The dashed arrow specifies a CVDT function link. Figure 1 shows a link between a continuous and discrete nodes pair created by the CDVT function.

Figure 2: The Discrete Score is a discrete approximation of the Continuous Score computed using Table Generator functionality. Figure 2 illustrates how the density function of a continuous node can be approximated by a discrete distribution using the CDVT function.

The CDVT function has some major limitations such as evidence can only be propagated along the direction of the CDVT link, only sum-normal equilibrium is supported and no type of analysis is supported on a network with CDVT links.

The discretization link should be used with extreme care.

4.6.12 Convolution of Frequency-Impact Distributions

The convolution dialog allows the user to compute the convolution of frequency-impact distributions where the convolution is computed as a weighted sum of independent variables distributed as impact with the number of terms in the sum defined by frequency. The frequency node represents the probability distribution over the number of events.

What is Convolution?

The assumption of the convolution functionality is that the frequency node predicts the expected number of events in one time period while the impact node predicts the cost of one given event. The frequency and impact nodes may dependent of the value of other nodes. The convolution of the frequency and impact distributions is the total impact. The convolution of a frequency distribution and an impact distribution is computed as the sum of independent variables as illustrated in this formula:

$$P(F = 1) * I + P(F = 2) * (I + I) + P(F = 3) * (I + I + I) + ...$$

where F is the frequency node and the *I's* are independent random variables distributed as the impact node. The convolution is a sum of distributions where the kth term is the probability of k events multiplied with the distribution of a sum of k cost variables where each cost variable is a random variable with a distribution equal to that of *impact* and all cost variables are independent.

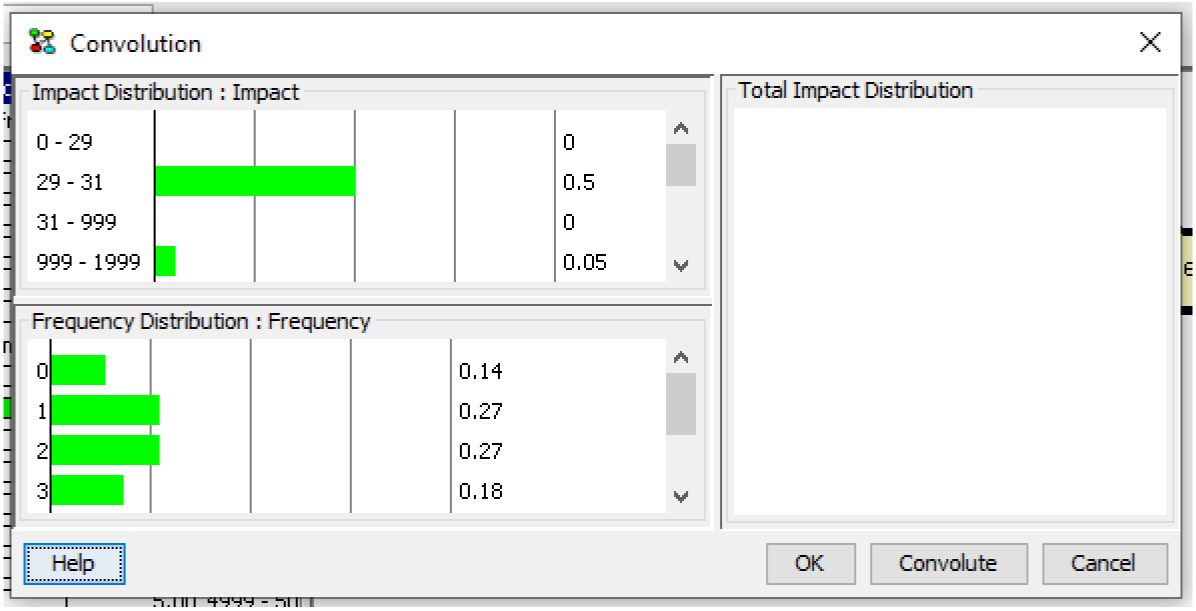
Computing a Convolution

Convolution of frequency-impact distributions is supported for a pair nodes consisting of a numbered node and an interval node. The numbered and interval nodes must be *well-formed*. The interval node is *well-formed* if its intervals cover the range from zero to infinity (and it otherwise meets the requirements for an interval node). The numbered node is *well-formed* if its state values are zero, one, two, three, e.t.c. (and it otherwise meets the requirements for a numbered node). The numbered node is interpreted as the frequency node and the interval node is interpreted as the impact node. The convolution of the frequency and impact distributions is computed with respect to the states of an interval node. This is the total impact node. The intervals of the total impact node are equal to the intervals of the impact node.

The convolution algorithm is based on the Table Generator implementation, i.e., for each interval a predefined number of values are used in the calculations (as opposed to only using the middle value of an interval).

Convolution is supported for a domain in sum-normal equilibrium.

Figure 1 shows the convolution pane appearing after activating the convolution analysis. In this pane the result of convoluting the frequency and impact distributions is shown (Figure 2). Initially, Figure 1 shows the (posterior) probability distributions of the frequency node and the impact node.

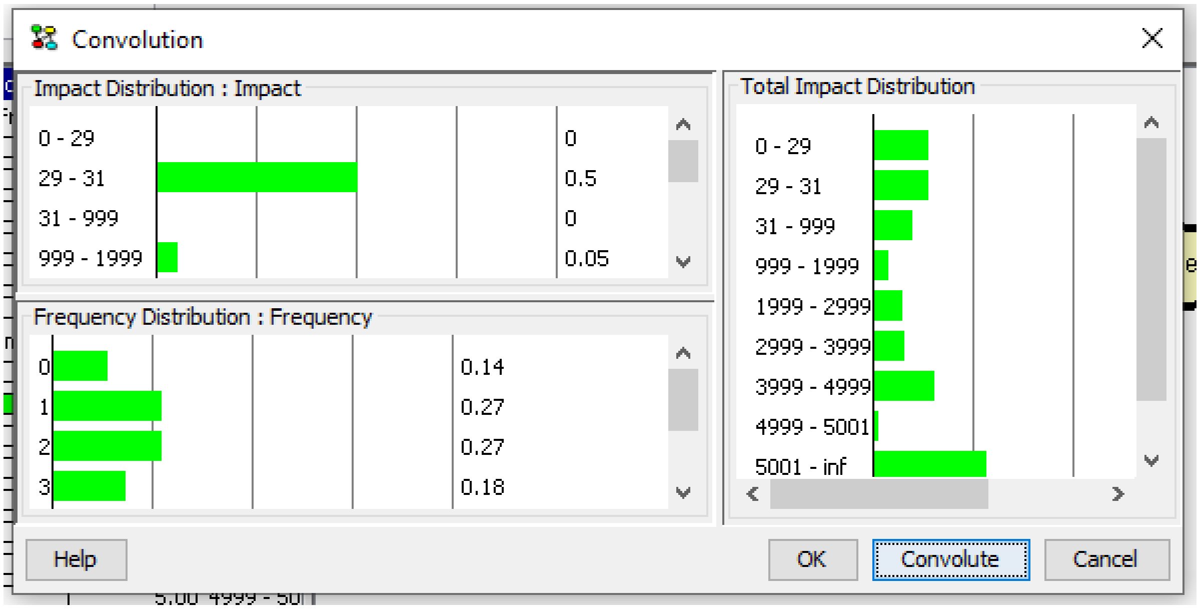

Convolution
Impact Distribution : Impact
Frequency Distribution : Frequency
Total Impact Distribution
Help
OK
Convolute
Cancel

Impact Distribution : Impact	
0 - 29	0
29 - 31	0.5
31 - 999	0
999 - 1999	0.05

Frequency Distribution : Frequency	
0	0.14
1	0.27
2	0.27
3	0.18

Figure 1: The dialog shows the distributions of the frequency node and the impact node.

The total impact distribution (i.e., the convolution of frequency and impact) is computed once the Convolute button is depressed.



Impact Distribution : Impact	
0 - 29	0
29 - 31	0.5
31 - 999	0
999 - 1999	0.05

Frequency Distribution : Frequency	
0	0.14
1	0.27
2	0.27
3	0.18

Figure 2: The total impact distribution.

Figure 2 shows the total impact distribution. The total impact distribution is computed as the convolution of the frequency and impact distributions. The convolution is computed over the intervals of the impact node.

4.6.13 The Exposure, Occurrence and Impact Loss Model

The XOI Loss Model dialog allows the user to compute a loss distribution based on the eXposure, Occurrence, and Impact Loss Model (XOI Loss Model). A XOI loss distribution can be computed by either MC simulation or by convolution of impact and exposure/frequency distribution.

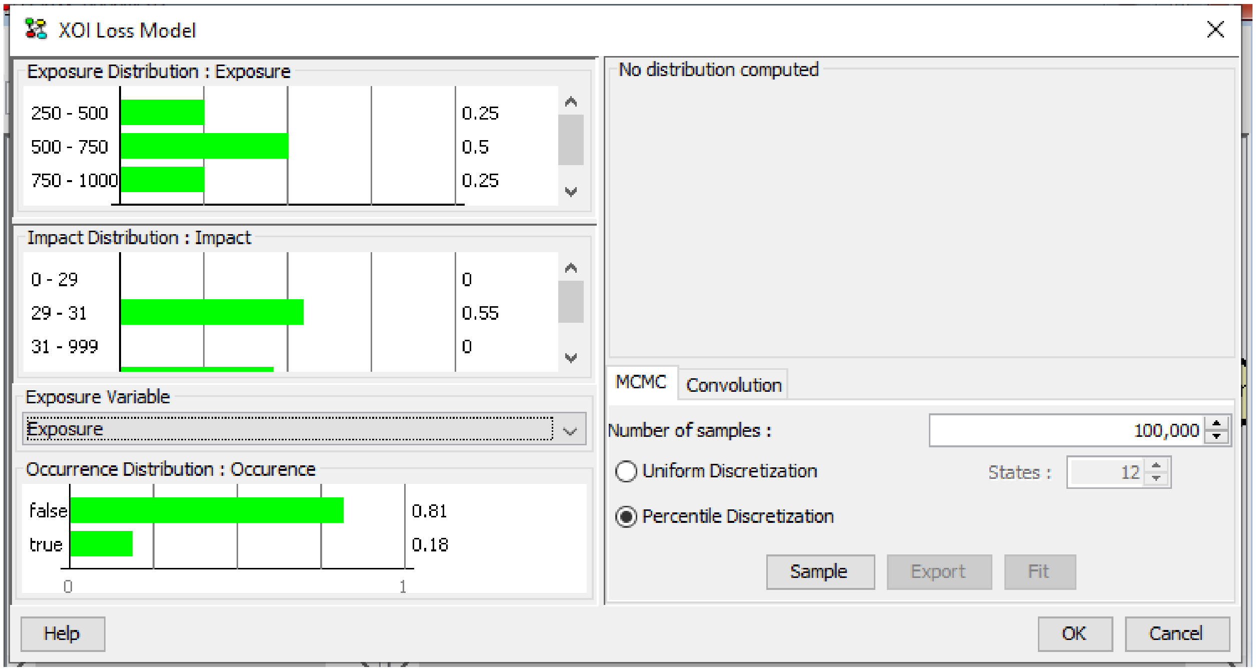
What is the XOI Loss Model?

The XOI Loss Model computes a total impact distribution based on exposure, occurrence, and impact.

- Exposure is a quantitative specification of the number of independent objects at risk.
- Occurrence is a Boolean specification of the occurrence of an event on one exposed object during one period.
- Impact is a quantiative specification of the severity of the occurrence of an event on one specific object.

The *occurrence* node is a Boolean chance node, the *impact node* is an interval node, and the *exposure node* is a numbered or interval node. Based on the states and distributions of exposure, occurrence, *impact* a total impact distribution is calculated. This distribution assesses the risk related to all exposed objects. The distribution is computed with respect to an interval node referred to as *total impact*. This node is created by HUGIN.

Figure 1 shows the XOI Loss Model dialog.



XOI Loss Model

Exposure Variable: Exposure

Number of samples : 100,000

○ Uniform Discretization — States : 12

● Percentile Discretization

Figure 1: The XOI Loss Model dialog.

The calculation of the loss distribution under the XOI Loss Model is a two step process. In the first step a (large) number of *samples* or tiny *intervals* with associated probability are generated. In the second step the samples or tiny intervals are used as the basis for identifying the intervals of *total impact*.

There are two models for generating the samples (we refer to the tiny intervals generated as samples too) of *total impact*: the MCMC model and the Convolution model. Under the MCMC model the samples are simulations of the values of *total impact* while under the Convolution model the samples are (tiny) intervals of total impact with an associated probability (i.e., *samples* are not sampled values, but correspond to a very fine-grained discretization of *total impact*.

As mentioned, we refer to these intervals as *samples*). In each case a large number of samples should be generated when a high accuracy is desired.

There are some requirements that must be fulfilled under the XOI Loss model:

- The first state of *impact* must include zero.
- The numbered and interval nodes must be *well-formed*. The interval node is *well-formed* when its intervals cover a positive range including zero (and it otherwise meets the requirements for an interval node).
- The *exposure node* can either be numbered or interval, but it must be positive, i.e., its states must specify positive values.

A few details on the implementation applying to both the MCMC and the Convolution algorithm:

- Once a value of exposure is determined, the posterior beliefs of impact and occurrence are updated, i.e., a full evidence propagation is performed. This implies that the distribution of each other node is conditioned on the value of exposure.
- If an interval has a probability mass larger than 10%, then it is not possible to perform the percentile fitting operation.
- It is important for the user to make sure that the probability associated with semi-infinite intervals are small (in order to minimize their influence on the calculations).
- When two interval nodes and one Boolean node are selected the user may have to change the choice of *exposure* (and *impact*) node.
- We do not allow evidence *exposure*, *impact*, and *occurrence*.

The MCMC Model

Under the MCMC model the loss distribution, i.e., the posterior probability distribution of *total impact*, is calculated based on Monte-Carlo simulation. A large number of samples are generated conditional on exposure and other evidence. The distribution of *total impact* is calculated as the accumulated loss.

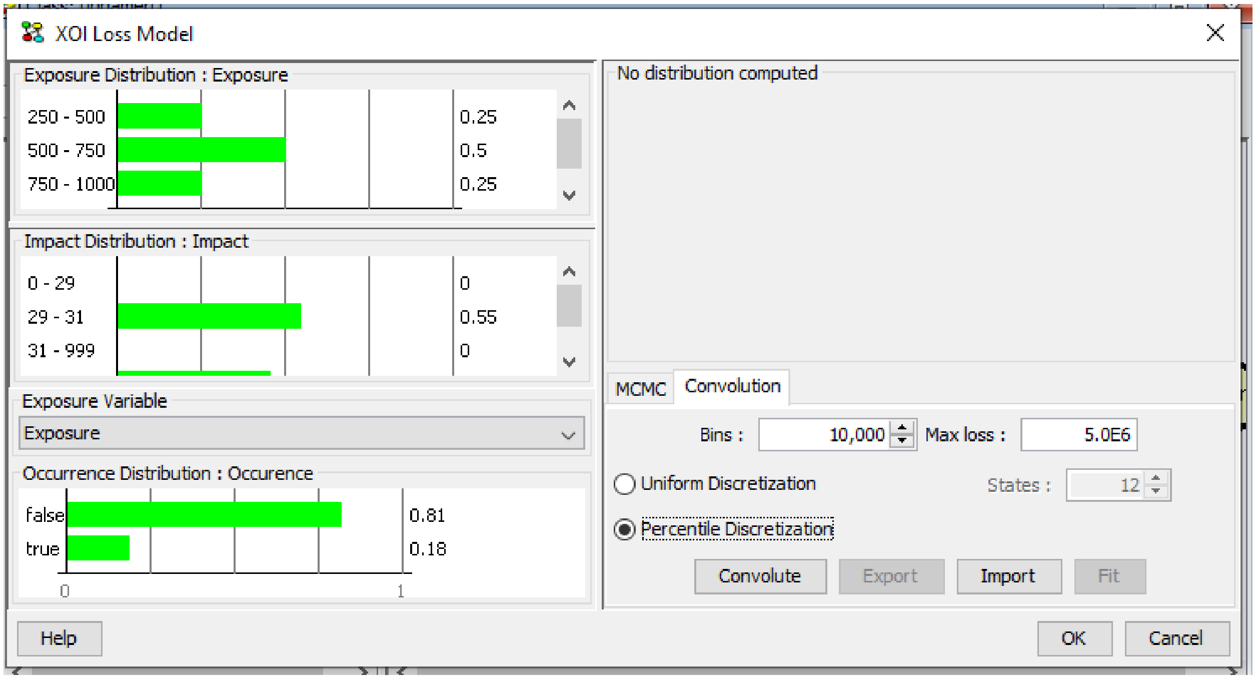
Each sample is generated as follows. First a value of exposure is sampled. Then we sample the remaining variables n times where n is the sampled value of *exposure* (holding the value of *exposure* fixed). If *occurrence* is *true*, we accumulate the value of *impact*. The *exposure* node is sampled as many times as specified by the user. Each sampled value of *total impact* requires as many samples as specified by exposure. The final accumulated value is the desired sampled result.

The Convolution Model

Under the Convolution model the loss distribution, i.e., the posterior probability distribution of *total impact*, is calculated based on the convolution of a frequency and an impact distribution.

The convolution function computes the convolution of a frequency distribution and an impact distribution. We use the convolution function to compute the distribution of *total impact* where a Binomial distribution is used as the frequency distribution. That is, the frequency distribution is simulated using a Binomial distribution with parameters n and p. The parameter n is equal to a value of exposure and the parameter p is equal to the probability of Occurrence being true. The convolution algorithm is based on the Table Generator implementation, i.e., when *exposure* is an interval node we for each interval use a predefined number of values in the calculations (as opposed to only using the middle value of an interval).

The initial number of bins is set to 10000 and the maximum loss is set to maximum exposure times maximum loss. The initial value of maximum loss may be too large when the probability of occurrence is low. When the maximum loss is too large a large number of intervals will be assigned zero probability. In this case the maximum loss can be reduced in order to obtain a more fine-grained initial discretization of *total impact*. Figure 2 shows the XOI Loss Model dialog once the Convolution tab is selected by the user.



Exposure	
250 - 500	0.25
500 - 750	0.5
750 - 1000	0.25

Impact	
0 - 29	0
29 - 31	0.55
31 - 999	0

Exposure Variable: Exposure

Occurence	
false	0.81
true	0.18

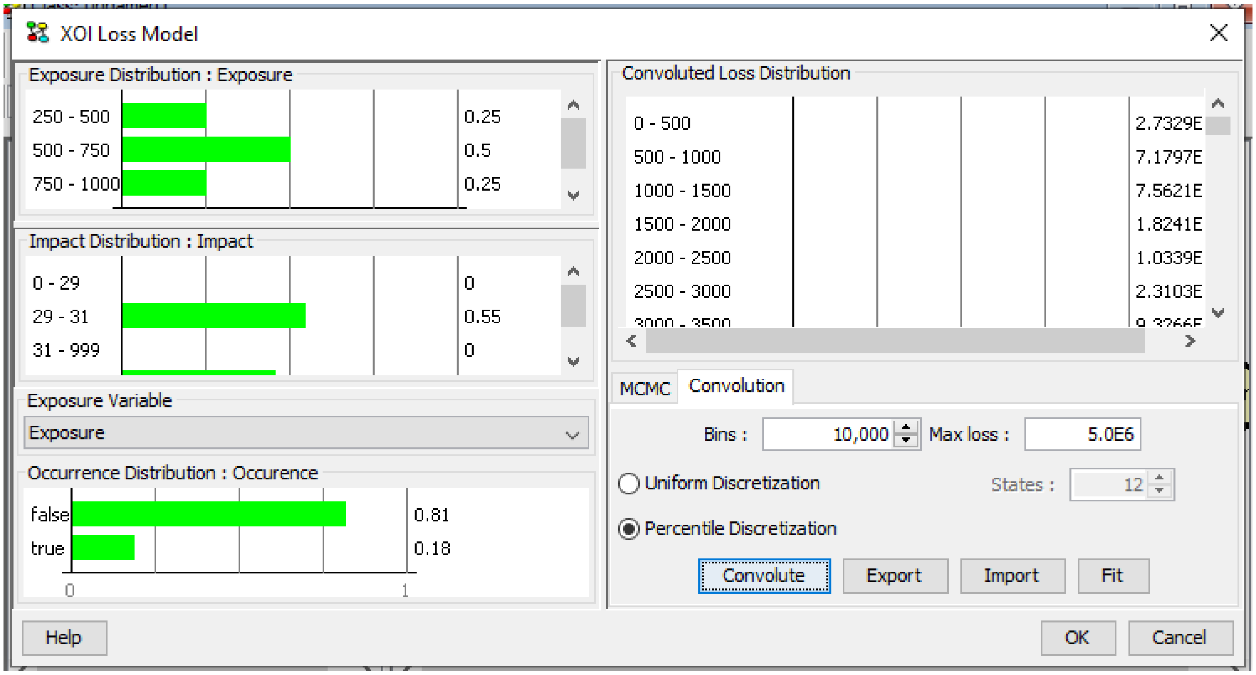
Bins : 10,000 Max loss : 5.0E6

○ Uniform Discretization States : 12

● Percentile Discretization

Figure 2: The XOI Model dialog once the *Convolution* tab is selected by the user.

Once the user selects to *convolute* the tool generates the samples for total impact. Figure 3 shows samples generated for total impact.


XOI Loss Model
Exposure Distribution : Exposure
Impact Distribution : Impact
Exposure Variable
Occurrence Distribution : Occurence
Convoluted Loss Distribution
MCMC
Convolution
Convolute
Export
Import
Fit
Help
OK
Cancel
0
1

Exposure	
250 - 500	0.25
500 - 750	0.5
750 - 1000	0.25

Impact	
0 - 29	0
29 - 31	0.55
31 - 999	0

Exposure Variable: Exposure

Occurence	
false	0.81
true	0.18

Convoluted Loss Distribution	
0 - 500	2.7329E
500 - 1000	7.1797E
1000 - 1500	7.5621E
1500 - 2000	1.8241E
2000 - 2500	1.0339E
2500 - 3000	2.3103E
3000 - 3500	9.3266E

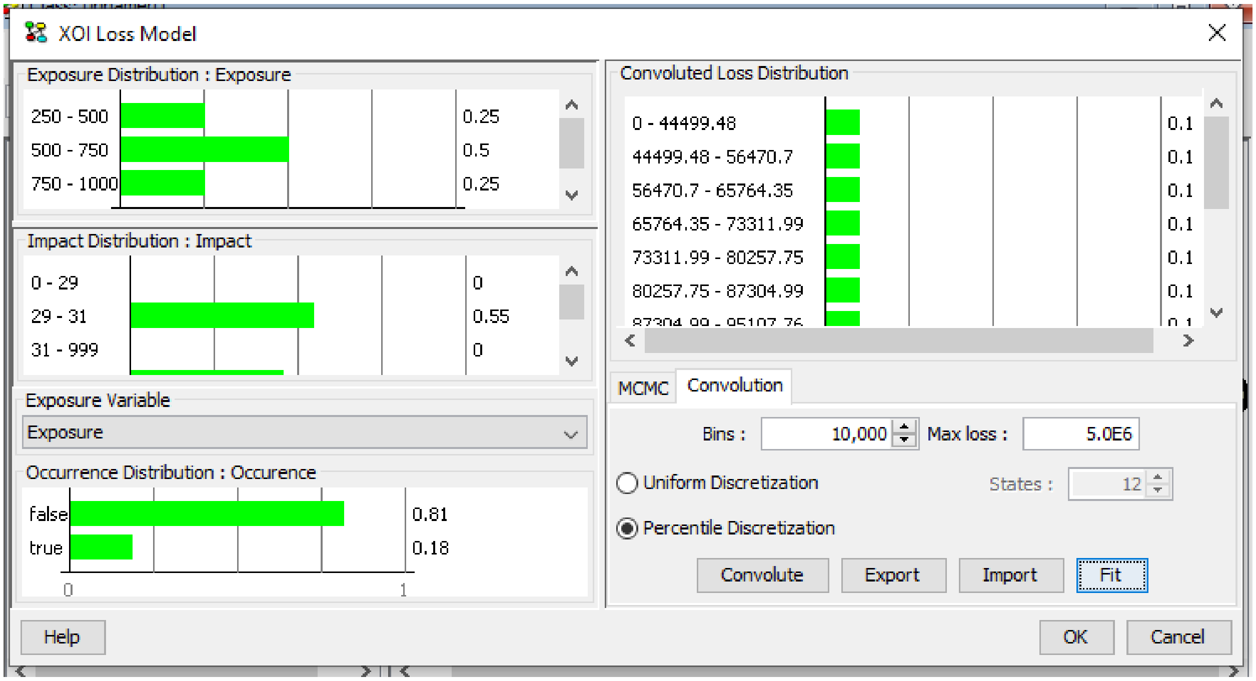
Bins : 10,000 Max loss : 5.0E6

○ Uniform Discretization States : 12

● Percentile Discretization

Figure 3: The samples generated once the user selects to convolute.

The *samples* generated for *total impact* can be *fitted* to a (significantly) reduced number of intervals. This *fitting* operation is a discretization process. The user has to select how the discretization should be performed. The user can either choose to perform a uniform discretization (and specify the number of states) or a percentile-based discretization (an example of the percentile-based discretization is show in Figure 4).



Exposure Distribution : Exposure

250 - 500	0.25
500 - 750	0.5
750 - 1000	0.25

Impact Distribution : Impact

0 - 29	0
29 - 31	0.55
31 - 999	0

Exposure Variable: Exposure

Occurrence Distribution : Occurence

false	0.81
true	0.18

Convoluted Loss Distribution

0 - 44499.48	0.1
44499.48 - 56470.7	0.1
56470.7 - 65764.35	0.1
65764.35 - 73311.99	0.1
73311.99 - 80257.75	0.1
80257.75 - 87304.99	0.1
87304.99 - 95107.76	0.1

Bins : 10,000 Max loss : 5.0E6

○ Uniform Discretization States : 12

● Percentile Discretization

Figure 4: The distribution of *total impact* as a result of convolution.

Figure 4 shows the distribution of *total impact* as a result of convolution.

4.6.14 Joint Probability Distribution

Figure 1 shows the joint analysis pane appearing after activating the joint configurations dialog. In this pane the joint probability distribution over a set of nodes is shown (Figure 3). The set of nodes in the joint can be defined by selecting the “Variables” tab (Figure 2).

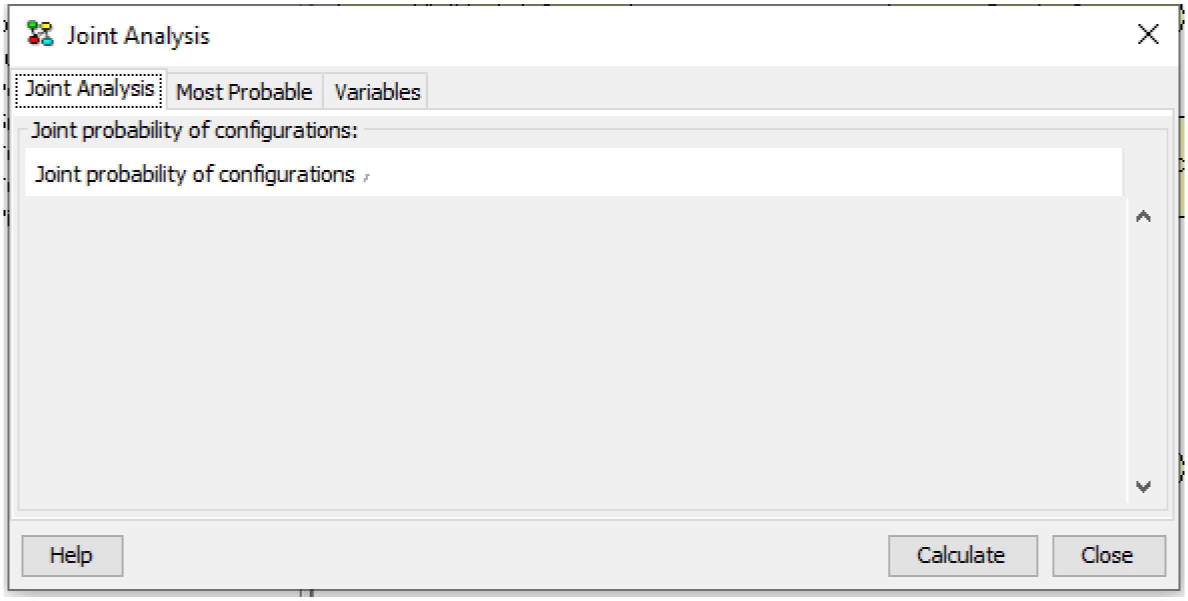
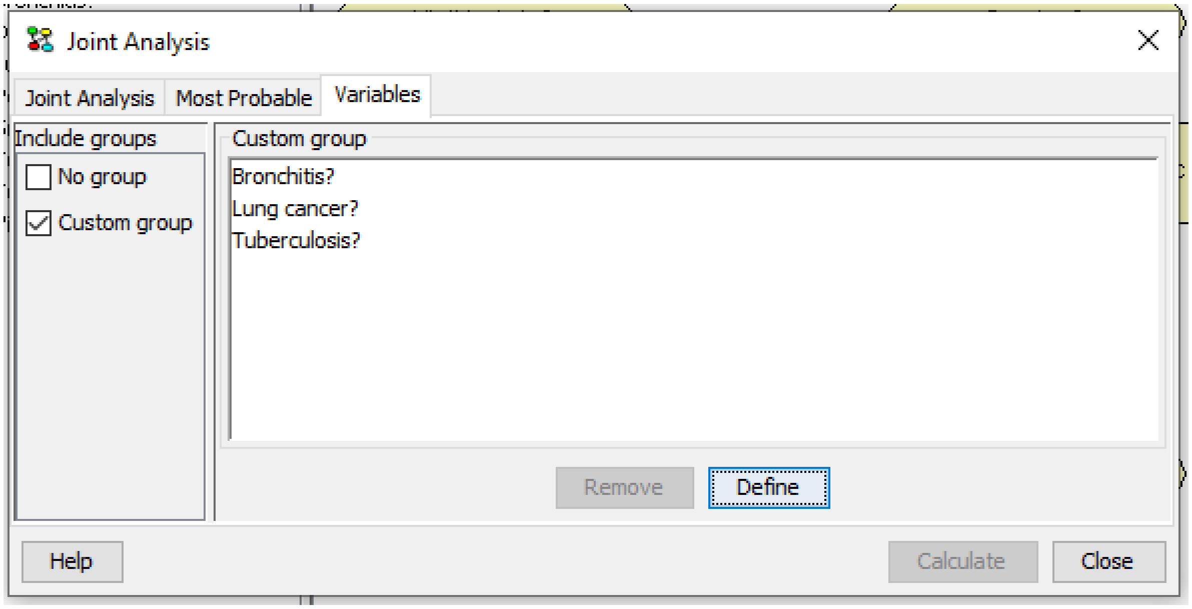

Joint Analysis
Joint Analysis
Most Probable
Variables
Joint probability of configurations:
Joint probability of configurations
Help
Calculate
Close

Figure 1: No joint distribution has been computed.

A joint probability distribution is computed over a set of nodes. The set of nodes can be selected as indicated in Figure 2.



☐ No group

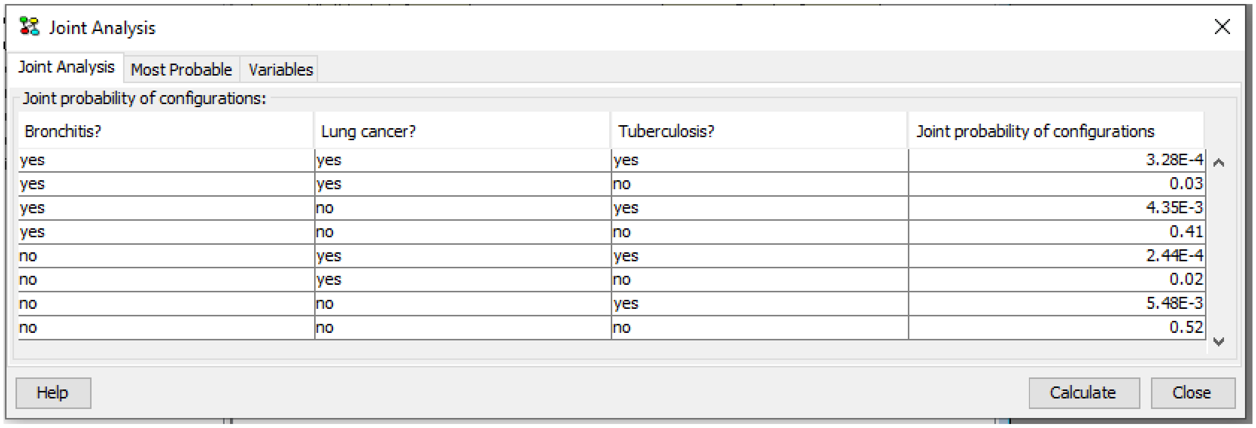
☑ Custom group

Bronchitis?
Lung cancer?
Tuberculosis?

Figure 2: Selecting the set of nodes in the joint distribution.

Selecting nodes to appear in the joint proceeds in the same way as selecting *Target(s)* of *Instantiations* in the d-Separation pane.

After selecting the set of nodes their joint probability distribution is computed by selecting *Calculate*. The results for the example are shown below in Figure 3 where the probabilities are sorted.



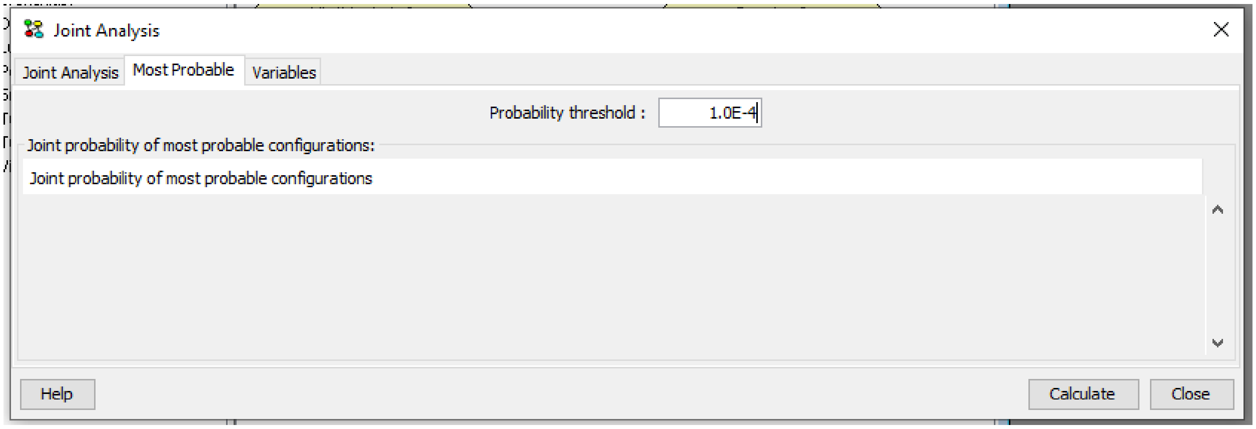
Joint probability of configurations:

Bronchitis?	Lung cancer?	Tuberculosis?	Joint probability of configurations
yes	yes	yes	3.28E-4
yes	yes	no	0.03
yes	no	yes	4.35E-3
yes	no	no	0.41
no	yes	yes	2.44E-4
no	yes	no	0.02
no	no	yes	5.48E-3
no	no	no	0.52

Figure 3: The joint distribution over the selected set of nodes.

The joint probability distribution over a set of nodes is computed by message passing in a junction tree. The feasibility of computing a joint distribution is in part determined by the size of the junction tree and the combined state space size of the nodes in the joint. The size of the joint distribution is equal to the product of the state space sizes of the nodes in the jiont. Thus, the size of the joint distribution grows exponentially with the number of nodes in the joint.

Notice that the most probable configurations can be identified by sorting the rightmost column. This do, however, require that the full joint is computed. Alternatively, the most probable configurations of a subset of nodes can be identified using the *Most probable* tab.



Probability threshold : 1.0E-4

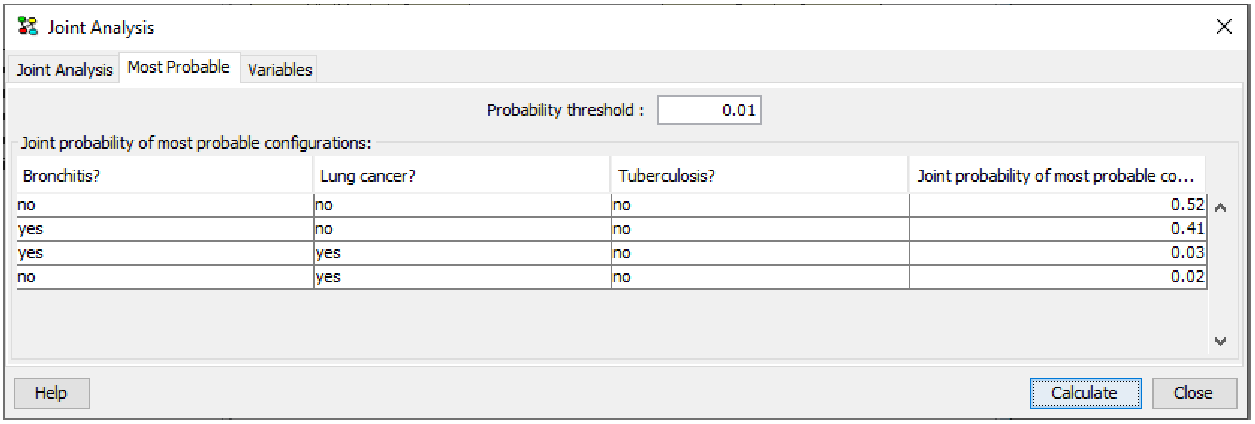
Joint probability of most probable configurations:

Joint probability of most probable configurations

Figure 4: The most probable configurations over the selected set of nodes with a probability higher than the threshold.

This dialog can be used to identify the most probable configurations over a set of nodes when their joint is too large to be represented in main memory or if the joint cannot be computed. The user is asked to supply a threshold on the probability of the configurations to be included in the result. The lower the thredhold is, the longer the algorithm has to run.

The result for the example above with a threshold of 0.01 is shown below.



Probability threshold : 0.01

Joint probability of most probable configurations:

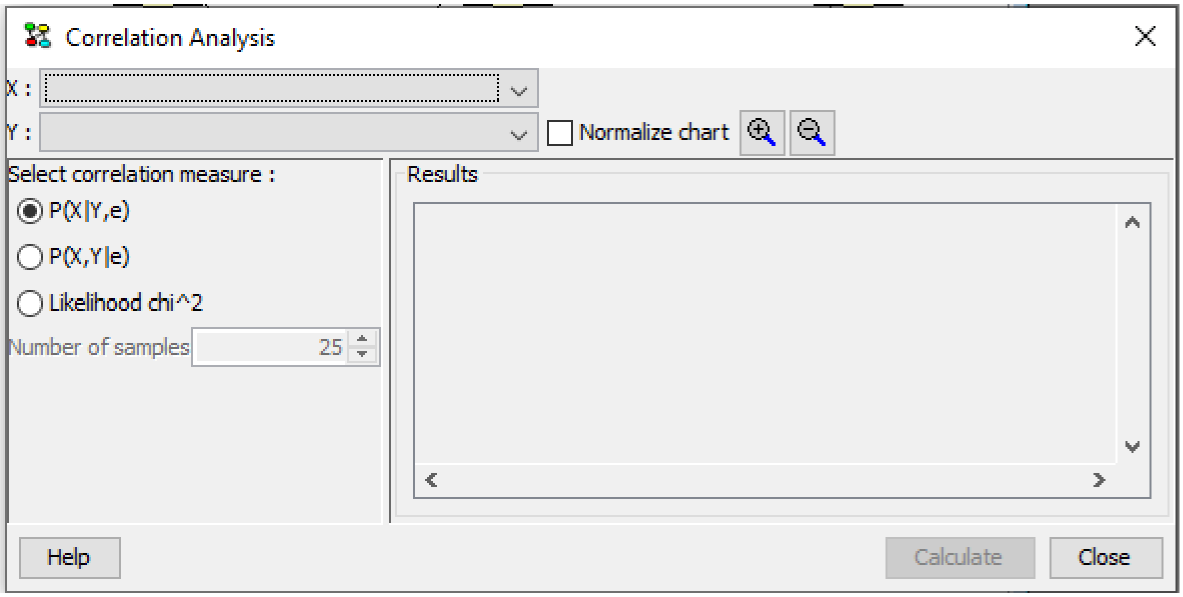
Bronchitis?	Lung cancer?	Tuberculosis?	Joint probability of most probable co...
no	no	no	0.52
yes	no	no	0.41
yes	yes	no	0.03
no	yes	no	0.02

Figure 5: The most probable configurations with a probability higher than the defined threshold.

This method is not supported for networks with decisions.

4.6.15 Correlation Analysis Dialog

Figure 1 shows the correlation analysis dialog as it appears after being activated. In dialog the correlation between a pair of discrete chance node can be analysed using a number of different measures.



□ Normalize chart

● P(X|Y,e)

○ P(X,Y|e)

○ Likelihood chi^2

Number of samples 25

Figure 1: The Correlation Analysis Dialog.

The dialog supports three different methods for analysing the correlation between a pair of nodes X and Y: the conditional probability distribution, the joint probability distribution, and the Pearson test.

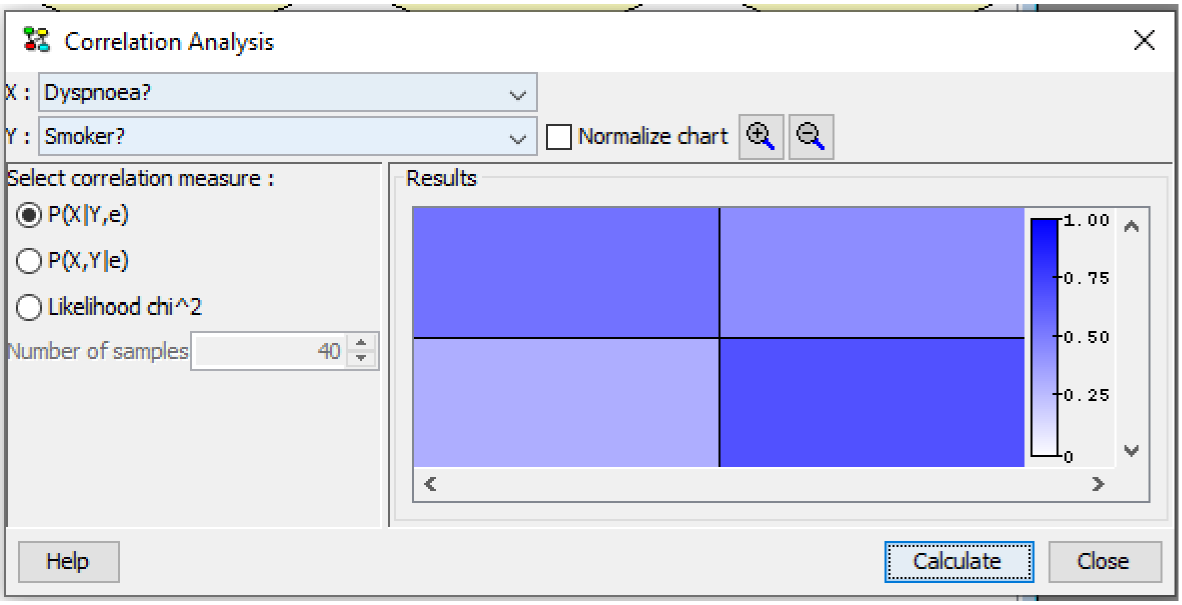
The correlation between the selected pair of nodes X and Y is illustrated using a color intensity chart. Each entry of the chart corresponds to a configuration of X and Y. The more intense the color is the higher the value displayed in the entry is. The entries of the color chart are computed using the selected correlation measure when the user press *Compute*.

If the *normalize chart* check box is checked, then the numbers displayed in the color chart are normalized such that the maximum value has the maximum color intensity. This is useful, for instance, if the values in the joint probability distribution are all much less than one. In this case the color intensity will be low. By normalizing the chart, the maximum value will have maximum color intensity.

In the following sections the three methods for analysing the correlation between a pair of discrete chance variables are described.

Conditional Probability

Figure 2 shows the conditional probability distribution of *X* given *Y* computed based on the entered and propagated evidence.



Correlation Analysis

X : Dyspnoea?

Y : Smoker?

□ Normalize chart

Select correlation measure :

● P(X|Y,e)

○ P(X,Y|e)

○ Likelihood chi^2

Number of samples 40

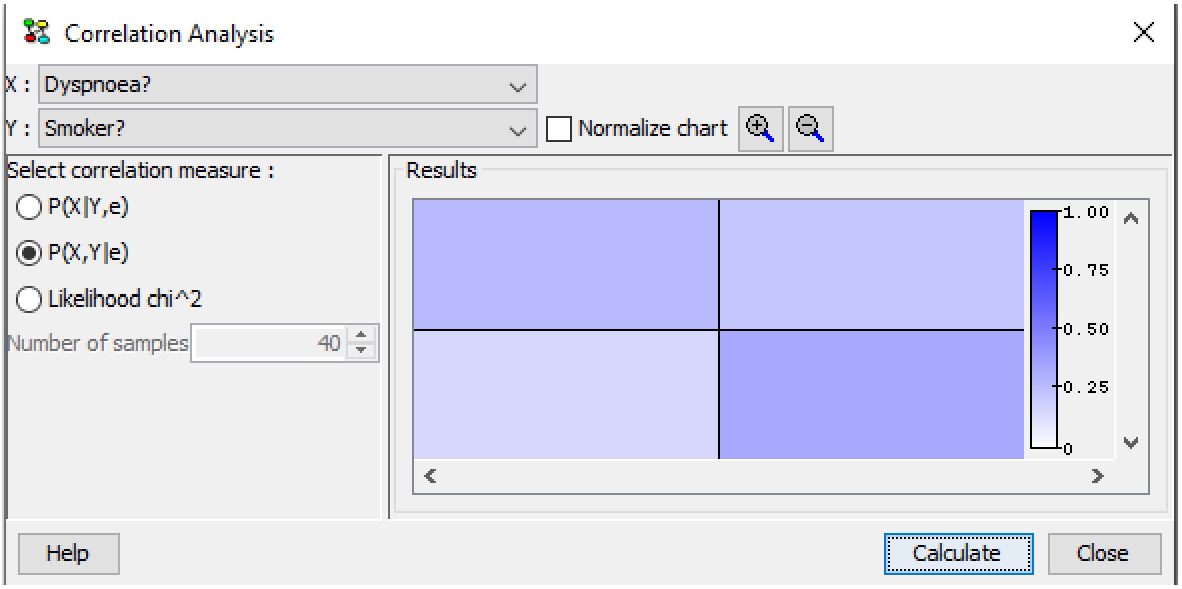
Figure 2: The conditional probability distribution of *X* given *Y*.

Notice that *X* and *Y* can in principle be any pair of discrete chance nodes in the model.

The computation of the conditional probability distribution may in principle fail with an out-of-memory error. This will happen if the tables in the underlying junction tree become too large during the process of computing the conditional.

Joint Probability Distribution

Figure 3 shows the joint probability distribution of X and Y computed based on the entered and propagated evidence.



Correlation Analysis

X : Dyspnoea?

Y : Smoker?

□ Normalize chart

Select correlation measure :

○ P(X|Y,e)

● P(X,Y|e)

○ Likelihood chi^2

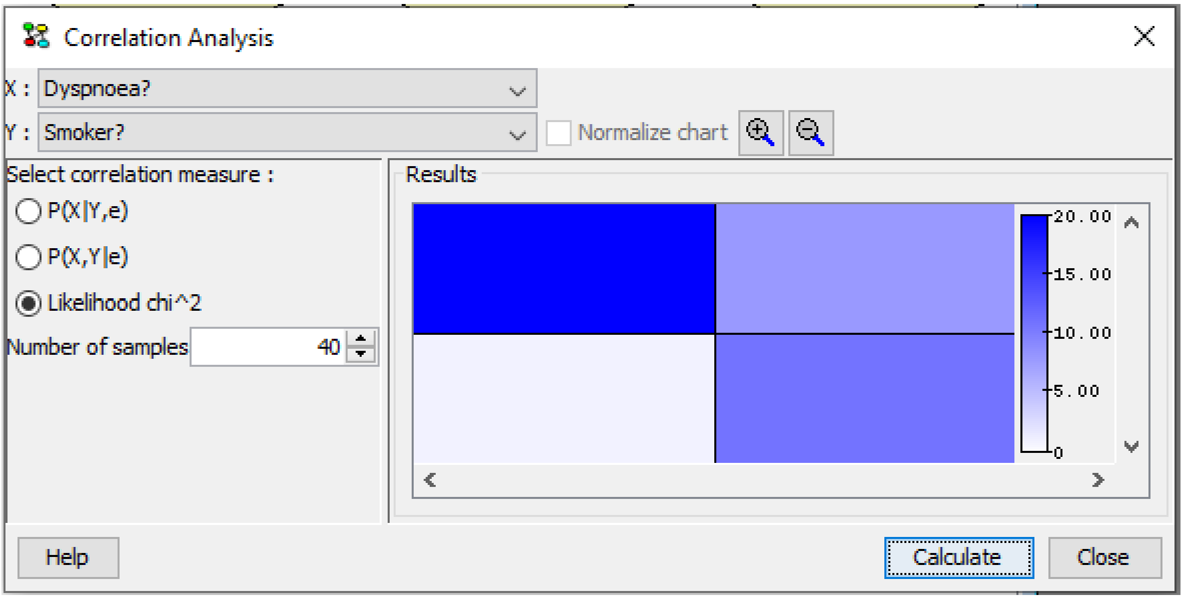
Number of samples 40

Figure 3: The joint probability distribution over X and Y.

Notice that X and Y can in principle be any pair of discrete chance nodes in the model. The computation of the joint probability distribution may in principle fail with an out-of-memory error. This will happen if the tables in the underlying junction tree become too large during the process of computing the joint.

Pearson Test

Figure 4 shows the joint counts table X and Y computed based on the entered and propagated evidence.



Correlation Analysis

X : Dyspnoea?

Y : Smoker?

□ Normalize chart

Select correlation measure :

○ P(X|Y,e)

○ P(X,Y|e)

● Likelihood chi^2

Number of samples 40

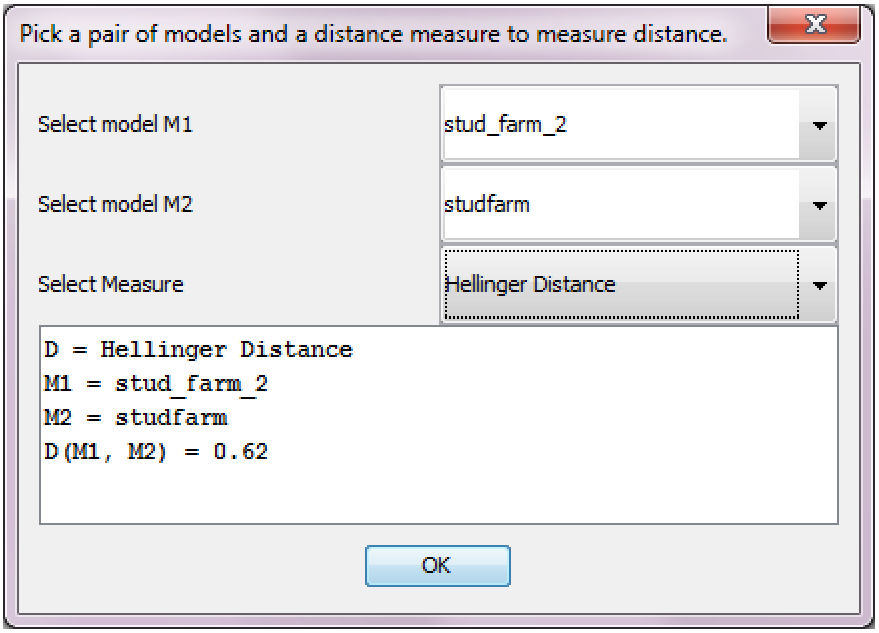
Figure 4: A table of counts over X and Y.

The table of counts is computed by generating a sample over X and Y consisting of the number of samples defined by the user. The sample is generated based on the evidence entered and propagated in the network. Based on this sample the likelihood chi^2 score is computed and displayed for each configuration of X and Y.

4.6.16 Distance Analysis Dialog

Interaction with the Dialog

Figure 1 shows the distance analysis dialog as it appears after being activated. In dialog the distance between a pair of Network Models can be measured. The models subject to analysis must be loaded before activating the dialog. Whether the model needs to be in runtime mode depends on the selected measurement method. The text area is used for output. If the dialog succeeds in calculating a measure, the configuration and result will written here.



Pick a pair of models and a distance measure to measure distance.

Select model M1: stud_farm_2

Select model M2: studfarm

Select Measure: Hellinger Distance

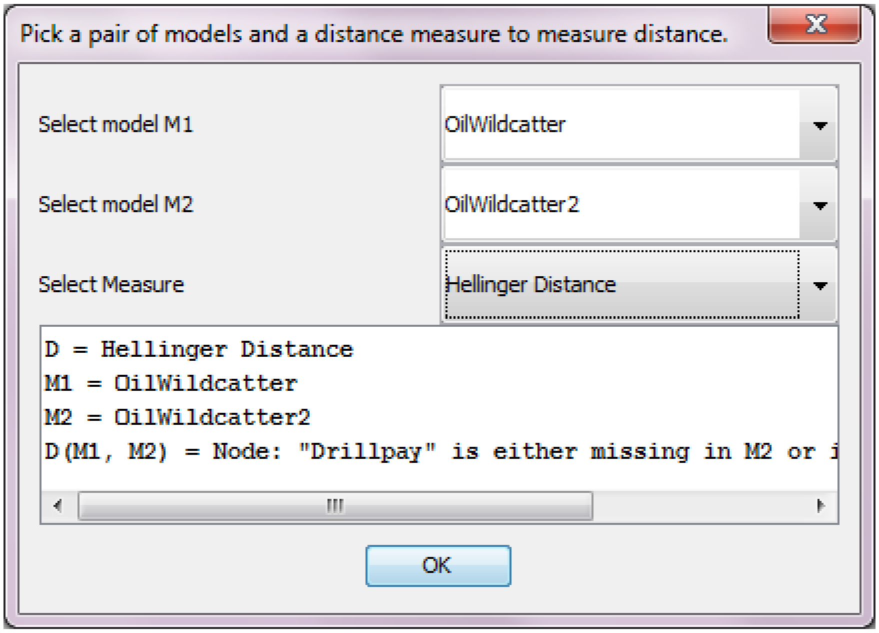
```
D = Hellinger Distance
M1 = stud_farm_2
M2 = studfarm
D(M1, M2) = 0.62
```

Figure 1: The Correlation Analysis Dialog.

Select the input parameters:

- **M1** and **M2** are the models used in the analysis. Choices are among the loaded Models.
- **Measure** is the measuring method used in the analysis. The available distance measures and additional details can be found under *Available Measures* (page 334) below.

The dialog will attempt to perform a distance calculation as soon as any input parameter has been changed. In case one of the models is not in the required mode, the dialog will inform about this inside the result text area below. Also, in case the measurement method encounters incompatible node types or any other limitation a message will also appear in the result text area.



Pick a pair of models and a distance measure to measure distance.

Select model M1: OilWildcatter

Select model M2: OilWildcatter2

Select Measure: Hellinger Distance

```
D = Hellinger Distance
M1 = OilWildcatter
M2 = OilWildcatter2
D(M1, M2) = Node: "Drillpay" is either missing in M2 or i
```

Figure 2: The out put will provide indications of any error or any encountered limitation.

Available Measures

Hellinger Distance

Requirements and Limitations

The Hellinger Distance measures the distance between 2 Bayesian Network Models: M1 and M2. Identical Variables are recognized in M1 and M2 by names of variables. Nodes with identical names in M1 and M2 must have the same number of states. Only discrete chance nodes are allowed. Hellinger Distance works for models both in edit mode and i run mode.

The Hellinger Distance between two probability distributions P and Q is defined as:

$$D_H(P,Q) = \sqrt{\sum_i \sqrt{p_i} - \sqrt{Q_i}}$$

Weighted Hellinger Distance

Requirements and Limitations

The Weighted Hellinger Distance measure has the same requirements and Limitations as *Hellinger Distance* (page 334) exept that it requires models to be in runmode.

In the Weighted Hellinger Distance: $D_H^w(P_1(X|\pi), P_2(X|\pi))$, the *Hellinger Distance* (page 334) $D_H(P_1(X|\pi), P_2(X|\pi))$ is weighted by the probability of the parent configuration $P(\pi)$.

4.6.17 Parameter Sensitivity Analysis

Parameter Sensitivity analysis is the analysis of how sensitive the results of a belief update (propagation of evidence) is to variations of the value of a parameter of the model. The parameters of a model are the entries of the conditional probability distributions.

Parameter Sensitivity Analysis

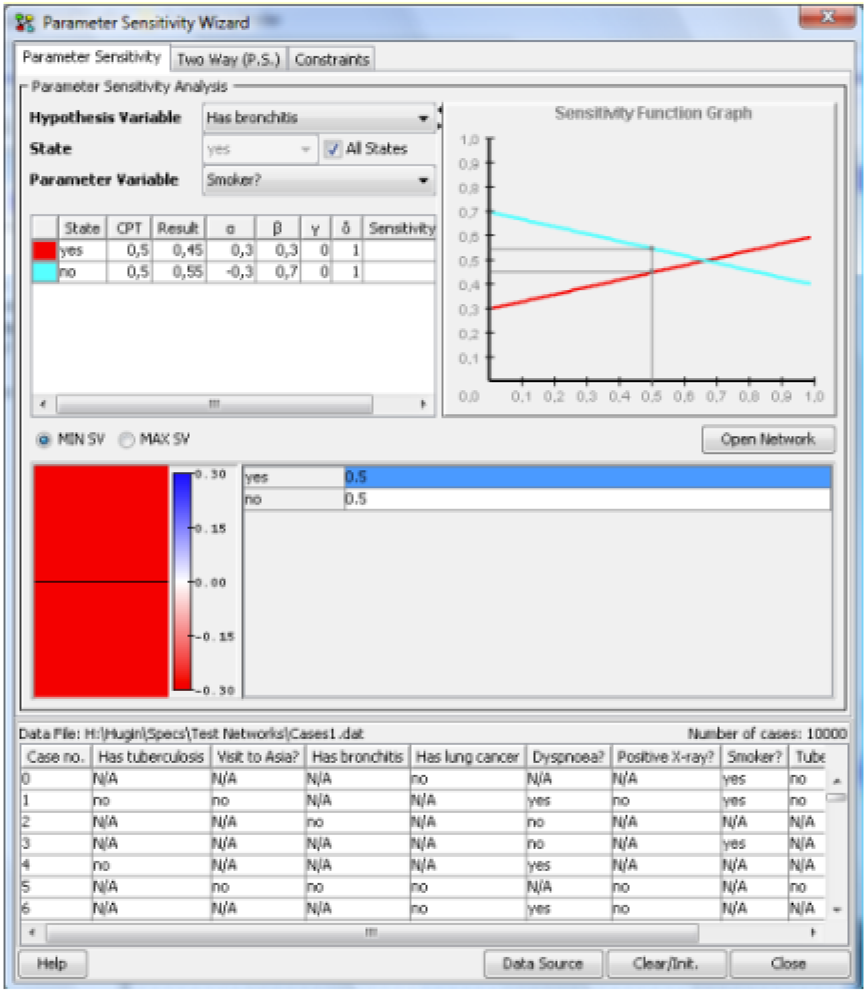
The Parameter Sensitivity panel allows us to perform sensitivity analysis on the hypothesis variable to changes on the value of the parameter variable. The sensitivity analysis is based on the sensitivity function:

$$f(t) = (\alpha * t + \beta)/(\gamma * t + \delta)$$

for computing the belief of a state of a node for a given CPT value of a parameter. The sensitivity value is computed based on the function

$$f'(t) = (\alpha * \delta - \beta * \gamma)/\gamma * t + \delta)^2$$

In the following example (Figure 1.) the “asia.net” network is loaded and compiled before entering the Parameter Sensitivity Panel (available from the “Wizards” menu in Run-mode). The “Has bronchitis” node is selected as the hypothesis variable and “All States” is checked so an analysis will be performed on all states of the Hypothesis node. The node “Smoker” is selected as the parameter variable. When selecting a parameter variable its CPT table is displayed in the middle of the panel (Figure 1).



Hypothesis Variable Has bronchitis

State yes ☑ All States

Parameter Variable Smoker?

	State	CPT	Result	α	β	γ	δ	Sensitivity
🟥	yes	0,5	0,45	0,3	0,3	0	1	
🟦	no	0,5	0,55	-0,3	0,7	0	1	

● MIN SV ○ MAX SV

yes	0.5
no	0.5

Data File: H:\Hugin\Specs\Test Networks\Cases1.dat Number of cases: 10000

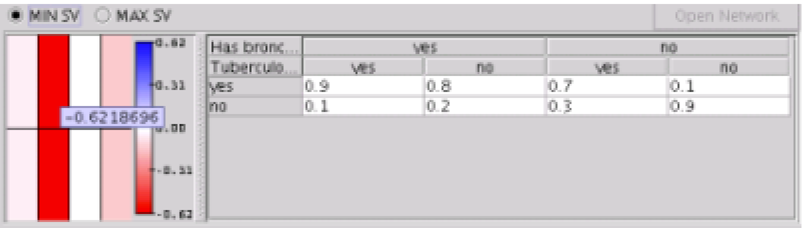
Case no.	Has tuberculosis	Visit to Asia?	Has bronchitis	Has lung cancer	Dyspnoea?	Positive X-ray?	Smoker?	Tube
0	N/A	N/A	N/A	no	N/A	N/A	yes	no
1	no	no	N/A	N/A	yes	no	yes	no
2	N/A	N/A	no	N/A	no	N/A	N/A	N/A
3	N/A	N/A	N/A	N/A	no	N/A	yes	N/A
4	no	N/A	N/A	N/A	yes	N/A	N/A	N/A
5	N/A	no	no	no	N/A	no	N/A	no
6	N/A	N/A	N/A	no	yes	no	N/A	N/A

Figure 1: Parameter Sensitivity Analysis

By clicking on yes = 0.5 in the CPT table, the sensitivity function is used to compute a value (belief) for each state of the hypothesis variable. A graph is displayed on the top right corner of the panel, showing two lines that represent how sensitive the states of the hypothesis variable are, to changes of the parameter values, i.e., the entry of the CPT selected which in the example is yes = 0.5.

From this graph we notice that the state of the hypothesis variable with highest probability changes when the parameter value is above approximately 0.7. Also, notice that the sensitivity function is a straight line. Given that "Smoker = yes = 0.5" the belief that the patient has bronchitis is 0.45 and the belief that he doesn't have bronchitis is 0.55. These points (0.5, 0.45) and (0.5, 0.55) are shown on the graph.

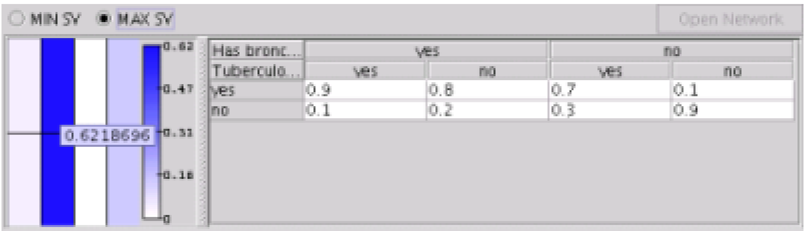
Sensitivity analysis can not be performed for extreme CPT values, i.e., zero and one. When the hypothesis node has many states, the graph can become confusing because of overlapping sensitivity function lines. By selecting a state in the graph information table (upper - left), only the sensitivity function line for the selected state will be displayed in the graph. To see all lines again, select a CPT index in the Parameter variable table. The table located in the middle of the panel (Figures 1a, 1b), is the CPT table of the node selected in the Parameter Variable combo. When selecting a cell in the table, sensitivity analysis is performed using the selected cpt value. The radio boxes "MIN SV" and "MAX SV" allows the user to view a color chart that represents the sensitivity values for every cpt index.



Has bronc...	yes		no	
Tuberculo...	yes	no	yes	no
yes	0.9	0.8	0.7	0.1
no	0.1	0.2	0.3	0.9

Figure 1a: CPT table of Dyspnoea with a color chart showing the minimum Sensitivity values.

When a color chart with the minimum sensitivity values is selected (Figure 1a), the darker the red color is, the lower the minimum sensitivity value is for the corresponding cpt value.



Has bronc...	yes		no	
Tuberculo...	yes	no	yes	no
yes	0.9	0.8	0.7	0.1
no	0.1	0.2	0.3	0.9

Figure 1b: CPT table of Dyspnoea with a color chart showing the maximum Sensitivity values.

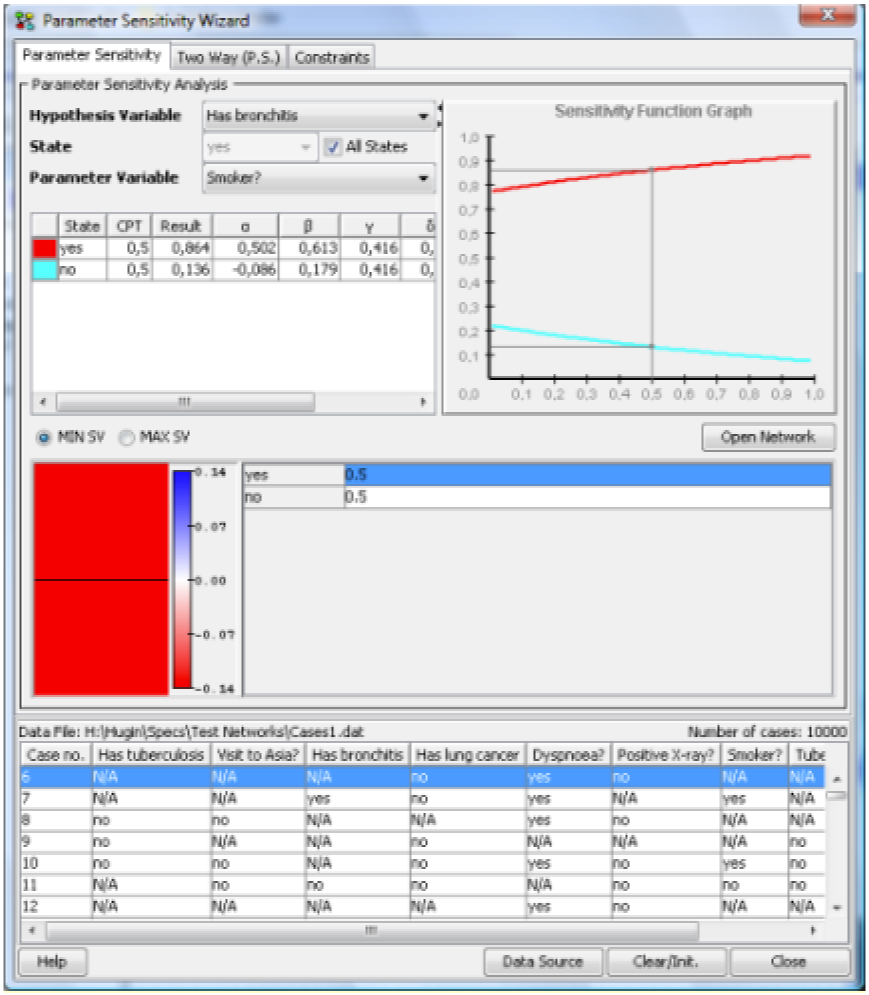
When a color chart with the maximum sensitivity values is selected, the darker the blue color is, the higher the maximum sensitivity value is for the corresponding cpt value (Figure 1b).

Pointing on a cell in the color chart, displays the actual value of the MIN/MAX sensitivity value; clicking on it selects the corresponding cpt index and performs sensitivity analysis.

The “MIN SV” and “MAX SV” are only enabled when the user has selected to perform sensitivity analysis on all states because otherwise, each cpt value corresponds to one sensitivity value so a color chart showing those values is displayed.

Import a case file

It is possible to perform parameter sensitivity analysis after inserting a case stored in a case file or a set of cases stored in a data file. A case file represents a single case whereas a data files represents a number of cases for a given network. The “Data Source” button imports a file in the wizard and gives the user the possibility to select cases and enter them as evidence in the domain. Sensitivity analysis can now be performed for each case in the file. In Figure 2, a case file “asia-cases” is imported and displayed as a table. When clicking on a case, it is entered as evidence in the domain and after performing sensitivity analysis as shown above (Figure 1), the sensitivity values calculated and the lines in the graph, are different because the new evidence is taken in to consideration.



Hypothesis Variable: Has bronchitis

State: yes ☑ All States

Parameter Variable: Smoker?

	State	CPT	Result	α	β	γ	δ
🟥	yes	0,5	0,864	0,502	0,613	0,416	0,
🟦	no	0,5	0,136	-0,086	0,179	0,416	0,

● MIN SV ○ MAX SV

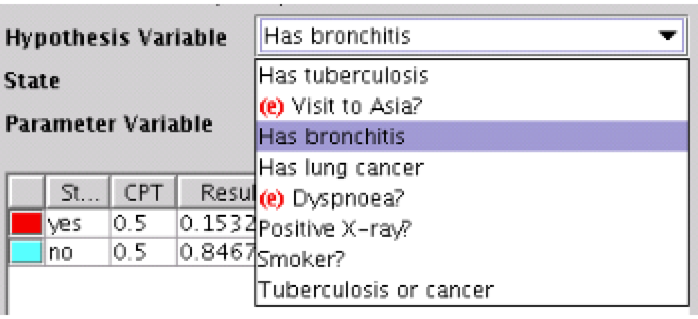
yes	0.5
no	0.5

Data File: H:\Hugin\Specs\Test Networks\Cases1.dat   Number of cases: 10000

Case no.	Has tuberculosis	Visit to Asia?	Has bronchitis	Has lung cancer	Dyspnoea?	Positive X-ray?	Smoker?	Tube
6	N/A	N/A	N/A	no	yes	no	N/A	N/A
7	N/A	N/A	yes	no	yes	N/A	yes	N/A
8	no	no	N/A	N/A	yes	no	N/A	N/A
9	no	N/A	N/A	no	N/A	N/A	N/A	no
10	no	no	N/A	no	yes	no	yes	no
11	N/A	no	no	no	N/A	no	no	no
12	N/A	N/A	N/A	N/A	yes	no	N/A	N/A

Figure 2: Sensitivity analysis with evidence from a case file.

Sensitivity analysis can not be performed on a hypothesis node with evidence entered. To avoid error messages, the node names of nodes with evidence are marked with (e) in the Hypothesis Variable Combo. (Figure 2a)



Hypothesis Variable: Has bronchitis

State

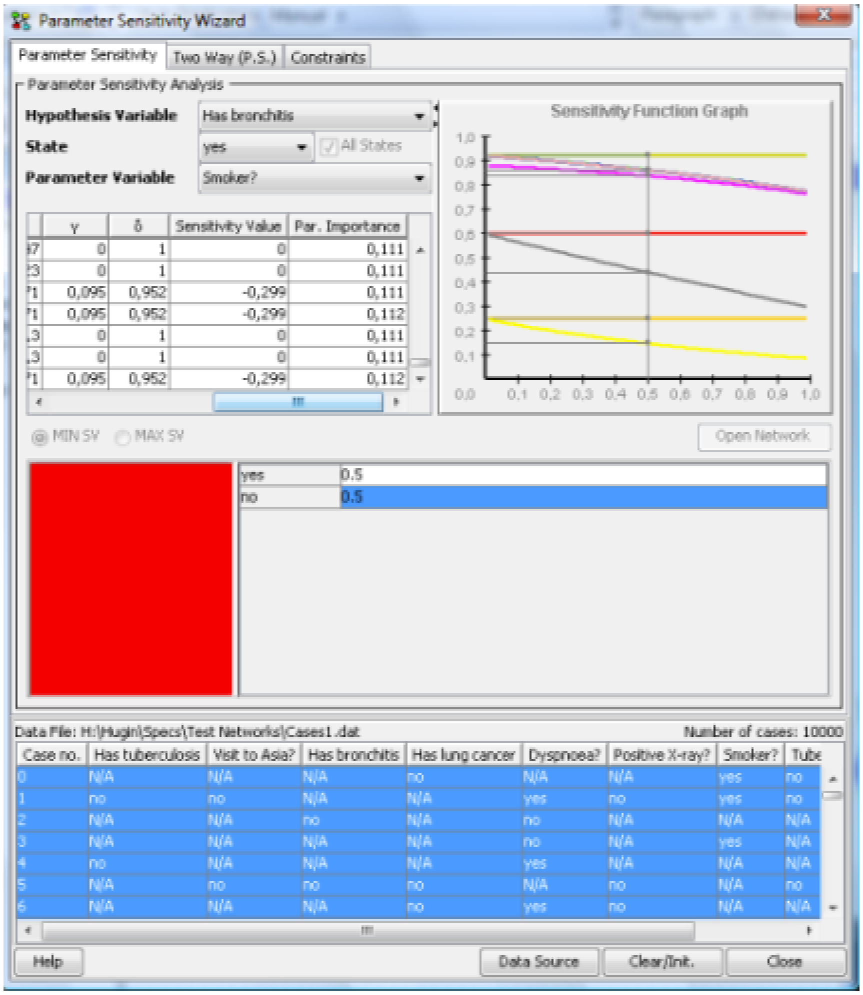
Parameter Variable

	St...	CPT	Resul
🟥	yes	0.5	0.1532
🟦	no	0.5	0.8467

Figure 2a: Hypothesis Variable Combo

Accross cases

When cases are loaded in the wizard, it is possible to perform parameter sensitivity analysis across many cases. Selecting all (Ctrl-A) or some cases in the cases table (Figure 2b) and then computing the sensitivity for the selected hypothesis and parameter, will generate a table in the information panel that displays the results for all cases. Each row in the table shows the results of the parameter sensitivity analysis for the selected state of the hypothesis, in each case. It is not possible to compute sensitivity for all states of the hypothesis accross cases. No results are shown for cases where the computation has failed (fx. when evidence is entered in the hypothesis).



Hypothesis Variable: Has bronchitis

State: yes ☑ All States

Parameter Variable: Smoker?

	γ	δ	Sensitivity Value	Par. Importance
47	0	1	0	0,111
23	0	1	0	0,111
71	0,095	0,952	-0,299	0,111
71	0,095	0,952	-0,299	0,112
13	0	1	0	0,111
13	0	1	0	0,111
71	0,095	0,952	-0,299	0,112

● MIN SV ○ MAX SV

yes	0.5
no	0.5

Data File: H:\Hugin\Specs\Test Networks\Cases1.dat

Number of cases: 10000

Case no.	Has tuberculosis	Visit to Asia?	Has bronchitis	Has lung cancer	Dyspnoea?	Positive X-ray?	Smoker?	Tube
0	N/A	N/A	N/A	no	N/A	N/A	yes	no
1	no	no	N/A	N/A	yes	no	yes	no
2	N/A	N/A	no	N/A	no	N/A	N/A	N/A
3	N/A	N/A	N/A	N/A	no	N/A	yes	N/A
4	no	N/A	N/A	N/A	yes	N/A	N/A	N/A
5	N/A	no	no	no	N/A	no	N/A	no
6	N/A	N/A	N/A	no	yes	no	N/A	N/A

Figure 2b: Sensitivity computed accross cases.

The information table in addition to the Sensitivity Value, also displays the Parameter Importance value as it adjusts through every case. The parameter importance value at the last row is the final one computed after going through all cases.

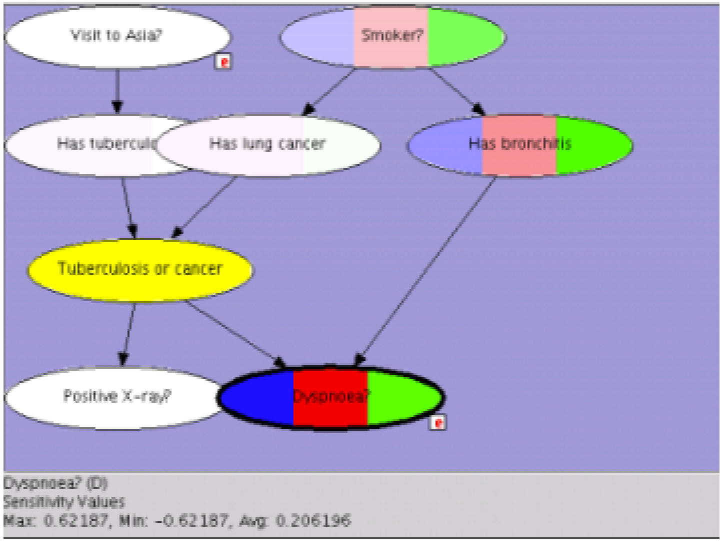
Selecting a row in the info table will show the Sensitivity function graph for the selected case.

Open Network

The "Sensitivity Set Graph Panel" is a tool to help visually capture the results of the parameter sensitivity analysis, directly on the network. It is accessibly by pressing the "Open Network" button in the "Parameter Sensitivity Panel". Every node is divided in three parts and every part is painted in a different colour. Starting from the left of a node, the colours used are blue, red and green which represent different values.

- The blue colour (left) represents the **maximum** sensitivity value.
- The red colour (center) represents the **minimum** (negative) sensitivity value.
- The green colour (right) represents the **average** sensitivity value.

The tone of each colour indicates how high the value they represent is. In case of the maximum and average sensitivity values (blue, green), the higher the number is, the darker the colour becomes. In case of the minimum (red) value the opposite applies. If a value is 0, the colour gets toned down to white. The yellow colour indicates that it was not possible to compute sensitivity values for that node, either because the propagation failed (because of inconsistent evidence or evidence is entered in the hypothesis) or because the values in the node's CPT table are 0 and 1.



```
Dyspnoea? (D)
Sensitivity Values
Max: 0.62187, Min: -0.62187, Avg: 0.206196
```

Figure 3: Sensitivity set graph

Figure 3 shows an example of the "asia" network. In this example, it is immediately visible which nodes have the highest values and which have values 0 or close to 0. By pointing on the "Dyspnoea" node, the node's minimum, maximum and average values are displayed at the bottom of the panel together with its name and label. Selecting a node on the network will set it as the parameter variable in the *"Parameter Sensitivity Panel"* (page 335).

4.6.18 Two Way Parameter Sensitivity

Parameter Sensitivity analysis is the analysis of how sensitive the results of a belief update (propagation of evidence) is to variations of the value of a parameter of the model. The parameters of a model are the entries of the conditional probability distributions.

The two way parameter sensitivity analysis observes the resultes of a belief update to variations of the values of two parameters of the model. The parameter values could be entries of the conditional probability distribution from one or two nodes.

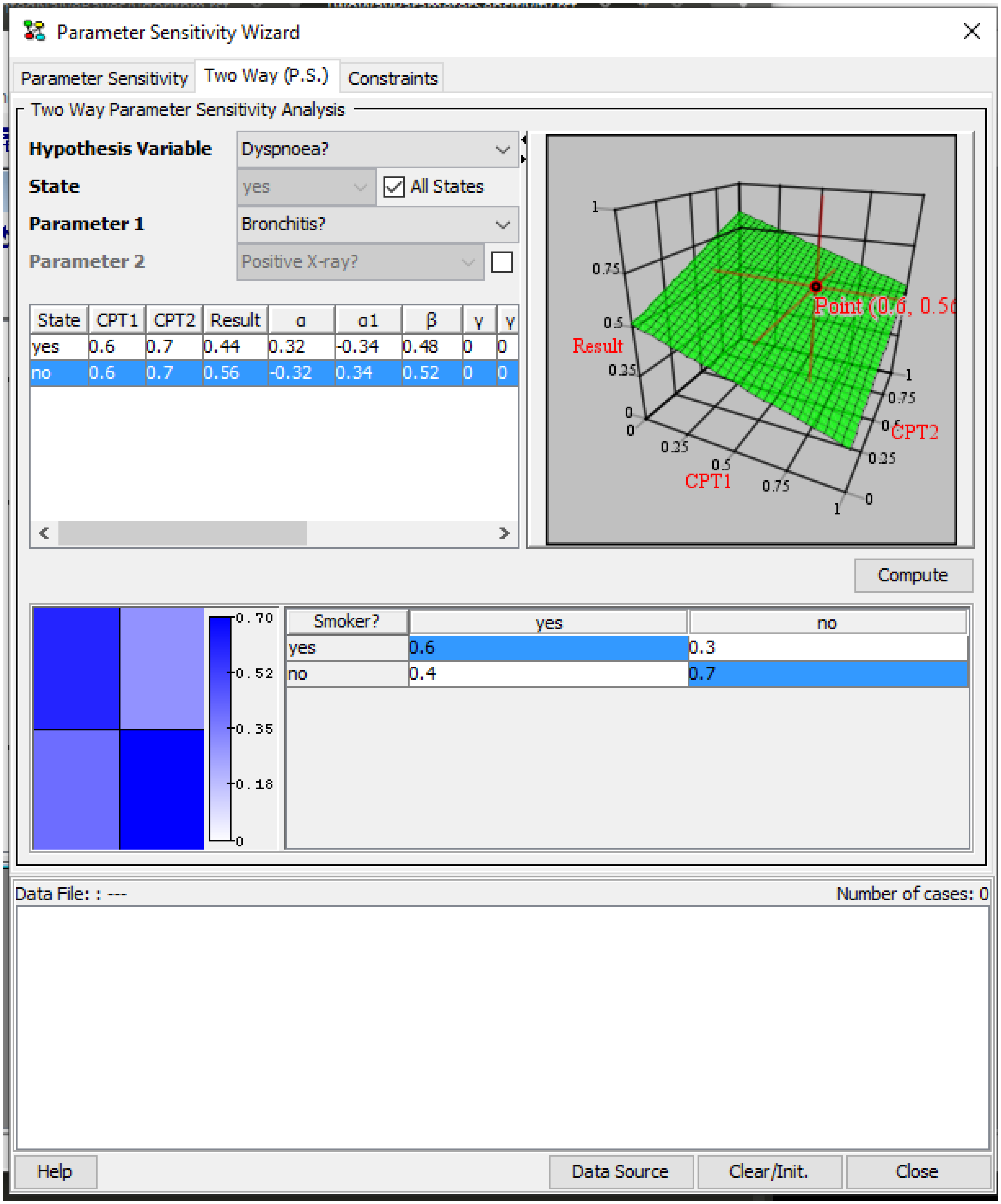
- *Two way parameter sensitivity Analysis* (page 341)

- *Import a case file* (page 343)
- *Sensitivity across cases* (page 343)

Two way sensitivity analysis

The Two Way (P.S) panel performs parameter sensitivity analysis with values from two parameters of the model. On the top left corner there are four drop down boxes that accept user input.

- **"Hypothesis variable"** : Select the node to be observed.
- **"State"** : Select the state of the hypothesis node to be observed. To observe all states check the "All states" check box.
- **"Parameter 1"** : Select the node from which we will select parameter values. The probability distribution table (CPT) of the selected node is displayed in the middle of the panel. To perform sensitivity analysis on changes to two parameter values from the same CPT, select two values in the CPT and press "compute". The results are shown in the information table and the 3D graph, simular to the *Parameter Sensitivity Panel* (page 335).
- **"Parameter 2"** : To perform parameter sensitivity analysis on changes to two parameter values from two CPTs, check the check box next to "Parameter 2" drop down box to enable it. Selecting a second node in this drop down will display a tabbed panel in the middle of the Two Way (P.S) Panel that holds two CPTs. Select one parameter value in each CPT and press "compute".



Hypothesis Variable Dyspnoea?

State yes ☑ All States

Parameter 1 Bronchitis?

Parameter 2 Positive X-ray? ☐

State	CPT1	CPT2	Result	α	α1	β	γ	γ
yes	0.6	0.7	0.44	0.32	-0.34	0.48	0	0
no	0.6	0.7	0.56	-0.32	0.34	0.52	0	0

Smoker?	yes	no
yes	0.6	0.3
no	0.4	0.7

Data File: : ---

Number of cases: 0

Figure 1: Parameter Sensitivity Analysis

When performing parameter sensitivity analysis with two parameters from the same CPT the information table displayed includes the following columns:

- **State**: The state of the hypothesis node.

- **CPT1, CPT2**: The first and the second parameters selected in the CPT.
- **Result**: The belief of the given state of the hypothesis.
- α, α_1, β, γ, γ_1, δ: Sensitivity constants.
- **Sensitivity value1**: The sensitivity value of the current states to changes of the value of the first parameter (CPT1). The sensitivity value1 is computed based on the function:

$$\frac{(\alpha_1 * \gamma_2 - \alpha_2 * \gamma_1) * cpt_2 + \alpha_1 * \delta - \gamma_1 * \beta}{(\gamma_1 * cpt_1 + \gamma_2 * cpt_2 + \delta)^2}$$

- **Sensitivity value2**: The sensitivity value of the current states to changes of the value of the second parameter (CPT2). The sensitivity value2 is computed based on the function:

$$\frac{(\alpha_2 * \gamma_1 - \alpha_1 * \gamma_2) * cpt_1 + \alpha_2 * \delta - \gamma_2 * \beta}{(\gamma_1 * cpt_1 + \gamma_2 * cpt_2 + \delta)^2}$$

The sensitivity function used to generate the 3D graph is: $\frac{\alpha_1 * cpt_1 + \alpha_2 * cpt_2 + \beta}{\gamma_1 * cpt_1 + \gamma_2 * cpt_2 + \delta}$

The Sensitivity values are currntly not computed when performing parameter sensitivity analysis with two parameters from two CPTs.

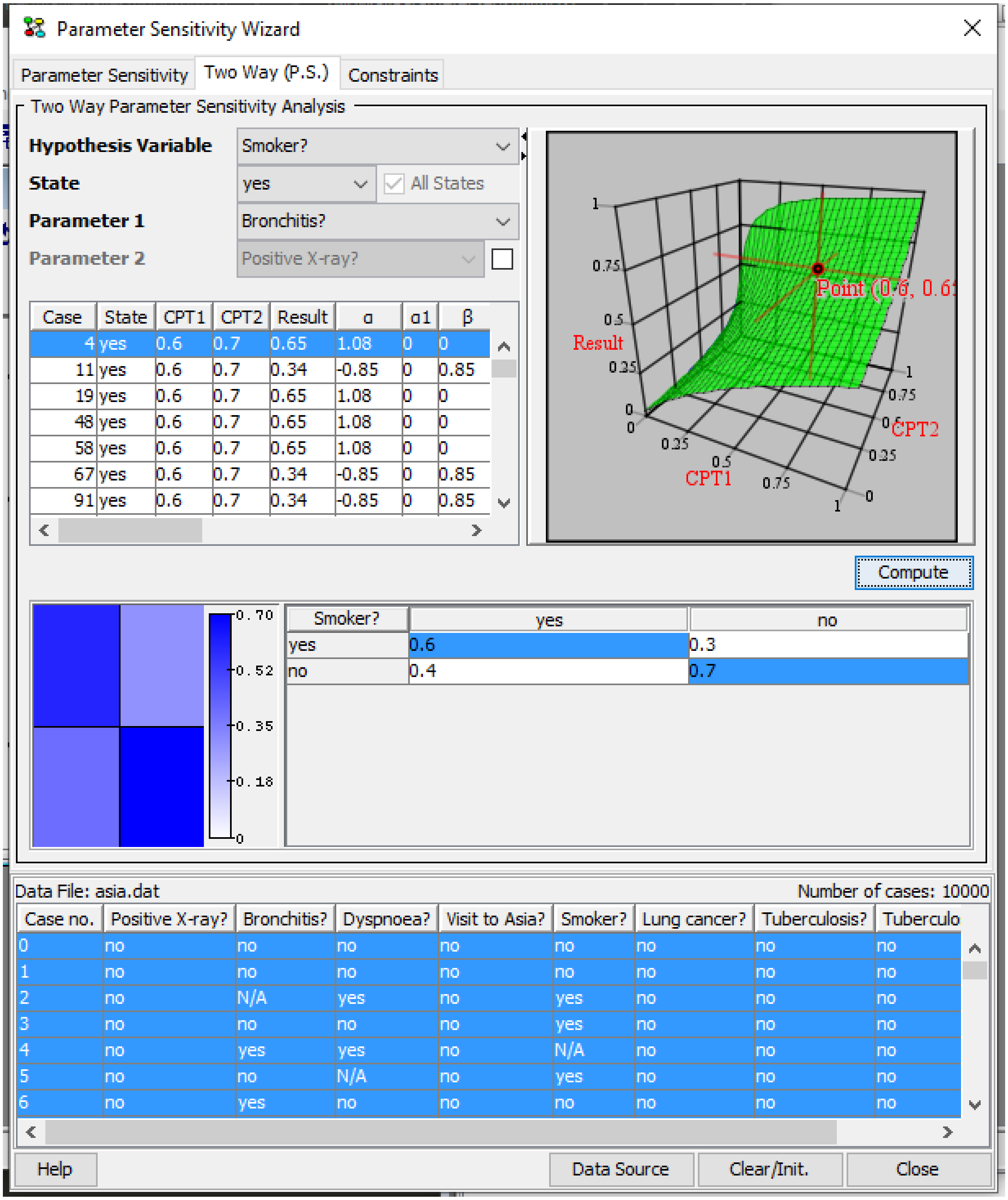
Import a case file

It is possible to perform parameter sensitivity analysis after importing cases from a *case file* (page 85) or a database. The “Data Source” button imports cases in the wizard and gives the user the option of selecting a case to be entered as evidence in the domain. Sensitivity analysis can now be performed for each case in the file. In Figure 2, a case file is imported and displayed as a table. When clicking on a case, it is entered as evidence in the domain and after performing sensitivity analysis as shown above (Figure 1), the sensitivity values calculated and the lines in the graph, are different because the new evidence is taken in to consideration.

Sensitivity analysis can not be performed on a hypothesis node with evidence entered. To avoid error messages, the node names of nodes with evidence are marked with (e) in the Hypothesis Variable Combo.

Across Cases

When cases are loaded in the wizard, it is possible to perform parameter sensitivity analysis across many cases. Selecting all (Ctrl-A) or some cases in the cases table (Figure 2) and then computing the sensitivity for the selected hypothesis and parameters, will generate a table in the information panel that displays the results for all cases. Each row in the table shows the results of the parameter sensitivity analysis for the selected state of the hypothesis, in each case. It is not possible to compute sensitivity for all states of the hypothesis across cases. No results are shown for cases where the computation has failed (fx. when evidence is entered in the hypothesis).



Hypothesis Variable Smoker?

State yes ☑ All States

Parameter 1 Bronchitis?

Parameter 2 Positive X-ray? □

Case	State	CPT1	CPT2	Result	α	α1	β
4	yes	0.6	0.7	0.65	1.08	0	0
11	yes	0.6	0.7	0.34	-0.85	0	0.85
19	yes	0.6	0.7	0.65	1.08	0	0
48	yes	0.6	0.7	0.65	1.08	0	0
58	yes	0.6	0.7	0.65	1.08	0	0
67	yes	0.6	0.7	0.34	-0.85	0	0.85
91	yes	0.6	0.7	0.34	-0.85	0	0.85

Smoker?	yes	no
yes	0.6	0.3
no	0.4	0.7

Data File: asia.dat Number of cases: 10000

Case no.	Positive X-ray?	Bronchitis?	Dyspnoea?	Visit to Asia?	Smoker?	Lung cancer?	Tuberculosis?	Tuberculo
0	no	no	no	no	no	no	no	no
1	no	no	no	no	no	no	no	no
2	no	N/A	yes	no	yes	no	no	no
3	no	no	no	no	yes	no	no	no
4	no	yes	yes	no	N/A	no	no	no
5	no	no	N/A	no	yes	no	no	no
6	no	yes	no	no	no	no	no	no

Figure 2: Sensitivity computed across cases.

The information table in addition to the Sensitivity Value, also displays the Parameter Importance value as it adjusts through every case. The parameter importance value at the last row is the final one computed after going through all cases. The parameter importance value is currently not computed when performing parameter sensitivity analysis with two parameters from two CPTs.

Selecting a row in the info table will show the Sensitivity function graph for the selected case.

4.7 Bayesian Network

4.7.1 Bayesian Networks

A *Bayesian network* is a set of nodes representing random variables and a set of links connecting these nodes in an acyclic manner. Each node has assigned a function which describes how the state of the node depends on the parents of the node.

In HUGIN networks, you can represent two kinds of random variables: *discrete chance nodes* (page 223) having a discrete finite state space and *continuous chance nodes* (page 224) having a continuous infinite state space.

For the discrete chance nodes, the function describing how the node depends on its parents is a *conditional probability table* (page 241). For *continuous chance nodes* (page 224) it is a probability density function (PDF) - in HUGIN it must be a *Gaussian (normal) distribution function* (page 117).

In HUGIN, you can also construct *influence diagrams* (page 345) which are Bayesian networks extended with decision nodes and a utility functions.

A Bayesian network is really just a smart representation of a domain of dependent random variables.

Many real-life situations can be modeled as a domain of random variables. In a medical domain such random variables could represent risk factors, diseases, symptoms, patho-physiological features, etc. A domain of random variables could form the basis of a decision support system to help decision makers make the decision that is most beneficial in a given situation.

If you want to represent a domain of random variables (all having a discrete and finite state space), you can always do this by the joint probability table of the entire domain. That is, a table with an entry for each configuration of the nodes of the domain. However, the number of configurations of a domain grows exponentially in the number of random variables, so this would only work for very small domains.

What you use to keep the representation size to a minimum in networks is the conditional independences in the domain: Very often the knowledge about a random variable being in a specific state will make other variables independent and thus it would be an overkill to have an entry for all combinations of these independent variables (they would all contain the same value).

This is not the right place to describe the theory behind Bayesian networks in detail. You should be able to find some useful literature about the subject elsewhere. A good place to start is the textbook *Bayesian Networks and Decision Graphs* (page 541). See also a brief overview of the three main *Paradigms of Expert Systems* (page 16).

4.7.2 Limited Memory Influence Diagram

A limited memory influence diagram is a *Bayesian network* (page 345) extended with decision facilities (*decision nodes* (page 224) and *utility nodes* (page 226)). The limited memory influence diagram (LIMID) extends the traditional influence diagram by relaxing two fundamental assumptions: a non-forgetting decision maker and a total order on decisions. In a LIMID all information available to the decision maker must be specified using informational *links* (page 147) and decisions may by unordered.

Due to the elimination of the non-forgetting assumption of the decision maker it is in a LIMID important to specify for each decision exactly what information is available to the decision maker. Each variable which is observed prior to making a decision should have a link going into the decision. If the state of a node will be known at the time of making a decision, this will (probably) have an impact on what the decision maker should do. Thus, one must add a *link* (page 147) from the node to the decision node.

If a node *X* is (informational) parent of a decision $\mathrm{D_i}$, but not parent of a subsequent decision $\mathrm{D_j}$, this implies that *X* is known prior to decision $\mathrm{D_i}$, and not known at subsequent decision $\mathrm{D_j}$,. This implies that the observation is not known by the decision maker at second decision.

The solution to a LIMID is a strategy consisting of one policy for each decision. The policy is a function from the known variables to the states of the decision. It is not a function of all past observations as the decision maker is assumed only to know the most recent observation on loses leaves. This is different from the traditional influence diagram where the policy would be a function from all past observations and decisions as the decision maker is assumed to be non-forgetting. There need not be a total order on the decisions.

Each decision in the LIMID has an initial policy which can be defined by the user either by hand or using expressions. The initial policy is a table specifying a mapping from configurations of the parents of the decision to states of the decision. This initial policy is updated in the process of solving the LIMID.

The solution to a LIMID is determined using *Single Policy Updating* (page 258).

An influence diagram cannot contain *continuous chance nodes* (page 224).

4.7.3 Object-Oriented Network

An *Object-Oriented Network* is a network (i.e., Bayesian network or (limited-memory) influence diagram) that, in addition to the usual nodes, contains *instance nodes*. An instance node is a node representing an instance of another network. In other words, an instance node represents a subnet. Of course, the network of which instances exist in other networks can itself contain instance nodes, whereby an object-oriented network can be viewed as a hierarchical description (or model) of a problem domain. There are three main advantages of constructing a network using instance nodes:

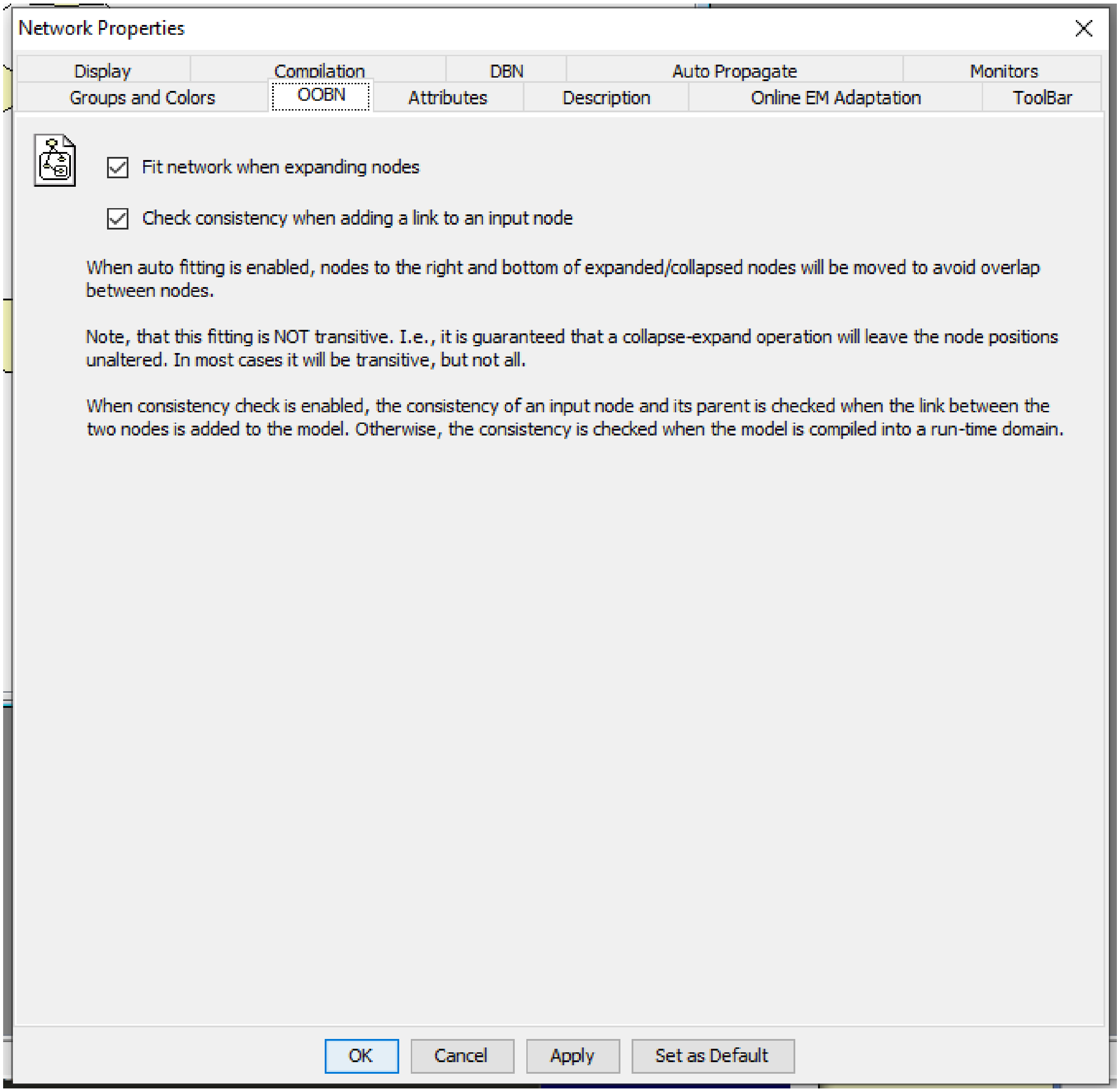
- The model construction activity most often involves repeated changes of level of abstraction. That is, it is performed in a top-down fashion, a bottom-up fashion, or a mix of the two. Such repeated changes of focus are due partly to the fact that humans naturally think about systems in terms of hierarchies of abstractions and partly due to lack of ability to mentally capture all details of a complex system simultaneously. The use of instance nodes provides support for working with different levels of abstraction in constructing network models.
- As systems often are composed of collections of identical or almost identical components, models of systems often contain repetitive patterns (i.e., commonly occurring solutions or problem types). In Bayesian networks and influence diagrams, such patterns are network fragments. The notion of instance nodes makes it very easy to construct multiple identical instances of a network fragment.
- Describing a network in a hierarchical fashion often makes the network much less cluttered, and thus provides a much better means of communicating ideas among knowledge engineers and users.

An instance node connects to other nodes via some of the (basic) nodes in the copy of the network (the *master*) of which it is an instance. (Note that an instance node should be thought of as a copy of the network of which it is an instance.) These nodes are known as *interface nodes*. As we wish to support information hiding, the interface nodes only comprise a subset of the nodes in the master network. Interface nodes are subdivided into a set of *input nodes* and *output nodes*:

- Input nodes of an instance of a master network are not real nodes but only to be considered as placeholders for (basic) nodes of the network(s) containing instances of the master network. These basic nodes are said to be *bound* to the input nodes (and vice versa). Note that if an input node of an instance hasn't been bound, a default potential (probability table) will be associated with it. The probabilities in this table is specified in the master network.
- Output nodes of an instance of a master network are real nodes that can be specified as parents of nodes in the network containing the instance node or can be bound to an input node of another instance node of the network.

Instance nodes can be displayed in two different modes: Expanded (showing its interface nodes) and collapsed (having the same size as other nodes). To avoid problems with overlapping nodes and increased node distances when expanding and collapsing an instance node, respectively, the HUGIN Graphical User Interface provides a rudimentary feature for moving nodes that would otherwise overlap with the instance node or be placed too far away from it when expanding or collapsing, respectively, the instance node. This feature can be enabled or disabled by clicking the *"Fit network when expanding notes"* check box of the OOBN tab of the *Network Properties Pane* (page 178) (see Figure 1).

When connecting a node to an to input nodes in an OOBN instance two nodes must be consistent. To be consistent, they must have the same *"Category"* and *"Kind"*. The Category indicates whether it is a chance node, a decision node, a utility node, or an instance node. The Kind indicates whether a chance or a decision node is discrete or continuous. Furthermore, if the kind of the node is discrete, it must have the same number of states, and finally, if it is Numeric, the state values must be the same. The Hugin Graphical User Interface offers to check whether these rules are satisfied while the model is being constructed or only when the model is compiled. To disable the checks while the model is constructed un-check the box labelled "Check consistency when adding a link to an input node" (see Figure 1).



☑ Fit network when expanding nodes

☑ Check consistency when adding a link to an input node

When auto fitting is enabled, nodes to the right and bottom of expanded/collapsed nodes will be moved to avoid overlap between nodes.

Note, that this fitting is NOT transitive. I.e., it is guaranteed that a collapse-expand operation will leave the node positions unaltered. In most cases it will be transitive, but not all.

When consistency check is enabled, the consistency of an input node and its parent is checked when the link between the two nodes is added to the model. Otherwise, the consistency is checked when the model is compiled into a run-time domain.

Figure 1: The Network Properties dialog box showing the OOBN tab.

The *tutorial on object-oriented networks* (page 56) shows how to construct an object-oriented network using the HUGIN

Graphical User Interface.

4.7.4 Conditional Independence and Dependence

The notion of conditional independence and dependence (CID) is central in Bayesian networks. For example, the CID statements of a Bayesian network play an important role in (structural) *model verification* (page 348), explanation (analysis of relevance and irrelevance), and (of course) for efficiency of inference.

The CID statements of a Bayesian network is encoded by its graphical structure, and can be read using the rules of *d-separation* (page 289).

4.7.5 Model Verification

A very important activity in model construction is verification of model structure through analysis of the conditional independence and dependence statements encoded by the structure.

Letting arrows point from effect to cause instead of from cause to effect is one of the most frequent mistakes made, and which can lead to severe problems with specifying the CPTs (e.g., if arrows point from a large number of effect variables to a single cause variable) and wrong inference.

As an example of the practical use of CID analyses to perform structural model verification, consider the problem of constructing a model for detection of fraud usage of credit cards. Assume we have the following three Boolean variables:

A: Two or more PC's bought within a few days using the same credit card.

B: Credit card used almost at the same time at different locations.

C: Fraud.

The model $A \rightarrow C \leftarrow B$ may at first glance appear to be natural (A and B are used as "inputs" to determine the probability of C).

From the rules of *d-separation* (page 289), however, we can see that this model tells us that observing A (or B) does not provide us with any information about B (or A) when C is unknown. This is wrong, as observing A (or B) increases our belief in C, which in turn increases our belief that we might also observe B (or A). Therefore, this model simply gives wrong probabilities!

The model $A \leftarrow C \rightarrow B$, on the other hand, makes A and B dependent when we have no (definite) knowledge about C (i.e., observing A (or B) will increase our belief that we will also observe B (or A)). When we have definite knowledge about C (i.e., the value of C is known), then this model tells us that A and B are independent, which seems to be quite reasonable (if we know for sure that we are considering a fraud case, then observing A (or B) will not change our belief about whether or not we are going to observe B (or A)).

4.7.6 OOBN EM Learning

Instead of estimating parameters for a class in isolation from the context of an object oriented model where the class may have numerous instances, OOBN EM estimates the class parameters using evidence on all instances and the entire object oriented model.

The initial contents of the experience tables of nodes in the object-oriented model form the basis for the computation of "prior counts". However, in the current implementation, prior experience is not supported for continuous nodes in objectoriented models (that is, the experience tables for such nodes must not contain positive values).

The contents of the updated experience tables reflect the fact that many instance nodes contribute to the learning of the same node in a class (i.e., the experience counts will be higher than the number of cases in the data set). Otherwise, everything specified in *EM Learning* (page 352) also apply for OOBN EM.

OOBN EM is an integrated part of EM learning in HUGIN, either press the EM Learning Button or select the *EM Learning Wizard* (page 281) item of the Wizards Menu.

Notes on OOBN data file format: Dot notation for class instances

When specifying a data file for an object oriented model, the instance nodes must be prefixed by a class instance name and a dot.

Example: How would a data file for EM in the Disease object oriented model (from the *How to build OOBNs tutorial* (page 56)) look like?

The class for which we are estimating parameters is displayed in Figure 1.

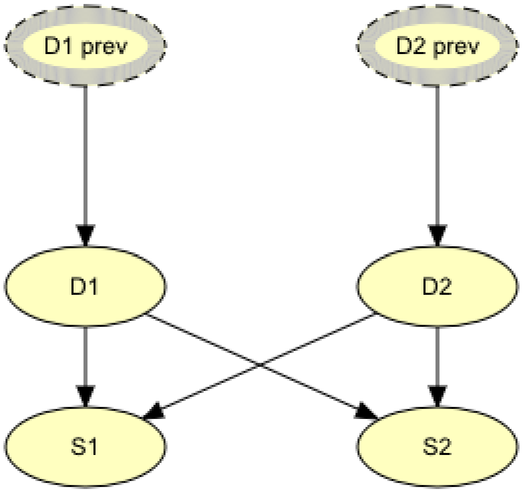


Figure 1: BN for a single time slice of the Diseases problem.

The object oriented model using instances of the class is displayed in figure 2.

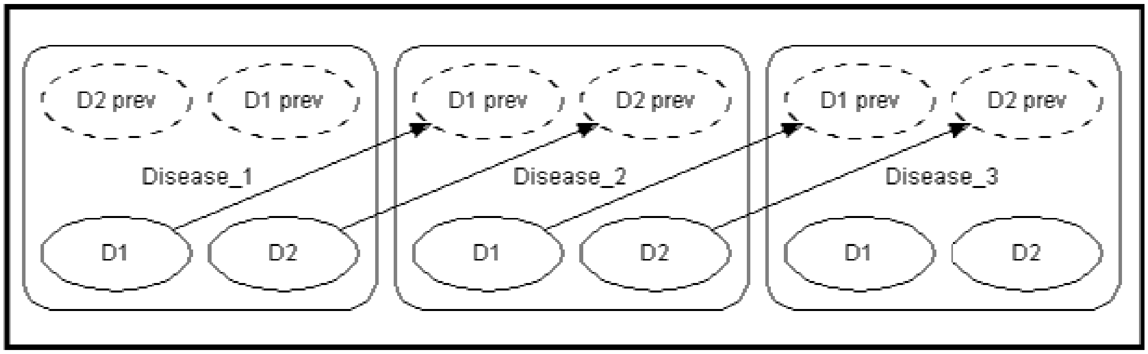


Figure 2: Object-oriented network representing the Diseases problem.

The data file header must be structured like this:

DiseaseSlice_3.**S2**, DiseaseSlice_3.**S1**, DiseaseSlice_3.**D2**, DiseaseSlice_3.**D1**, DiseaseSlice_2.**S2**, DiseaseSlice_2.**S1**, DiseaseSlice_2.**D2**, DiseaseSlice_2.**D1**, DiseaseSlice_1.**S2**, DiseaseSlice_1.**S1**, DiseaseSlice_1.**D2**, DiseaseSlice_1.**D1**, DiseaseSlice_1.**D2_prev**, DiseaseSlice_1.**D1_prev**

Notice how the the class nodes **D1**, **D2**, **S1**, **S2**, **D1_prev** and **D2_prev** have been prefixed by the class instance names *DiseaseSlice_3*, *DiseaseSlice_2* and *DiseaseSlice_1*.

See also description of *data files* (page 389).

4.7.7 Dynamic Bayesian Network

Dynamic Bayesian (or Belief) Networks (DBNs) are used to model systems that evolve over time. The state of the system at a single time instant is modeled using an ordinary static Bayesian network. By linking together a number of these networks, a dynamic model is obtained. The part of the network corresponding to a single time instant is called a time slice.

When modeling a system that evolves over time, random variables at a given time instant typically depend on the state of the system at past time instants. In order to specify time dependencies, *temporal clones* (page 131) can be constructed for the regular nodes.

Time window

In runmode the DBN is unfolded and the time slices can be navigated through the *node list pane* (page 243).

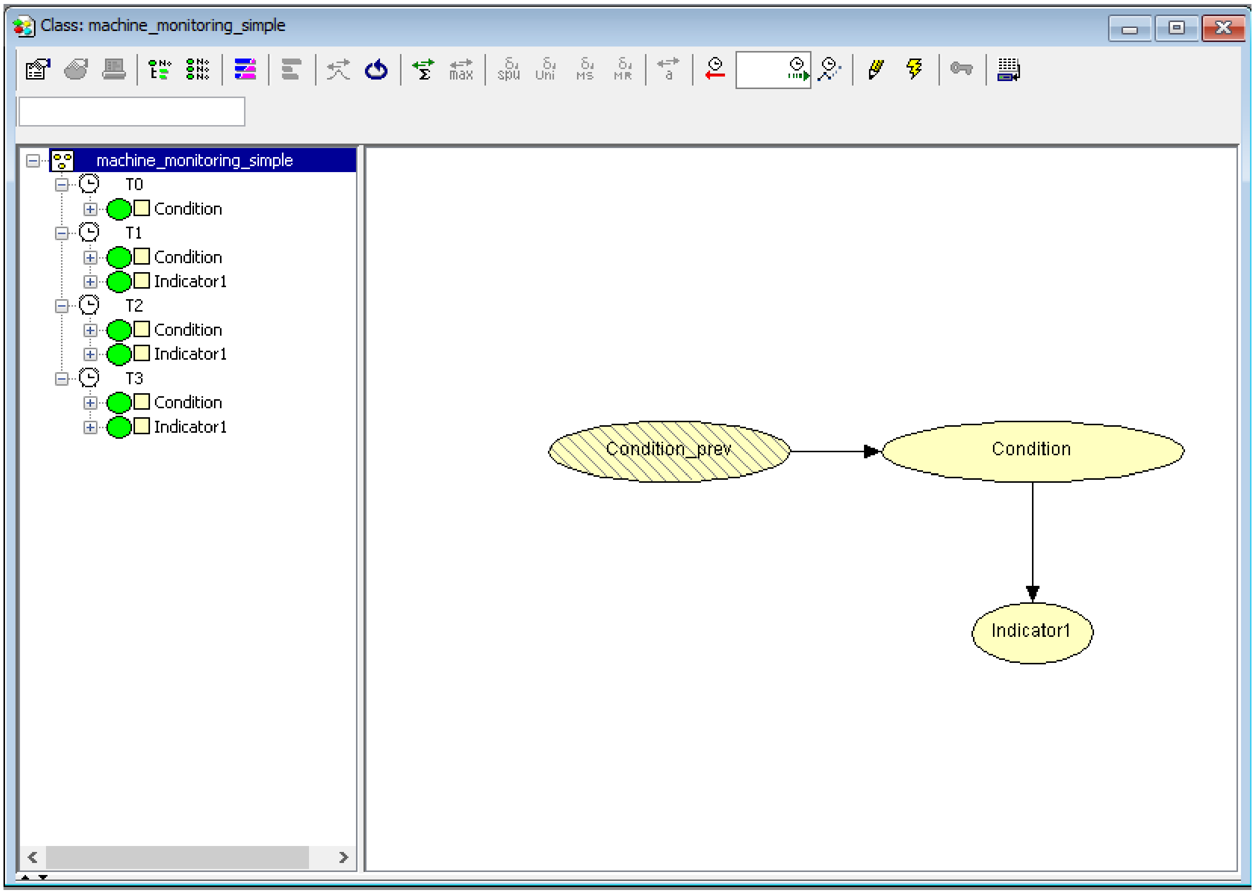


Figure 1: Dynamic Bayesian Network in Runmode, time window representing three time slices (T1-T3), node *Condition_prev* is *temporal clone* (page 131) of *Condition*.

T1-n represents the time slices. T0 represents the temporal nodes prior to T1.

Moving the time window

To move the time window one step forward, click the clock icon inside the move time window button .

It is possible to move the time window an arbitrary number of steps forward. To do this type a number inside the white text area next to the move time window button, eg. 12 . The mouse pointer must hover over the text field while editing the number. The maximum possible number of steps is 231-1 but to use numbers larger than a few thousands is not advisable due to long response times. If the text field is left empty or illegal values are entered, the default behavior is to move ahead one step.

Evidence for nodes that disappear from the time window become “frozen” (i.e., it cannot be retracted or altered). But the evidence is not forgotten - it is “incorporated” in the beliefs of nodes in future time slices.

To compute beliefs for nodes in time slices that lie beyond the time window use the DBN *prediction* (page 138) function.

Dynamic Bayesian Networks with Mixed Variables

All kinds of nodes are allowed in HUGIN DBNs. The DBN may include both discrete and continuous chance nodes as well as have temporal clones. In this case, the continuous nodes must follow a conditional linear Gaussian distribution, while both discrete and continuous nodes can have *temporal clones* (page 131) as demonstrated in the figure below.

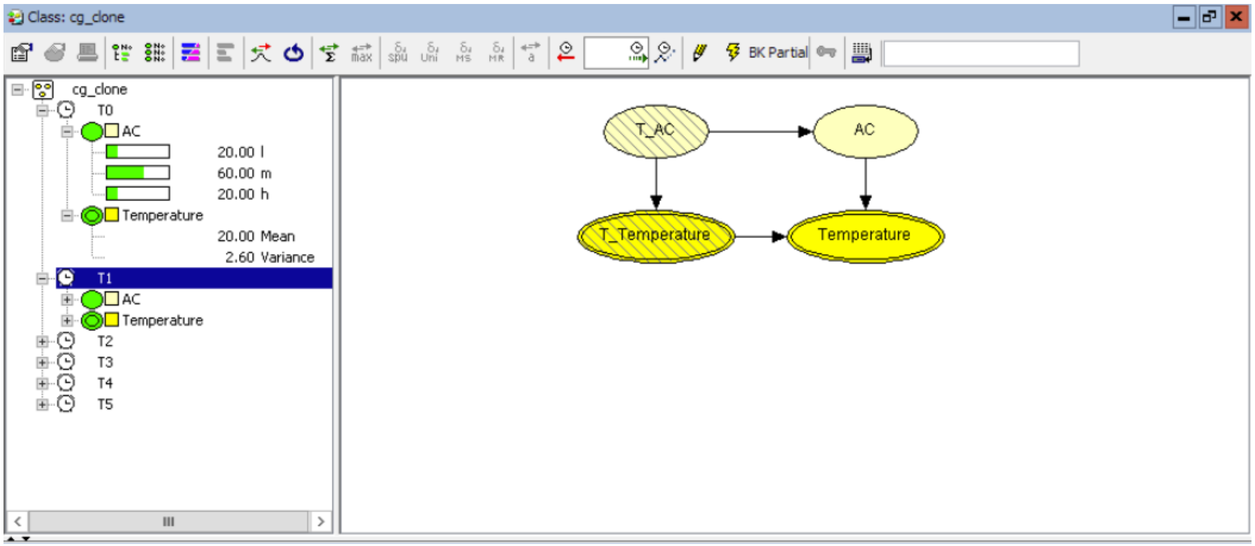


Figure 2: Dynamic Bayesian Network with both discrete and continuous nodes.

However, in the case of mixed discrete and continuous nodes, the model can only be compiled using partial Boyen-Koller approximation, see DBN network properties.

4.8 Data Manipulation

4.8.1 EM Learning

This section describes the feature of using data (i.e., a set of cases) to estimate the conditional probability distributions and densities when only the graphical structure is given. This is known as batch learning. Note that if the initial conditional probability distributions and densities are also given, it is still possible to use batch learning.

Batch learning requires that all data are available when the learning process starts, whereas sequential learning allows the knowledge to be updated incrementally, but sequential learning requires that initial assessments of the conditional probabilities are given, whereas batch learning only requires the graphical structure. The batch learning method currently implemented in HUGIN is called EM-algorithm or EM-learning and was developed by *Lauritzen (1995)* (page 541); see also the work of *Cowel & Dawid (1992)* (page 541).

A case is an assignment of values to some or all the nodes of a domain. If values have been assigned to all nodes, the case is said to be *complete*; otherwise, it is said to be *incomplete*. The data used by the learning algorithm is comprised of a set of cases. The cases are numbered sequentially from *0 to N-1*, where *N* is the total number of cases.

Before learning can take place, in addition to the data set, the set of nodes for which conditional probability distributions and densities should be estimated must be specified. This set is specified as the set of nodes having experience tables (described in section *Learning - Adaptation* (page 244)). The input to the learning algorithm is the cases data; this set of cases must be non-empty. Note that if the domain is a LIMID model, then the domain's decision nodes must be instantiated in all cases.

The learning algorithm needs to do inference, so the domain must be compiled before the learning algorithm is called. If successful, the procedure will update the conditional probability (or density) and the experience table for each node of the domain that has an experience table. Moreover, the inference engine will be reset and use the new conditional probability (or density).

If the contents of the experience tables are all zeros, then the algorithm will compute the best *("maximum likelihood")* conditional probability tables and densities, assuming that any table is valid (i.e., there are no restrictions on the form of the tables). If the contents are not all zeros, then the experience table and the conditional probability tables or density is used to form counts *("prior counts")* that will be added to those derived from the data set. This is known as *"penalized EM"*. If the experience count for a parent configuration is negative, then parameter estimation is disabled for that parent configuration.

The starting point or the EM algorithm is the conditional probability tables and densities specified prior to calling the algorithm, assuming that the domain has been compiled with these tables. If no table has been specified for a discrete node, uniform distributions are assumed whereas initial values for the mean and variances are computed from the data for a continuous node. Sometimes it is desirable to enforce zeros in the conditional probability tables for the configurations that should be impossible (i.e., having zero probability).

The EM algorithm performs a number of iterations. For each iteration the logarithm of the probability of the case data given the current joint probability distribution is computed. This quantity is known as the *log-likelihood*, and the EM-algorithm attempts to maximize this quantity. There are two ways in which the number of iterations can be controlled:

1. A *maximum number of iterations* can be specified (zero means no maximum).
2. If a maximum number of iterations has been specified and not exceeded, the EM algorithm terminates when the relative difference between the log-likelihood for two successive iterations is sufficiently small. That is, when the relative difference between the log-likelihood for two successive iterations becomes less than a *tolerance*, which must be a positive number. By default, the value of the tolerance is 10^4.

To activate EM learning, either press the EM Learning Button or select the *EM Learning Wizard* (page 281) item of the *Wizards Menu* (page 190).

See also notes on *EM learning in OOBNs* (page 348).

4.8.2 Structure Learning

The HUGIN Graphical User Interface provides the user with powerful structure learning capabilities (i.e., learning the structure of a Bayesian-network model from data (a set of cases)). Structure learning can be performed via the *Learning Wizard* (page 267) (which allows data to be read from data files, to be preprocessed, etc.) or by activating one of the structure learning algorithms directly.

A number of algorithms are available for structure learning: *The PC algorithm* (page 246), the *NPC algorithm* (page 246), the *Greedy search-and-score algorithm* (page 258), the *Chow-Liu tree algorithm* (page 254), the *Rebane-Pearl polytree algoritm* (page 253), the *Tree Augmented Naive Bayes algorithm* (page 253) or the *Hierarchical Naive Bayes algorithm* (page 252). See the also *Structure Learning Tutorial* (page 97) for more information.

4.8.3 Auto Filtering (DBN)

With a Dynamic Bayesian network (DBN) loaded HUGIN supports auto filtering from a data file containing a sequence of observations. This is done using the propagation facility in the *data frame* (page 391). To perform Auto Prediction of a DBN start by loading your DBN model. Press Ctrl + O, navigate to the file containing your DBN model and press 'Open' and ensure the model is operating in *run-mode* (page 123).

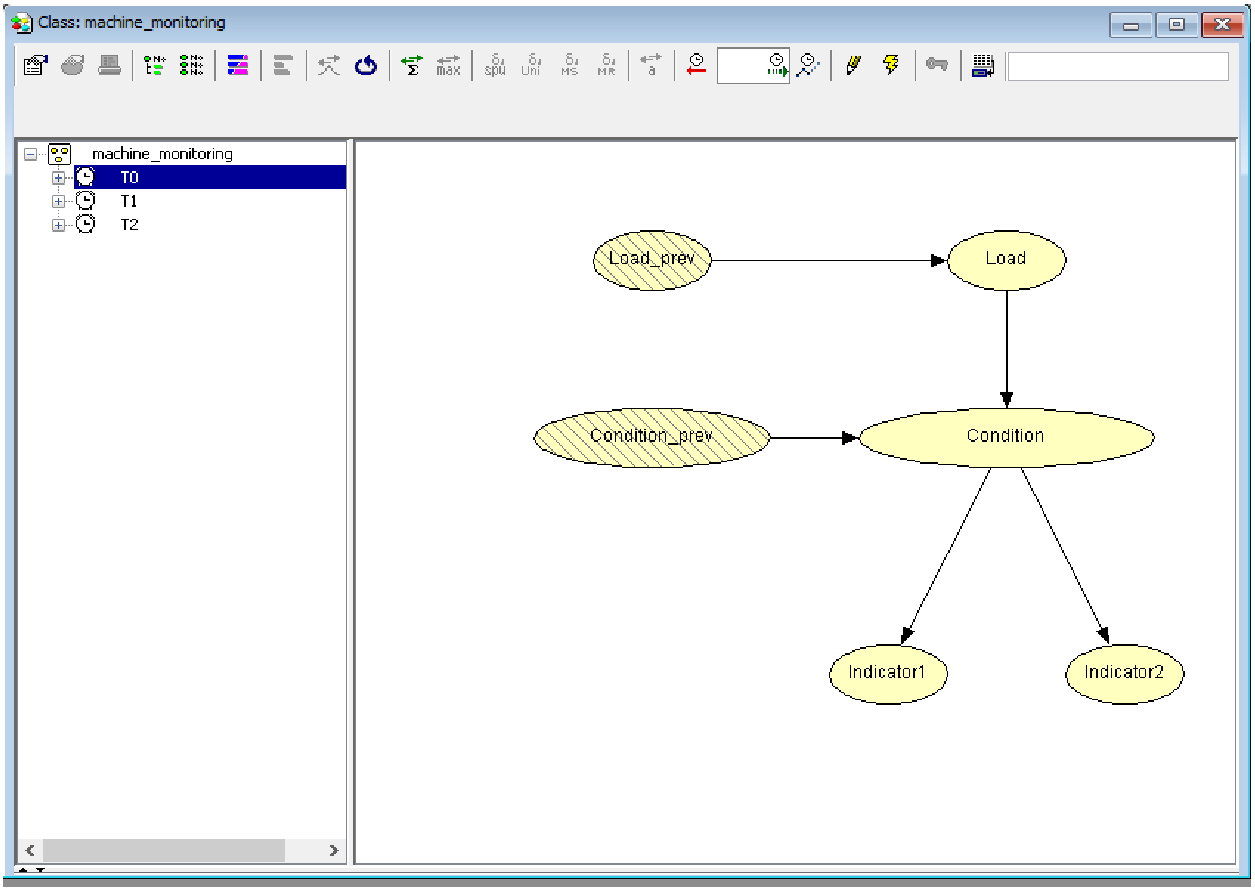
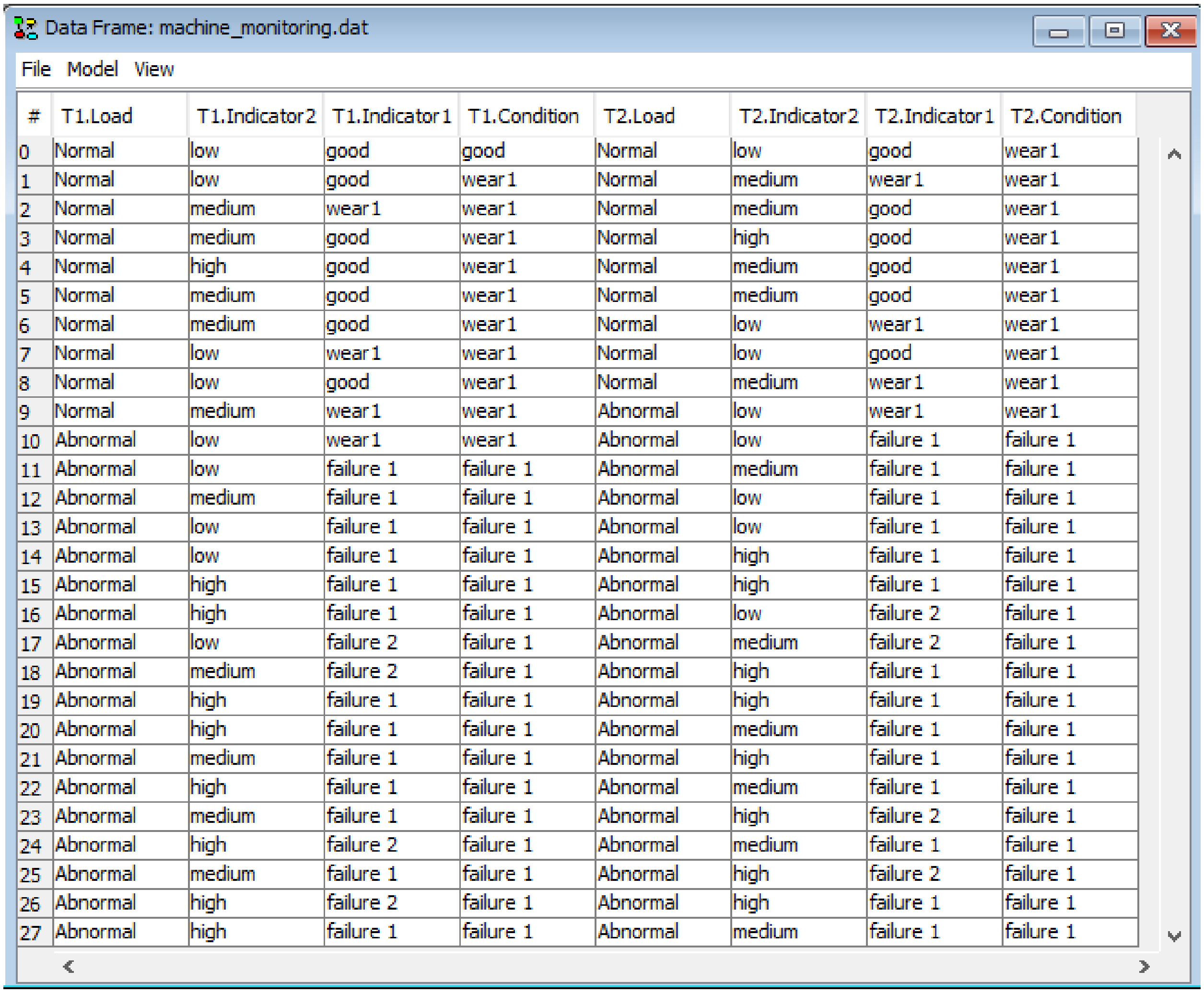


Figure 1: A DBN Model in run-mode. For this example we have loaded *machine_monitoring.oobn* and configured it with 2 time slices (using the *Network Properties Panel* (page 178)).

By following these links you can read how to:

- *open your data file* (page 361),
- *associate you data file with a run-mode DBN model* (page 362),
- *point out the Sequence Identifier if you have multiple sequences in you data file, and* (page 400)
- *convert the data to time windows.* (page 363)

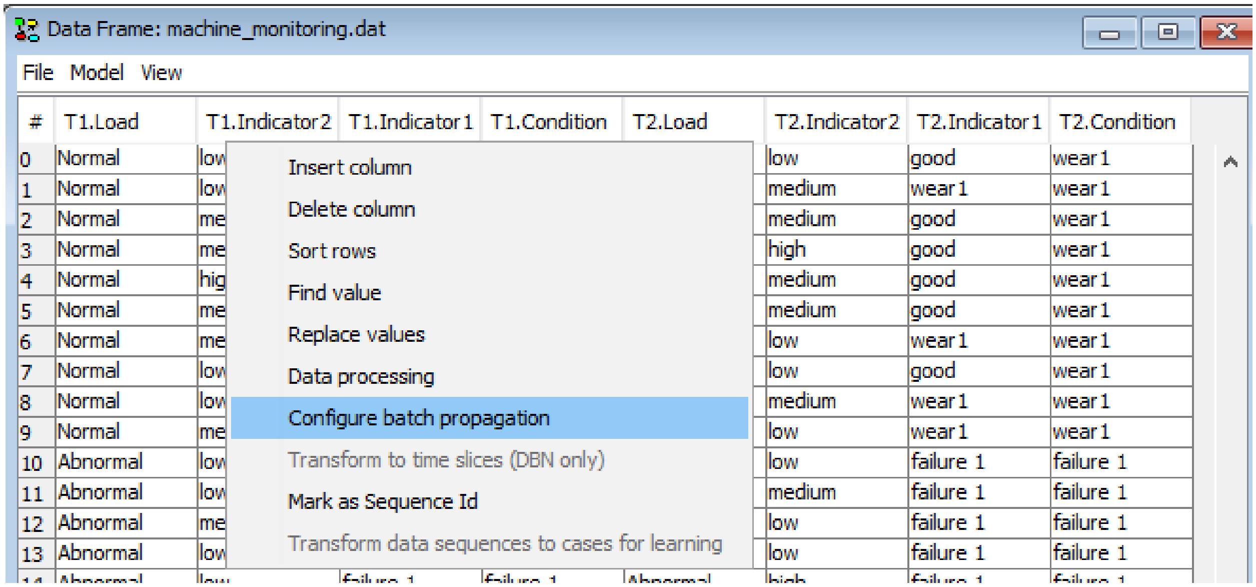
When you have completed these steps your data frame should look something like the content in Figure 2. Here the data file is assumed to contain a single sequence and thus a Sequence Id is not specified.



#	T1.Load	T1.Indicator2	T1.Indicator1	T1.Condition	T2.Load	T2.Indicator2	T2.Indicator1	T2.Condition
0	Normal	low	good	good	Normal	low	good	wear1
1	Normal	low	good	wear1	Normal	medium	wear1	wear1
2	Normal	medium	wear1	wear1	Normal	medium	good	wear1
3	Normal	medium	good	wear1	Normal	high	good	wear1
4	Normal	high	good	wear1	Normal	medium	good	wear1
5	Normal	medium	good	wear1	Normal	medium	good	wear1
6	Normal	medium	good	wear1	Normal	low	wear1	wear1
7	Normal	low	wear1	wear1	Normal	low	good	wear1
8	Normal	low	good	wear1	Normal	medium	wear1	wear1
9	Normal	medium	wear1	wear1	Abnormal	low	wear1	wear1
10	Abnormal	low	wear1	wear1	Abnormal	low	failure 1	failure 1
11	Abnormal	low	failure 1	failure 1	Abnormal	medium	failure 1	failure 1
12	Abnormal	medium	failure 1	failure 1	Abnormal	low	failure 1	failure 1
13	Abnormal	low	failure 1	failure 1	Abnormal	low	failure 1	failure 1
14	Abnormal	low	failure 1	failure 1	Abnormal	high	failure 1	failure 1
15	Abnormal	high	failure 1	failure 1	Abnormal	high	failure 1	failure 1
16	Abnormal	high	failure 1	failure 1	Abnormal	low	failure 2	failure 1
17	Abnormal	low	failure 2	failure 1	Abnormal	medium	failure 2	failure 1
18	Abnormal	medium	failure 2	failure 1	Abnormal	high	failure 1	failure 1
19	Abnormal	high	failure 1	failure 1	Abnormal	high	failure 1	failure 1
20	Abnormal	high	failure 1	failure 1	Abnormal	medium	failure 1	failure 1
21	Abnormal	medium	failure 1	failure 1	Abnormal	high	failure 1	failure 1
22	Abnormal	high	failure 1	failure 1	Abnormal	medium	failure 1	failure 1
23	Abnormal	medium	failure 1	failure 1	Abnormal	high	failure 2	failure 1
24	Abnormal	high	failure 2	failure 1	Abnormal	medium	failure 1	failure 1
25	Abnormal	medium	failure 1	failure 1	Abnormal	high	failure 2	failure 1
26	Abnormal	high	failure 2	failure 1	Abnormal	high	failure 1	failure 1
27	Abnormal	high	failure 1	failure 1	Abnormal	medium	failure 1	failure 1

Figure 2: The content of the data frame when the file machine_monitoring.oobn was loaded and converted to a sequence of time windows.

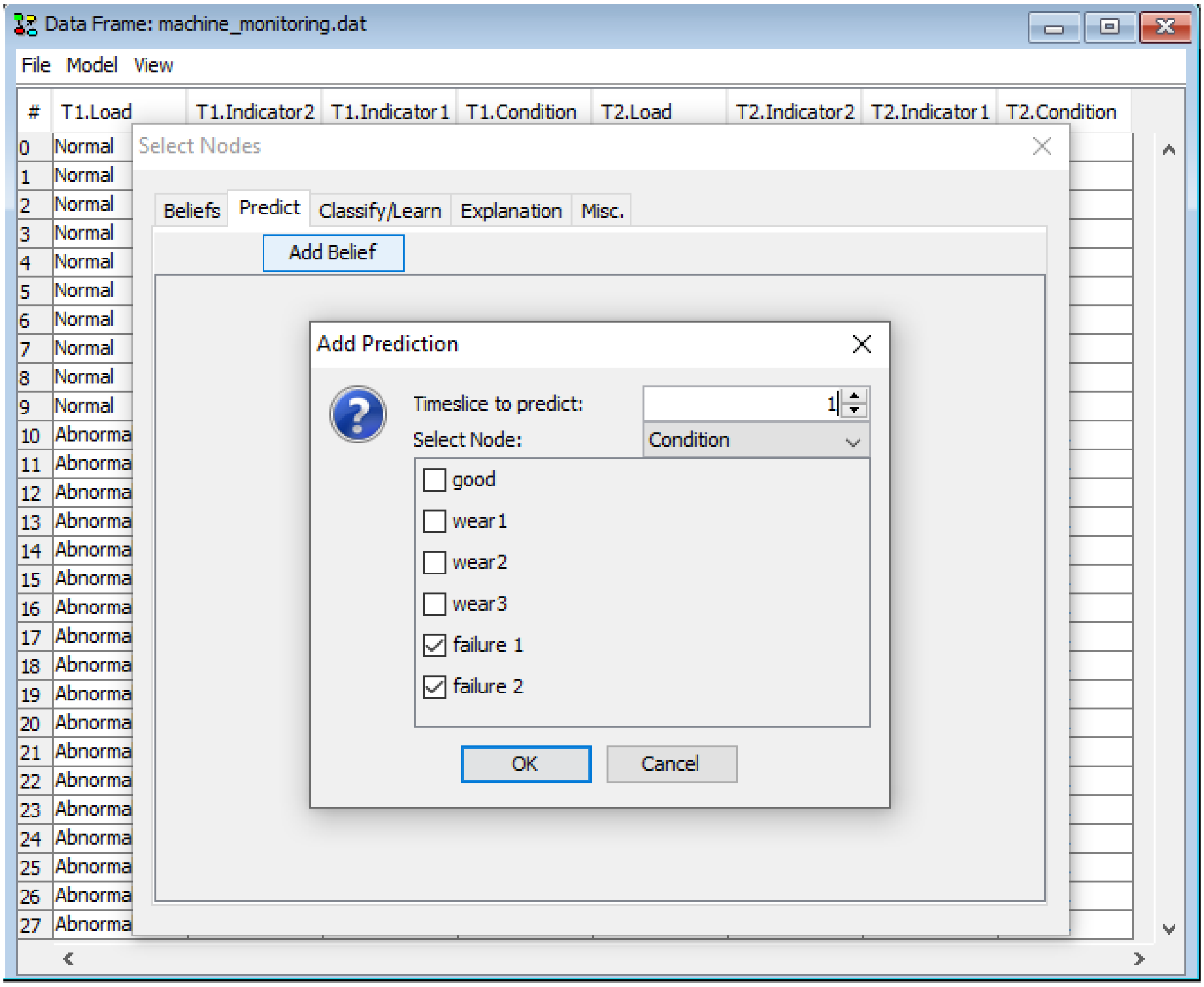
To perform filtering right click on a column header and select ‘Configure batch propagation’ (Figure 3) to access HUGIN’s *Batch propagation configuration menu* (page 392) in the data frame.


Data Frame: machine_monitoring.dat
File
Model
View
Insert column
Delete column
Sort rows
Find value
Replace values
Data processing
Configure batch propagation
Transform to time slices (DBN only)
Mark as Sequence Id
Transform data sequences to cases for learning

#	T1.Load	T1.Indicator2	T1.Indicator1	T1.Condition	T2.Load	T2.Indicator2	T2.Indicator1	T2.Condition
0	Normal	low				low	good	wear 1
1	Normal	low				medium	wear 1	wear 1
2	Normal	me				medium	good	wear 1
3	Normal	me				high	good	wear 1
4	Normal	hig				medium	good	wear 1
5	Normal	me				medium	good	wear 1
6	Normal	me				low	wear 1	wear 1
7	Normal	low				low	good	wear 1
8	Normal	low				medium	wear 1	wear 1
9	Normal	me				low	wear 1	wear 1
10	Abnormal	low				low	failure 1	failure 1
11	Abnormal	low				medium	failure 1	failure 1
12	Abnormal	me				low	failure 1	failure 1
13	Abnormal	low				low	failure 1	failure 1
14	Abnormal	low	failure 1	failure 1	Abnormal	high	failure 1	failure 1

Figure 3: Select Configure batch propagation to enter the batch propagation configuration dialog.

Navigate to the ‘Predict’ tab, click ‘Add Belief’ and configure a new prediction on the Time slice and Variable you wish to perform filtering on. For this example, assume you have the model in Figure 1 and you wish to perform Bayesian filtering on T1.Condition and T2.Condition (states ‘failure 1’ and ‘failure 2’). Then add 2 predictions with Node=Condition, and states ‘failure 1’ and ‘failure 2’ for time slice 1 and 2 respectively. Eventually, you should obtain the screen in Figure 5.



#	T1.Load
0	Normal
1	Normal
2	Normal
3	Normal
4	Normal
5	Normal
6	Normal
7	Normal
8	Normal
9	Normal
10	Abnorma
11	Abnorma
12	Abnorma
13	Abnorma
14	Abnorma
15	Abnorma
16	Abnorma
17	Abnorma
18	Abnorma
19	Abnorma
20	Abnorma
21	Abnorma
22	Abnorma
23	Abnorma
24	Abnorma
25	Abnorma
26	Abnorma
27	Abnorma

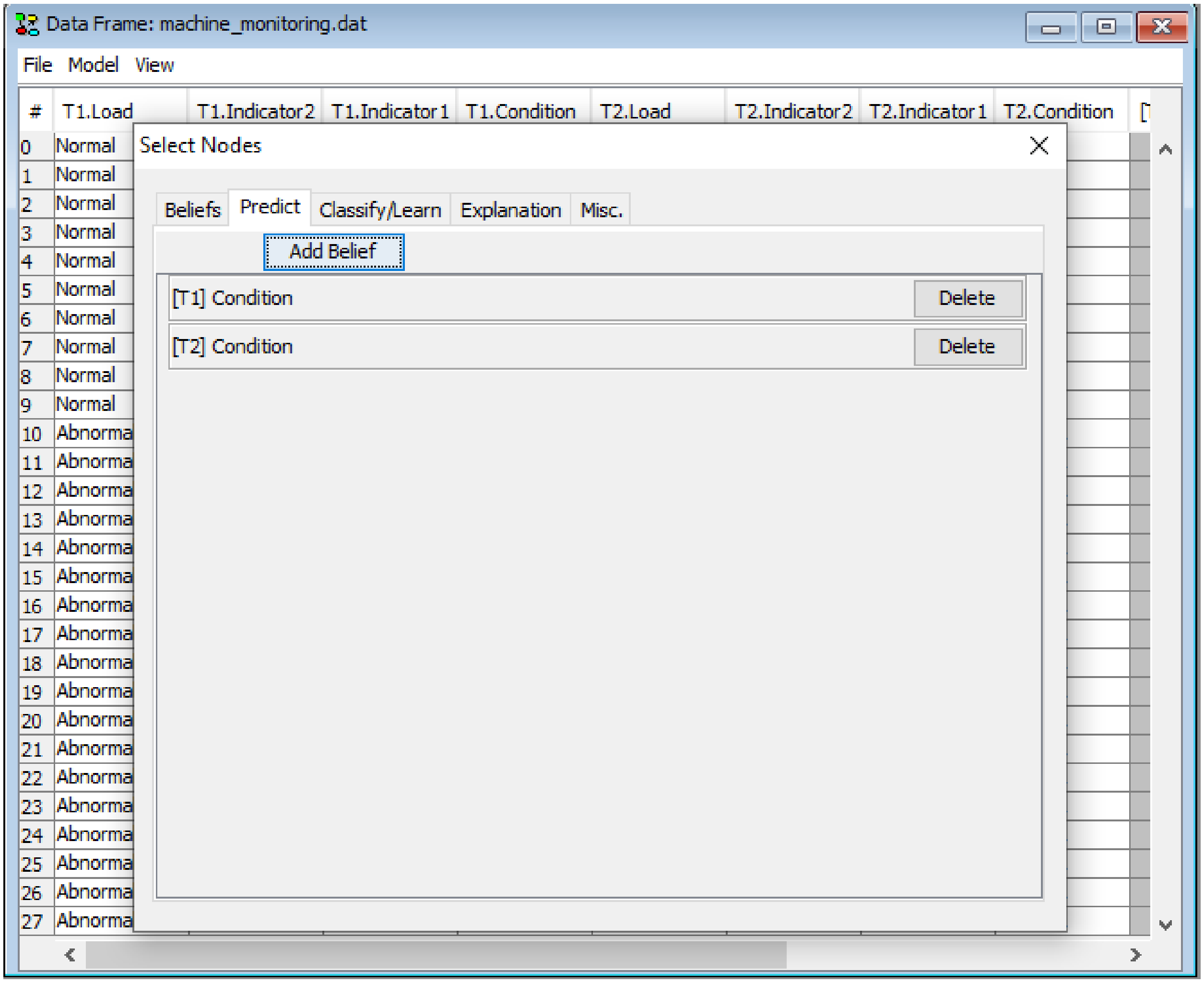
T1.Indicator2 T1.Indicator1 T1.Condition T2.Load T2.Indicator2 T2.Indicator1 T2.Condition

Timeslice to predict: 1

Select Node: Condition

- □ good
- □ wear1
- □ wear2
- □ wear3
- ☑ failure 1
- ☑ failure 2

Figure 4: Configure batch propagation by adding 2 predictions on nodes T1.Condition and T2.Condition.



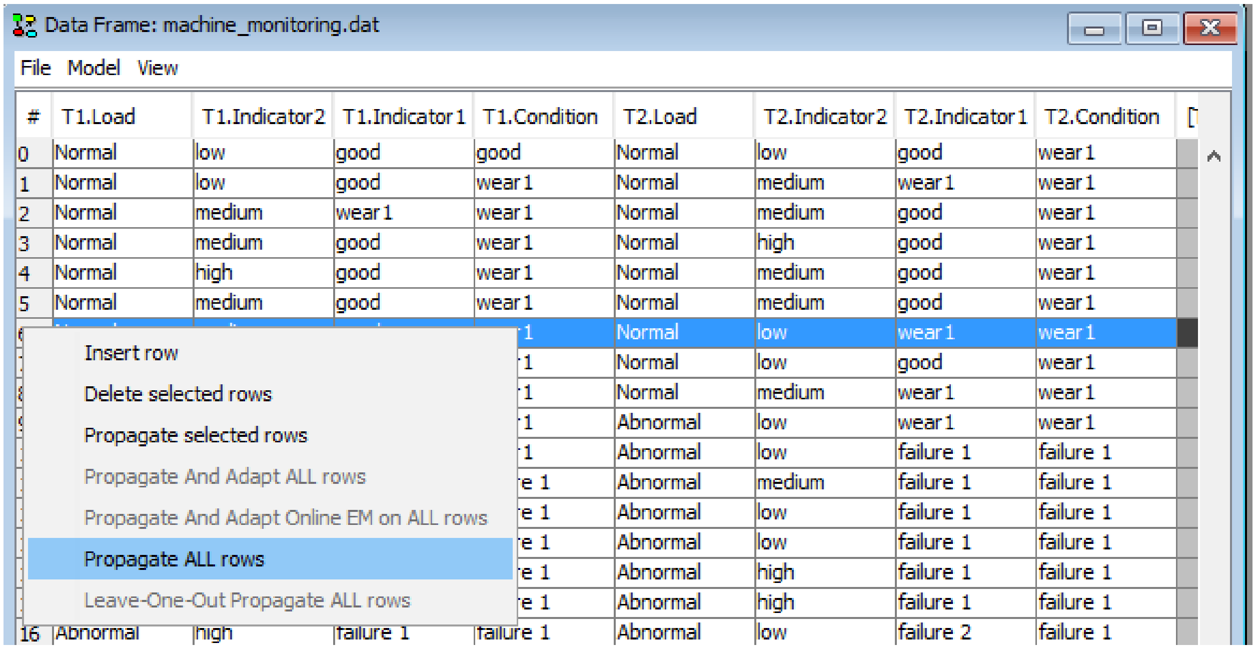
[T1] Condition

[T2] Condition

#	T1.Load	T1.Indicator2	T1.Indicator1	T1.Condition	T2.Load	T2.Indicator2	T2.Indicator1	T2.Condition
0	Normal							
1	Normal							
2	Normal							
3	Normal							
4	Normal							
5	Normal							
6	Normal							
7	Normal							
8	Normal							
9	Normal							
10	Abnorma							
11	Abnorma							
12	Abnorma							
13	Abnorma							
14	Abnorma							
15	Abnorma							
16	Abnorma							
17	Abnorma							
18	Abnorma							
19	Abnorma							
20	Abnorma							
21	Abnorma							
22	Abnorma							
23	Abnorma							
24	Abnorma							
25	Abnorma							
26	Abnorma							
27	Abnorma							

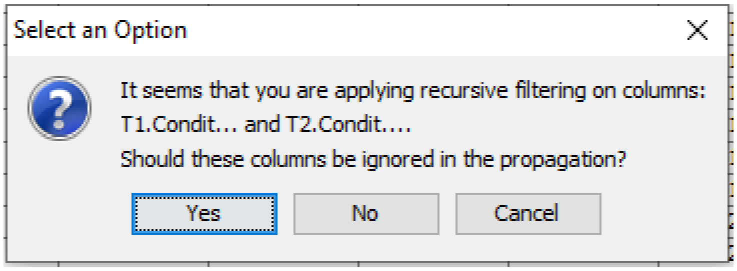
Figure 5: The Batch propagation configuration dialog with the filters added.

To propagate the data, right-click on a row header and select “Propagate ALL rows” or “Propagate selected rows”. To read more about propagation of data in DBN models, *click here!* (page 369) Answer ‘Yes’ when you are asked whether the variables should be ignored in the filtering (Figure 7).


Data Frame: machine_monitoring.dat
File
Model
View
Insert row
Delete selected rows
Propagate selected rows
Propagate And Adapt ALL rows
Propagate And Adapt Online EM on ALL rows
Propagate ALL rows
Leave-One-Out Propagate ALL rows

#	T1.Load	T1.Indicator2	T1.Indicator1	T1.Condition	T2.Load	T2.Indicator2	T2.Indicator1	T2.Condition
0	Normal	low	good	good	Normal	low	good	wear 1
1	Normal	low	good	wear 1	Normal	medium	wear 1	wear 1
2	Normal	medium	wear 1	wear 1	Normal	medium	good	wear 1
3	Normal	medium	good	wear 1	Normal	high	good	wear 1
4	Normal	high	good	wear 1	Normal	medium	good	wear 1
5	Normal	medium	good	wear 1	Normal	medium	good	wear 1
				1	Normal	low	wear 1	wear 1
				1	Normal	low	good	wear 1
				1	Normal	medium	wear 1	wear 1
				1	Abnormal	low	wear 1	wear 1
				1	Abnormal	low	failure 1	failure 1
				e 1	Abnormal	medium	failure 1	failure 1
				e 1	Abnormal	low	failure 1	failure 1
				e 1	Abnormal	low	failure 1	failure 1
				e 1	Abnormal	high	failure 1	failure 1
				e 1	Abnormal	high	failure 1	failure 1
16	Abnormal	high	failure 1	failure 1	Abnormal	low	failure 2	failure 1

Figure 6: Propagate the data by right-clicking on a row header and selecting “Propagate ALL rows”.



It seems that you are applying recursive filtering on columns: T1.Condit... and T2.Condit....
Should these columns be ignored in the propagation?

Figure 7: Answer ‘Yes’ when you are asked whether the variables should be ignored in the filtering.

You can use HUGIN’s Classifier Performance tool to get an indication of how well the model has performed. Right-click in a cell and select ‘Classifier Performance’.

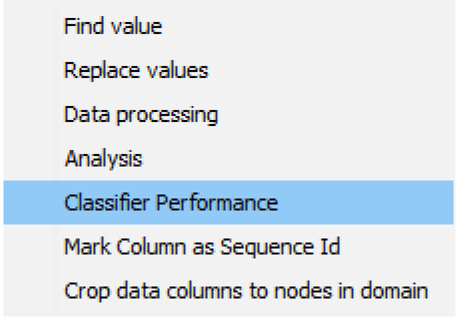


Figure 8: To evaluate how well the variable was predicted, right-click in a cell and select ‘Classifier Performance’

- As **‘Actual Class’** select one of the columns you renamed previously to leave them out of the batch propagation.
- As **‘Target Class’** select one of the states in the ‘Actual Class’ column that you just selected.

- **As 'P(Target Class')** select the column which contains the propagated belief of the 'Target Class' you just selected.

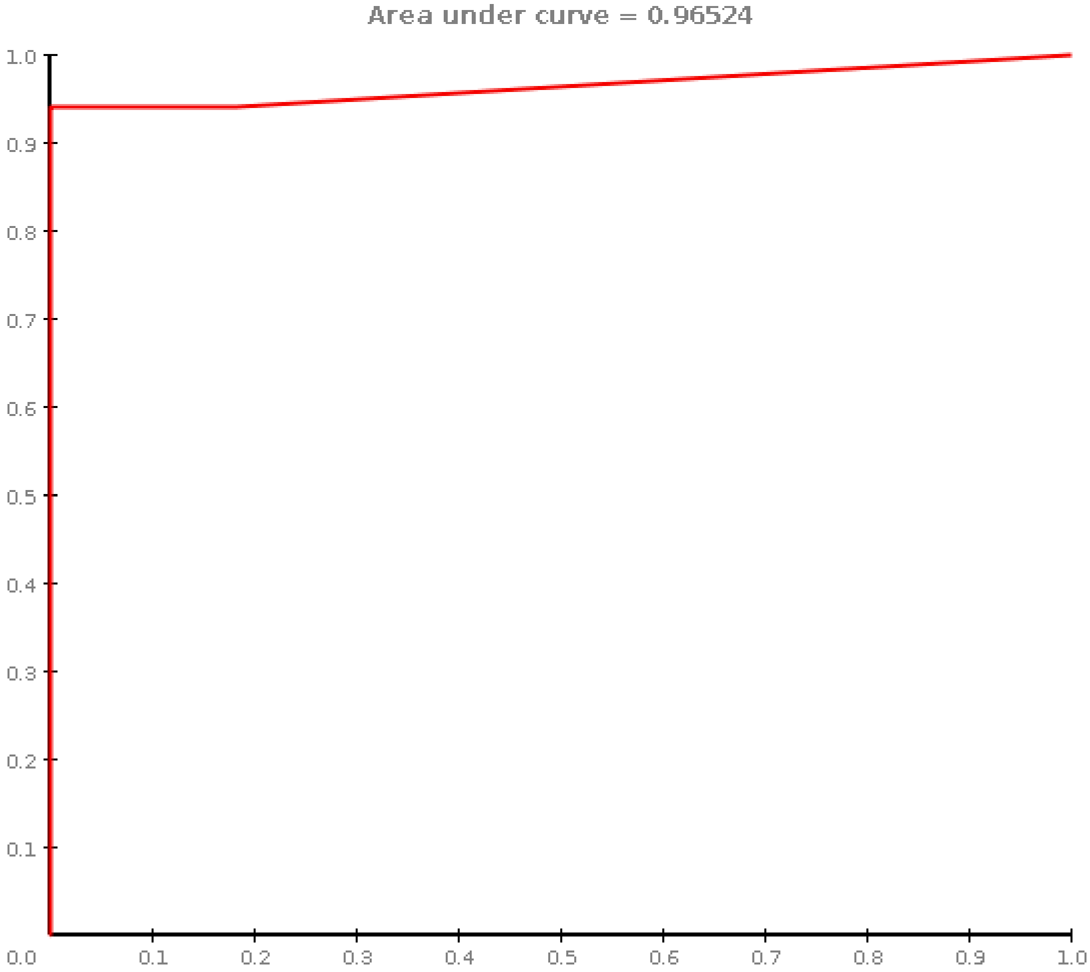


Figure 9: ROC curve indicating the performance of the Bayesian filter on the node T1.Condition.

4.8.4 Auto Prediction (DBN)

With a Dynamic Bayesian network (DBN) loaded HUGIN has the ability to perform auto prediction from a data file containing a sequence of observations. This is done using the propagation facility in the *data frame* (page 391). To perform Auto Prediction of a DBN start by loading your DBN model. Press Ctrl + O, navigate to the file containing your DBN model and press 'Open' and ensure the model is operating in *run-mode* (page 123).

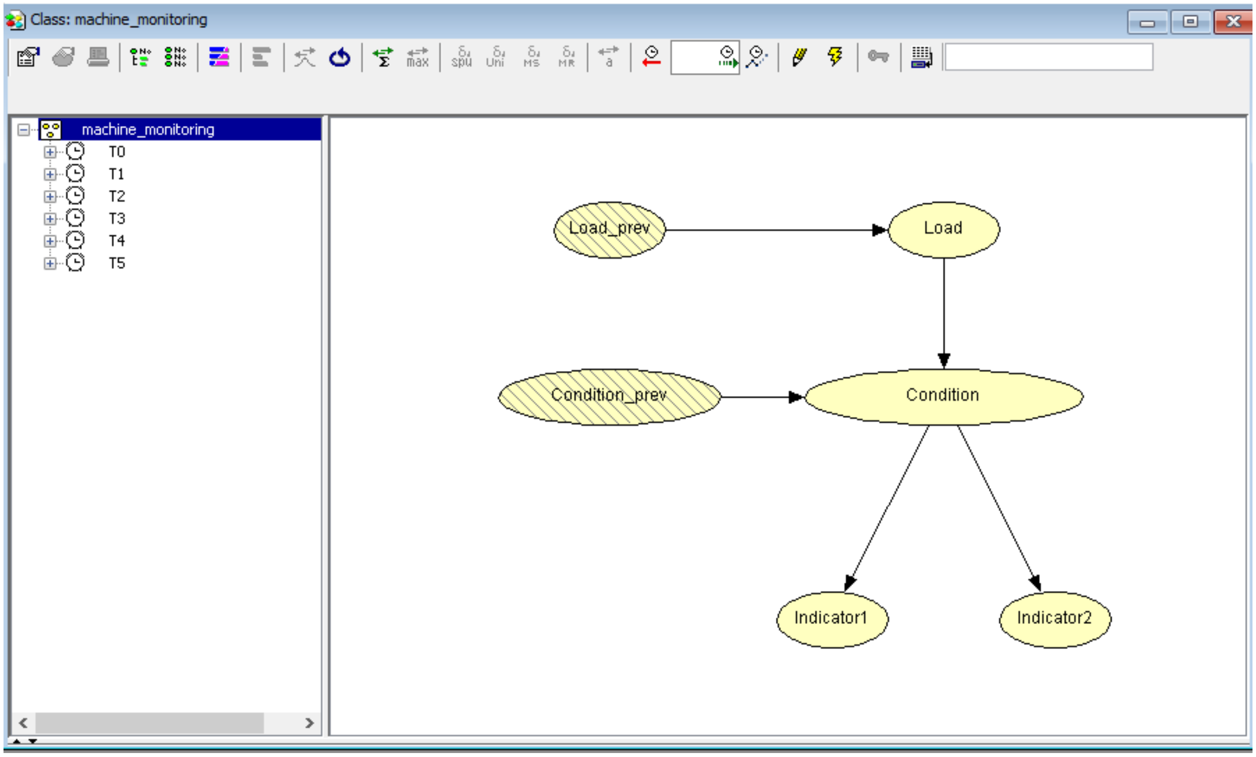
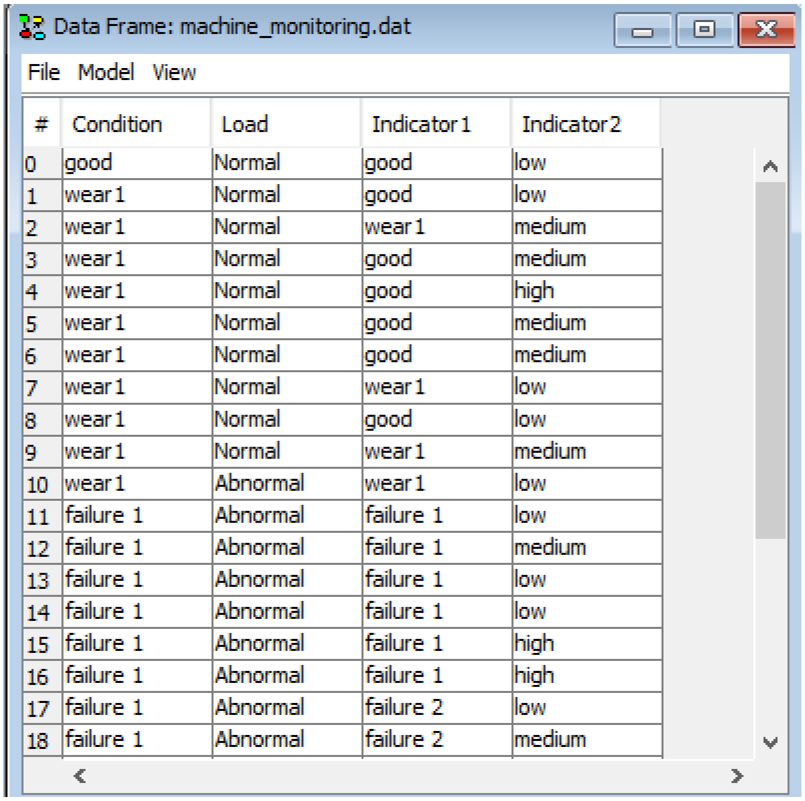


Figure 1: A DBN Model in run-mode.

Now **open your data file** containing your data sequence. To open a new data frame and load a data file, click in the menu of the main HUGIN frame File -> Open Data File and select a data file. The file is assumed to be in csv format.

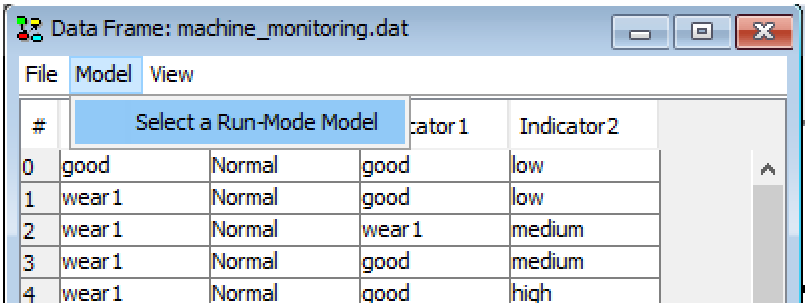


#	Condition	Load	Indicator1	Indicator2
0	good	Normal	good	low
1	wear1	Normal	good	low
2	wear1	Normal	wear1	medium
3	wear1	Normal	good	medium
4	wear1	Normal	good	high
5	wear1	Normal	good	medium
6	wear1	Normal	good	medium
7	wear1	Normal	wear1	low
8	wear1	Normal	good	low
9	wear1	Normal	wear1	medium
10	wear1	Abnormal	wear1	low
11	failure 1	Abnormal	failure 1	low
12	failure 1	Abnormal	failure 1	medium
13	failure 1	Abnormal	failure 1	low
14	failure 1	Abnormal	failure 1	low
15	failure 1	Abnormal	failure 1	high
16	failure 1	Abnormal	failure 1	high
17	failure 1	Abnormal	failure 2	low
18	failure 1	Abnormal	failure 2	medium

Figure 2: The Data sequence loaded in HUGIN's Data Frame.

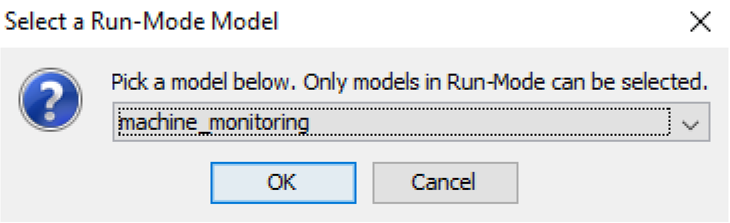
Note the auto prediction functionallity is only available in the dataframe if it has been associated with a DBN model in run-mode!

To associate the data frame with your DBN Model in *run-mode* (page 123), click the *Model* menu and then click Select a *Run-Mode Model*. A dialog appears, containing all the current project frames to pick from.



#			ator1	Indicator2
0	good	Normal	good	low
1	wear1	Normal	good	low
2	wear1	Normal	wear1	medium
3	wear1	Normal	good	medium
4	wear1	Normal	good	high

Figure 3: To associate the data sequence with the DBN model select 'Select Run-Mode Model' form the 'Model' menu.



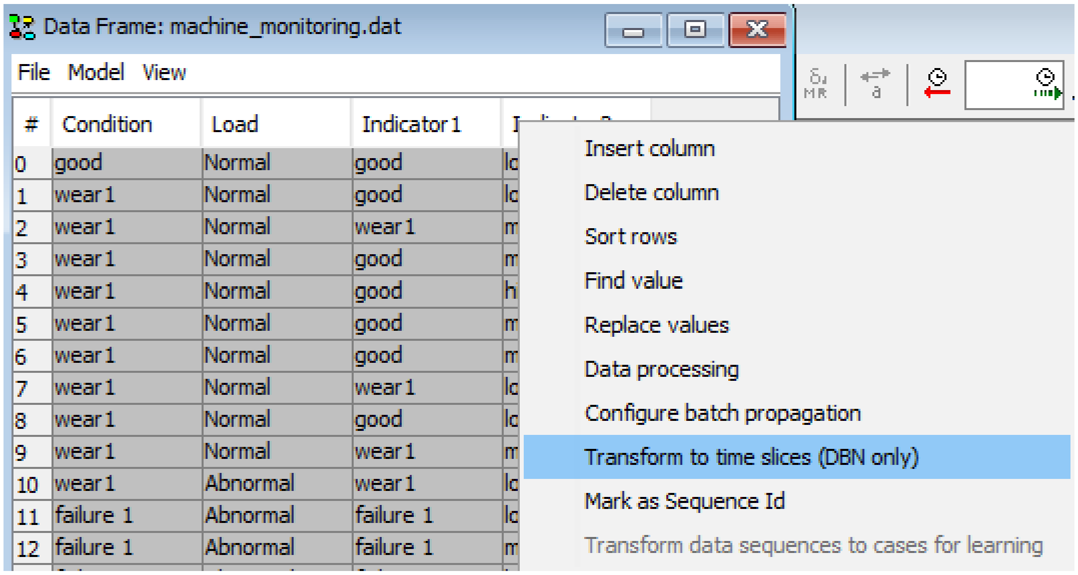
Select a Run-Mode Model

Pick a model below. Only models in Run-Mode can be selected.

machine_monitoring

Figure 4: Pick a project for doing the DBN auto Prediction.

Convert your data file to a sequence of time windows if the data you have loaded is not already in the *auto prediction time window format* (page 364). To do this, right click a column header in the data matrix and select **Transform to time slices (DBN only)**.



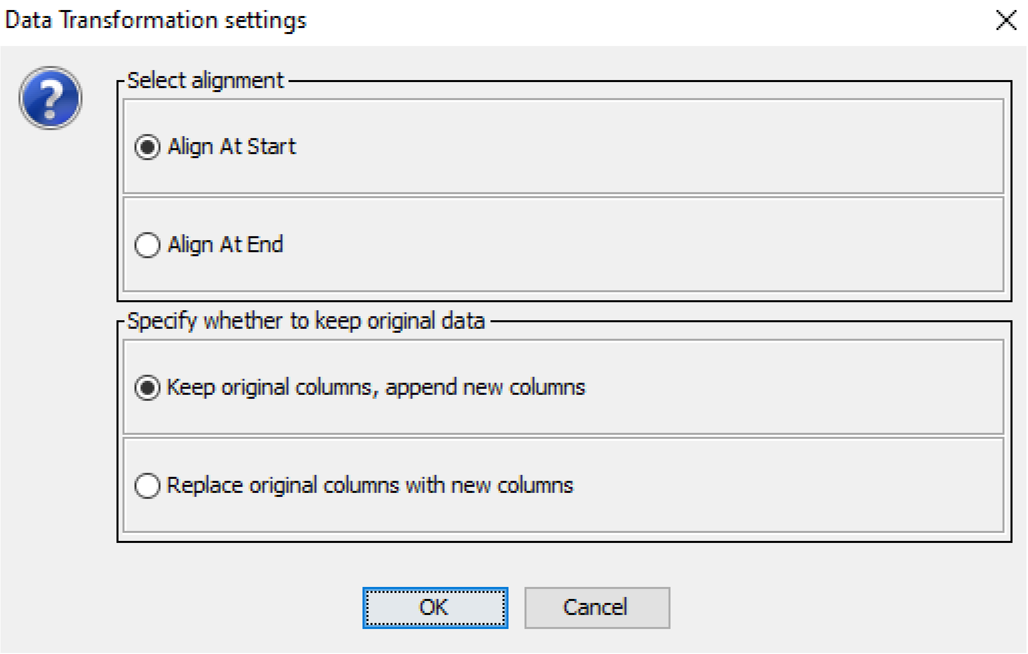
Data Frame: machine_monitoring.dat

File
Model
View
Insert column
Delete column
Sort rows
Find value
Replace values
Data processing
Configure batch propagation
Transform to time slices (DBN only)
Mark as Sequence Id
Transform data sequences to cases for learning

#	Condition	Load	Indicator1	I
0	good	Normal	good	lc
1	wear1	Normal	good	lc
2	wear1	Normal	wear1	m
3	wear1	Normal	good	m
4	wear1	Normal	good	h
5	wear1	Normal	good	m
6	wear1	Normal	good	m
7	wear1	Normal	wear1	lc
8	wear1	Normal	good	lc
9	wear1	Normal	wear1	m
10	wear1	Abnormal	wear1	lc
11	failure 1	Abnormal	failure 1	lc
12	failure 1	Abnormal	failure 1	m

Figure 5: Select configure batch propagation *(Note: if this option is dimmed out, then switch the associated network model to run-mode first)*.

The dialog in Figure 6 will appear asking you how you wish to align your data and whether you wish to keep or replace the original columns in the dataset.



Data Transformation settings

Select alignment

● Align At Start

○ Align At End

Specify whether to keep original data

● Keep original columns, append new columns

○ Replace original columns with new columns

Figure 6: A dialog will appear asking how to align the original data in the time slices and whether the original columns should be replaced.

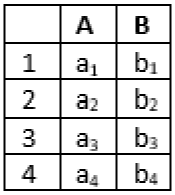
When the tool performs a transformation it consults the associated run-mode model when it looks for the following parameters:

- *Time Slices*: The number of time slices in the run-mode DBN model determines the time horizon of the transformed data set.
- *Node Names*: The column headers in the original time series data must be equal to the corresponding the node names in the run-mode DBN model. Columns without a corresponding node name will be ignored.

Let T be the number of Time Slices configured in the associated run-mode DBN and N be the number of columns for which the column header equals a node name in the associated run-mode DBN and finally, let the columns in the original time series data with matching nodes in the associated model be labeled n_1, n_2, … n_N. Then the process generates $T * N$ columns as following: "T1.n_1", "T1.n_2",…"T1.n_N", "T2.n_1",…, "TT_N".

The data in each case is transformed from a time series to a sequence of horizon time windows with horizon T.

Thus, if we for instance have a datafile with a time series over the variables A and B as follows:



	A	B
1	a_1	b_1
2	a_2	b_2
3	a_3	b_3
4	a_4	b_4

If the associated run-mode model is a DBN with 2 time slices over the variables A and B, the conversion produces:

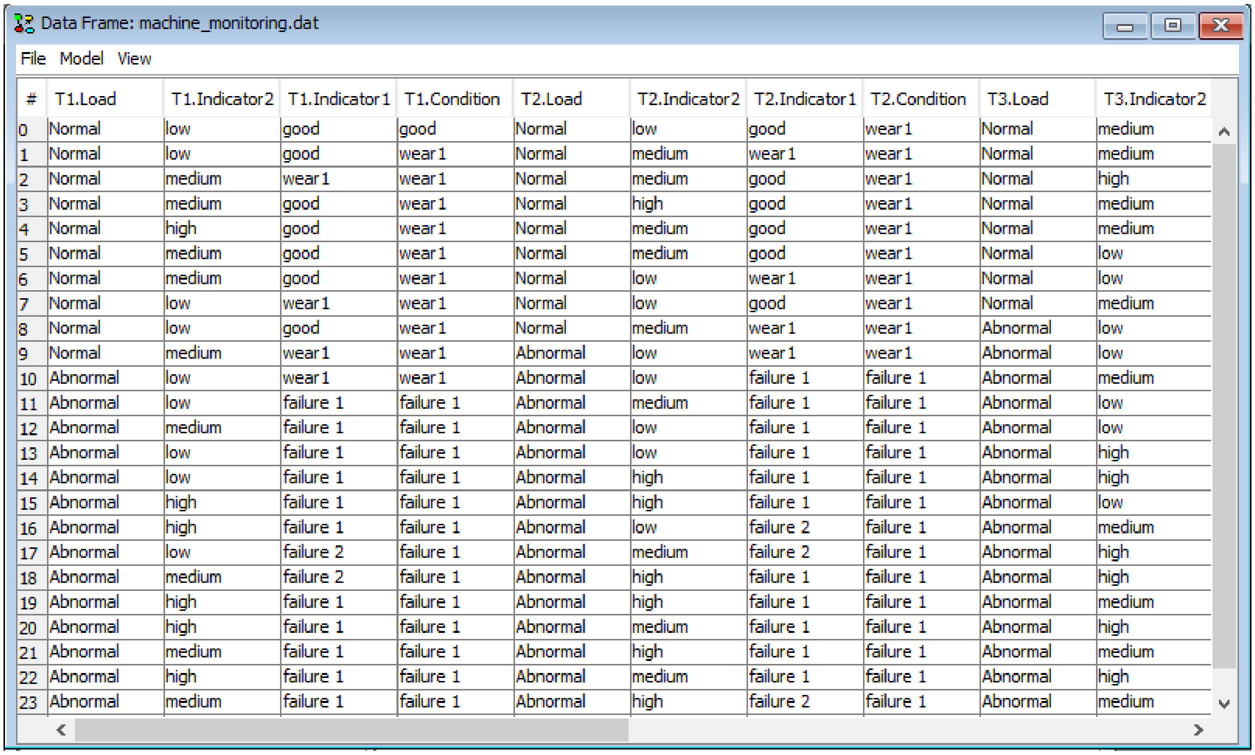
- If "Align at Start" has been choosen:

	T1.A	T1.B	T2.A	T2.B
1	a_1	b_1	a_2	b_2
2	a_2	b_2	a_3	b_3
3	a_3	b_3	a_4	b_4
4	a_4	b_4		

- If “Align at End” has been choosen:

	T1.A	T1.B	T2.A	T2.B
1			a_1	b_1
2	a_1	b_1	a_2	b_2
3	a_2	b_2	a_3	b_3
4	a_3	b_3	a_4	b_4

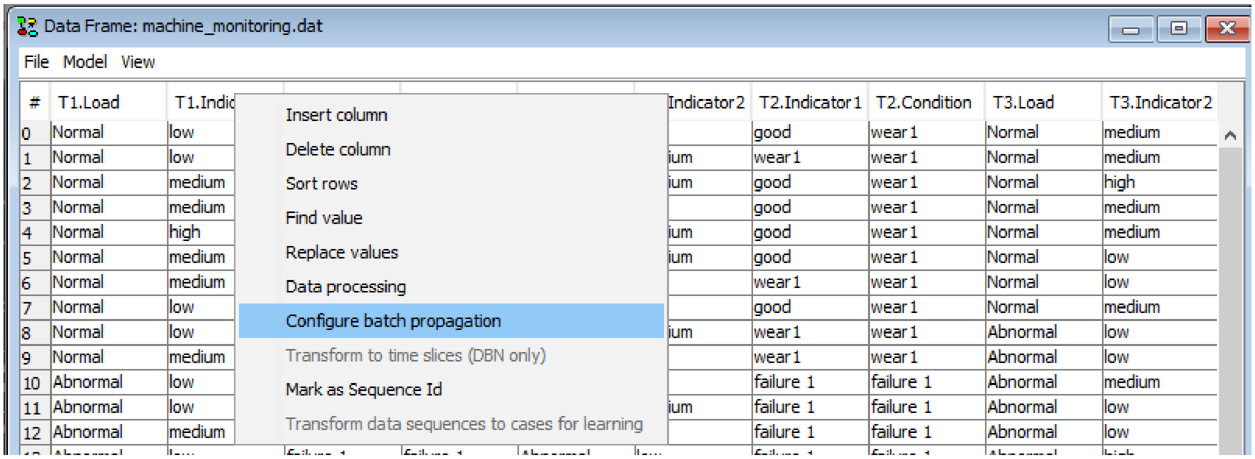
The result of the transformation in the machine_monitoring.dat and machine_monitoring.oobn files which can be found in the example folder is outlined in Figure 7.



#	T1.Load	T1.Indicator2	T1.Indicator1	T1.Condition	T2.Load	T2.Indicator2	T2.Indicator1	T2.Condition	T3.Load	T3.Indicator2
0	Normal	low	good	good	Normal	low	good	wear1	Normal	medium
1	Normal	low	good	wear1	Normal	medium	wear1	wear1	Normal	medium
2	Normal	medium	wear1	wear1	Normal	medium	good	wear1	Normal	high
3	Normal	medium	good	wear1	Normal	high	good	wear1	Normal	medium
4	Normal	high	good	wear1	Normal	medium	good	wear1	Normal	medium
5	Normal	medium	good	wear1	Normal	medium	good	wear1	Normal	low
6	Normal	medium	good	wear1	Normal	low	wear1	wear1	Normal	low
7	Normal	low	wear1	wear1	Normal	low	good	wear1	Normal	medium
8	Normal	low	good	wear1	Normal	medium	wear1	wear1	Abnormal	low
9	Normal	medium	wear1	wear1	Abnormal	low	wear1	wear1	Abnormal	low
10	Abnormal	low	wear1	wear1	Abnormal	low	failure 1	failure 1	Abnormal	medium
11	Abnormal	low	failure 1	failure 1	Abnormal	medium	failure 1	failure 1	Abnormal	low
12	Abnormal	medium	failure 1	failure 1	Abnormal	low	failure 1	failure 1	Abnormal	low
13	Abnormal	low	failure 1	failure 1	Abnormal	low	failure 1	failure 1	Abnormal	high
14	Abnormal	low	failure 1	failure 1	Abnormal	high	failure 1	failure 1	Abnormal	high
15	Abnormal	high	failure 1	failure 1	Abnormal	high	failure 1	failure 1	Abnormal	low
16	Abnormal	high	failure 1	failure 1	Abnormal	low	failure 2	failure 1	Abnormal	medium
17	Abnormal	low	failure 2	failure 1	Abnormal	medium	failure 2	failure 1	Abnormal	high
18	Abnormal	medium	failure 2	failure 1	Abnormal	high	failure 1	failure 1	Abnormal	high
19	Abnormal	high	failure 1	failure 1	Abnormal	high	failure 1	failure 1	Abnormal	medium
20	Abnormal	high	failure 1	failure 1	Abnormal	medium	failure 1	failure 1	Abnormal	high
21	Abnormal	medium	failure 1	failure 1	Abnormal	high	failure 1	failure 1	Abnormal	medium
22	Abnormal	high	failure 1	failure 1	Abnormal	medium	failure 1	failure 1	Abnormal	high
23	Abnormal	medium	failure 1	failure 1	Abnormal	high	failure 2	failure 1	Abnormal	medium

Figure 7: Outline of the result from transforming machine_monitoring.dat with machine_monitoring.oobn loaded as DBN run-mode model.

To configure which variable to target for prediction right-click on a column header in the data frame window and select ‘Configure batch propagation’.


Data Frame: machine_monitoring.dat
File Model View
Insert column
Delete column
Sort rows
Find value
Replace values
Data processing
Configure batch propagation
Transform to time slices (DBN only)
Mark as Sequence Id
Transform data sequences to cases for learning

#	T1.Load	T1.Indic	Indicator2	T2.Indicator1	T2.Condition	T3.Load	T3.Indicator2
0	Normal	low		good	wear1	Normal	medium
1	Normal	low	ium	wear1	wear1	Normal	medium
2	Normal	medium	ium	good	wear1	Normal	high
3	Normal	medium		good	wear1	Normal	medium
4	Normal	high	ium	good	wear1	Normal	medium
5	Normal	medium	ium	good	wear1	Normal	low
6	Normal	medium		wear1	wear1	Normal	low
7	Normal	low		good	wear1	Normal	medium
8	Normal	low	ium	wear1	wear1	Abnormal	low
9	Normal	medium		wear1	wear1	Abnormal	low
10	Abnormal	low		failure 1	failure 1	Abnormal	medium
11	Abnormal	low	ium	failure 1	failure 1	Abnormal	low
12	Abnormal	medium		failure 1	failure 1	Abnormal	low

Figure 8: Select 'Configure batch propagation' in the table header menu in order to specify which variables to predict.

In the Select Nodes dialog navigate to the 'Predict' tab and click 'Add Prediction' to specify a new variable for prediction.

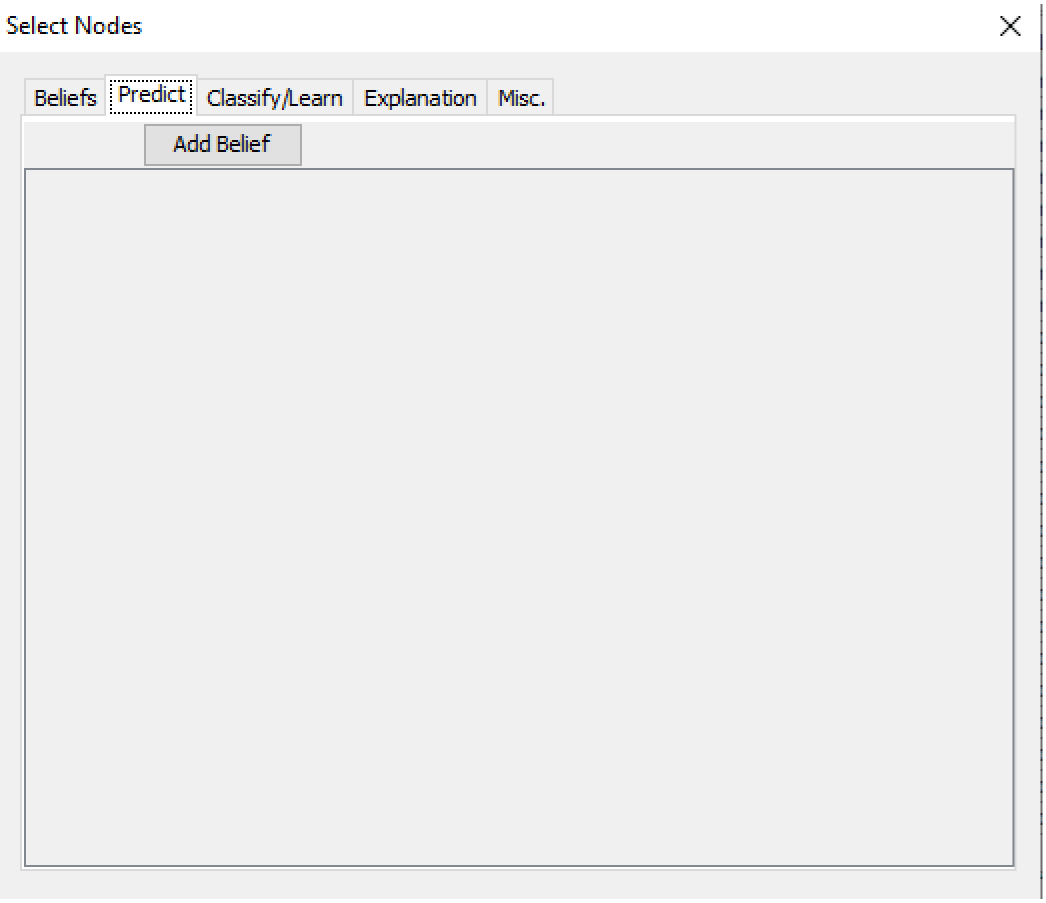
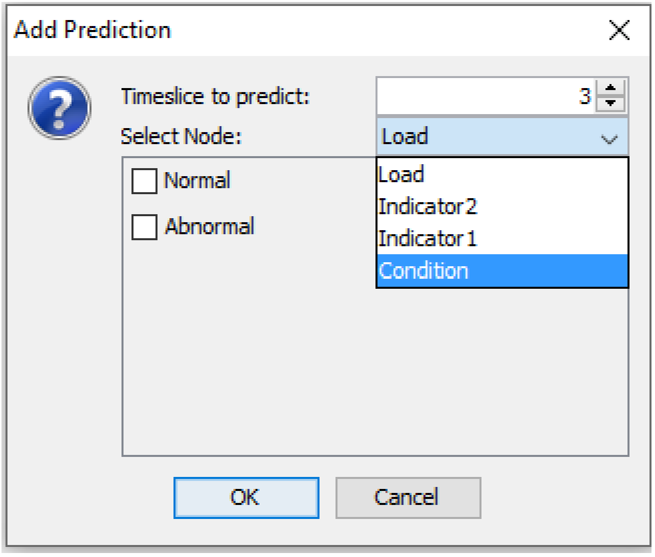


Figure 9: Click 'Add Prediction' to specify a new variable for prediction.

In the 'Add Prediction' dialog, specify which variable and which timeslot you wish to predict.

Discrete valued chance nodes, Continuous chance nodes, Discrete Function Nodes, and Continuous Function Nodes can be selected for prediction.



Add Prediction

Timeslice to predict: 3

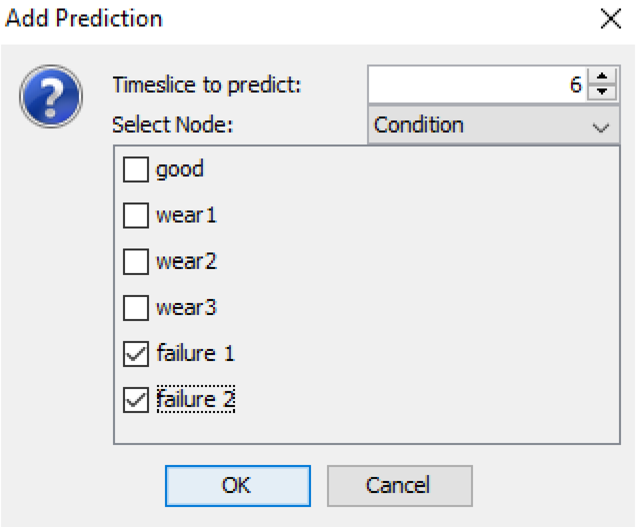
Select Node: Load

□ Normal
□ Abnormal

Figure 10: Select time slice, node and states which should be marked for prediction.

When a variable has been selected, further parameters can be selected for the prediction. If the node is a discrete chance node or a discrete function node the desired states can be marked for prediction. **To mark all states** select the variable twice in the 'Select Node' drop down menu.

Note: As soon as you click 'OK' your choices will be stored in the dataframe. A new column will be added for each selected state. (See *'How Predictions are stored in the dataframe'* (page 368) to understand how this works).



Add Prediction

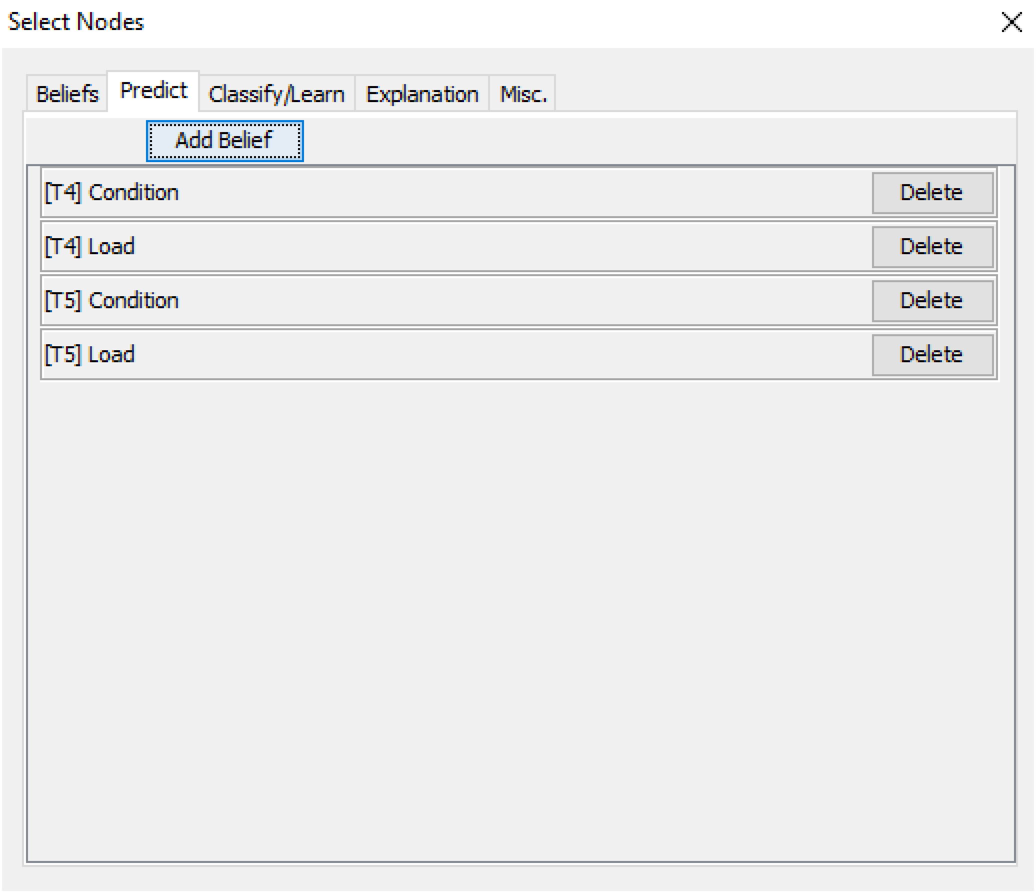
Timeslice to predict: 6

Select Node: Condition

□ good
□ wear1
□ wear2
□ wear3
☑ failure 1
☑ failure 2

Figure 11: Press 'OK' to store your selection and corresponding columns will be added to the data frame.

You can add as many Prediction elements as you want. If you add a new prediction element with the same variable and time slice as an existing prediction element the selected states will be 'absorbed' by the existing element and a new element will not be added to the list but new columns will be added for each new selected state (See *'Predictions are stored in the dataframe'* (page 368)).



[T4] Condition

[T4] Load

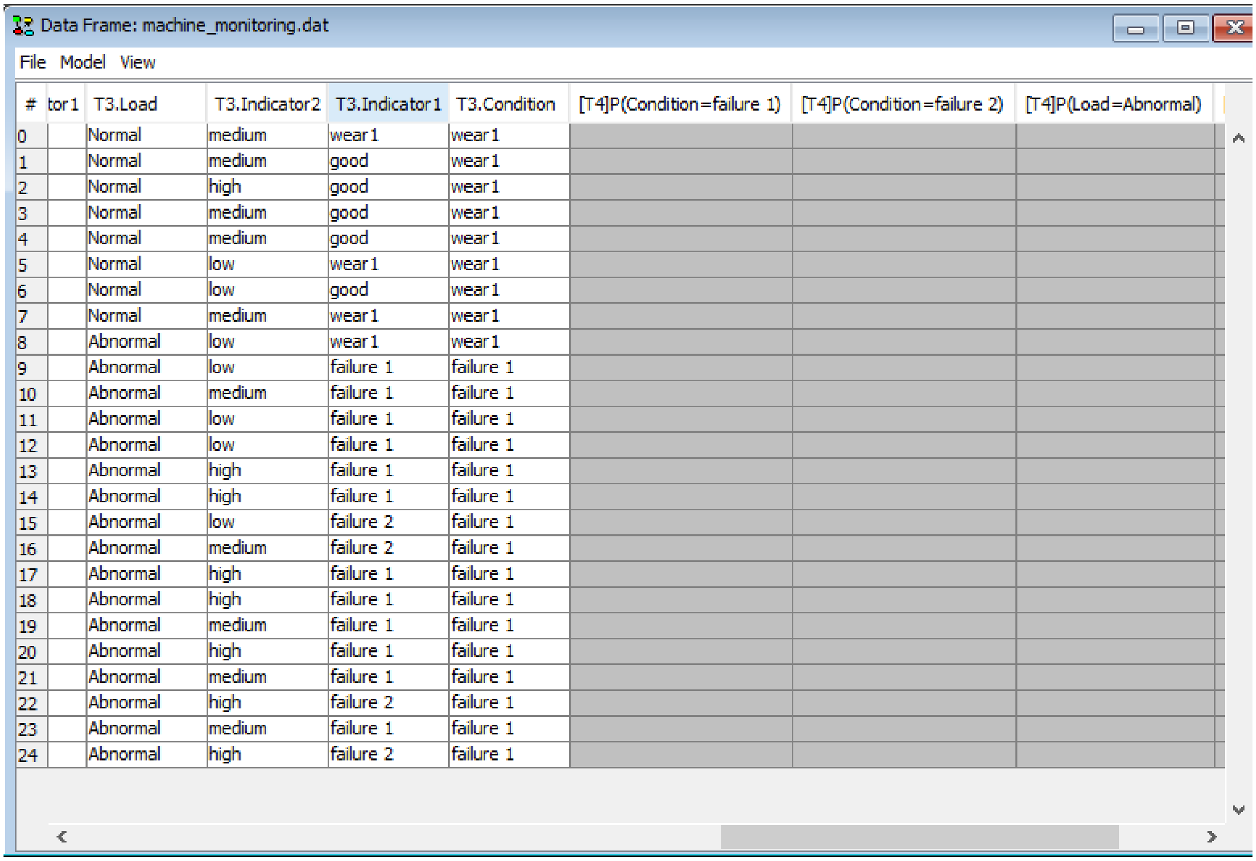
[T5] Condition

[T5] Load

Figure 12: You may add as many prediction elements as you like. The predictions you select are immediately added as new columns in the data frame.

Predictions are stored as columns in the dataframe. Assume that each individual state s1 , s2 , … sn for node N in timeslot t are marked for predicition using the 'Add Prediction' dialog. Then a columns with a headers [T_t]P(N=s_2), [T_t]P(N=s_2),…[T_t]P(N=s_n) are added to the data in the data frame.

An example excerpt of the the machine_monitoring.dat and machine_monitoring.oobn examples from the HUGIN Example folder with states selected for prediction is outlined in Figure 13.



#	tor1	T3.Load	T3.Indicator2	T3.Indicator1	T3.Condition	[T4]P(Condition=failure 1)	[T4]P(Condition=failure 2)	[T4]P(Load=Abnormal)
0		Normal	medium	wear1	wear1			
1		Normal	medium	good	wear1			
2		Normal	high	good	wear1			
3		Normal	medium	good	wear1			
4		Normal	medium	good	wear1			
5		Normal	low	wear1	wear1			
6		Normal	low	good	wear1			
7		Normal	medium	wear1	wear1			
8		Abnormal	low	wear1	wear1			
9		Abnormal	low	failure 1	failure 1			
10		Abnormal	medium	failure 1	failure 1			
11		Abnormal	low	failure 1	failure 1			
12		Abnormal	low	failure 1	failure 1			
13		Abnormal	high	failure 1	failure 1			
14		Abnormal	high	failure 1	failure 1			
15		Abnormal	low	failure 2	failure 1			
16		Abnormal	medium	failure 2	failure 1			
17		Abnormal	high	failure 1	failure 1			
18		Abnormal	high	failure 1	failure 1			
19		Abnormal	medium	failure 1	failure 1			
20		Abnormal	high	failure 1	failure 1			
21		Abnormal	medium	failure 1	failure 1			
22		Abnormal	high	failure 2	failure 1			
23		Abnormal	medium	failure 1	failure 1			
24		Abnormal	high	failure 2	failure 1			

Figure 13: An example excerpt of the the machine_monitoring.dat and machine_monitoring.oobn examples from the HUGIN Example folder with states selected for prediction.

To perform the propagation of the selected timeslots, variables and states right click on a row-header and you now have two possibilities:

- Select either '**Propagate selected rows**' to propagate either the selected row or a subset of rows in sequence. Prior to this choice, the rows you wish to propagate must be selected with the mouse: Hold down left mouse button over the first or last row you wish to select and pull either up or down,
- or select '**Propagate ALL rows**' to propagate all cases stored in the data frame in sequence.

Note: The order in which the rows are propagated matters. Each time a row is propagated, the time window offset is advanced with 1 time step in the associated run-mode DBN model and all evidence will be accumulated. This corresponds with entering the evidence in the model window and pressing (move time window) which will affect the result of all subsequent propagations. If you start a propagation of any number of rows and the time window offset is not 0 you will see the warning in Figure 16 but the system will proceed with the prediction. This allow interleaved batch propagation of subsets of cases in the data frame and manual interaction with the model for inspecting the propability distributions in the model at a particular state.

If this is not desired, it is highly recommended to press (compile) which will reset the time window offset to 0 and then use the 'Propagate ALL rows' option.

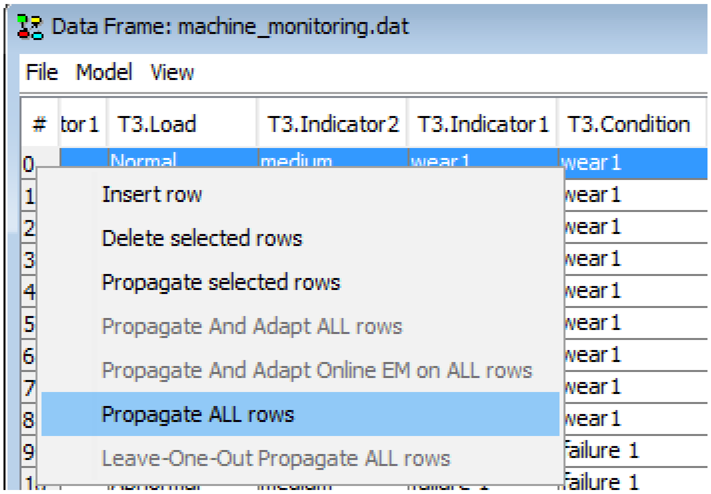
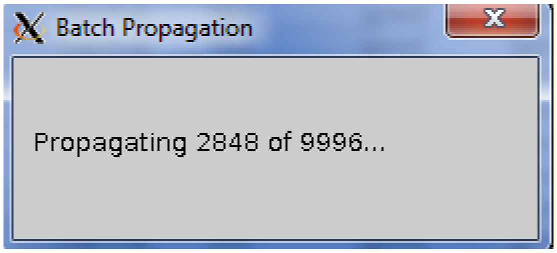


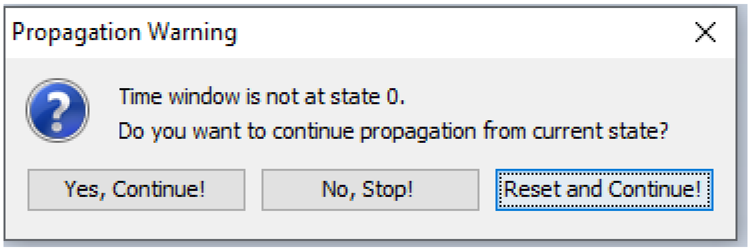
Figure 14: The recommended way to perform propagation is to press ⚡ (compile) in the run-mode DBN model which will reset the time window offset to 0 and then use the 'Propagate ALL rows' option.

During the batch propagation you will see the progress. To stop the propagation simply close the progress window.



Propagating 2848 of 9996...

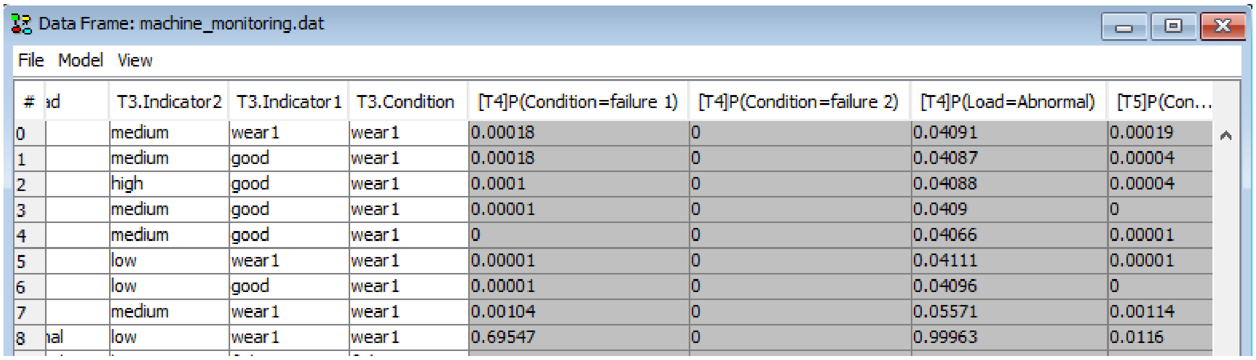
Figure 15: The progress window shown during the batch propagation. To stop the propagation simply close the progress window.



Time window is not at state 0.
Do you want to continue propagation from current state?

Figure 16: If you start to propagate a number of rows and the time window offset is not in state 0 you will see this warning but the propagation will proceed as soon as you press 'Yes, Continue!'.

The propagated values will be written to the respective cells in the data matrix.



#	ad	T3.Indicator2	T3.Indicator1	T3.Condition	[T4]P(Condition=failure 1)	[T4]P(Condition=failure 2)	[T4]P(Load=Abnormal)	[T5]P(Con…
0		medium	wear1	wear1	0.00018	0	0.04091	0.00019
1		medium	good	wear1	0.00018	0	0.04087	0.00004
2		high	good	wear1	0.0001	0	0.04088	0.00004
3		medium	good	wear1	0.00001	0	0.0409	0
4		medium	good	wear1	0	0	0.04066	0.00001
5		low	wear1	wear1	0.00001	0	0.04111	0.00001
6		low	good	wear1	0.00001	0	0.04096	0
7		medium	wear1	wear1	0.00104	0	0.05571	0.00114
8	ıal	low	wear1	wear1	0.69547	0	0.99963	0.0116

Figure 17: The propagated probabilities will be written to the cells in the data frame. Cells will be overwritten when the rows are propagated again.

To Evaluate a prediction HUGIN's *Data Processing Tool* (page 408) and HUGIN's *Data Classifier Performance Tool* (page 422) can be used. First the rows containing the true future values must be aligned with the prediction in the data set. This can be obtained using HUGIN's *Data Processing Tool* (page 408) .

To align the rows containing the true future values with the predictions open the Data Processing Tool by right clicking on a column header in the data frame and select Data Processing (Fig. 18).

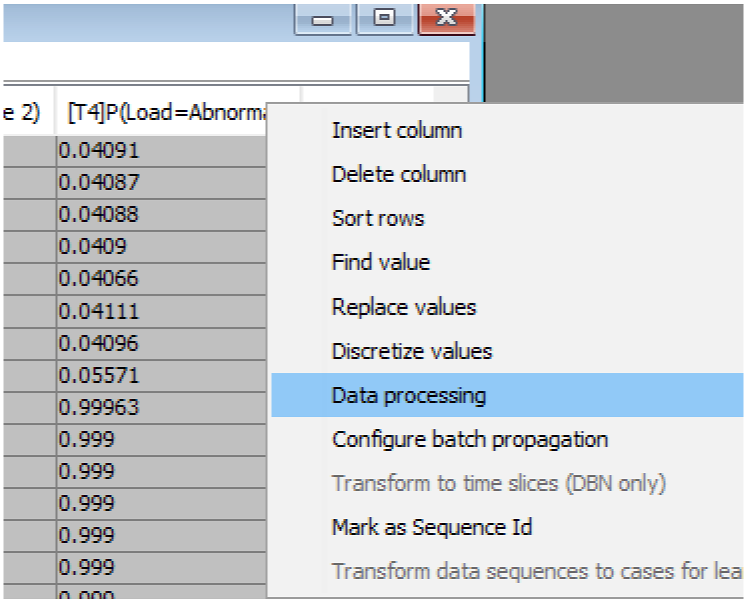


Figure 18: To open the Data Processing Tool right click on a column header and select Data Processing.

When The Data Processing Tool appears press '+' to create a new Data Processor. You can edit the Data Processor directly in the Details section in the dialog window below.

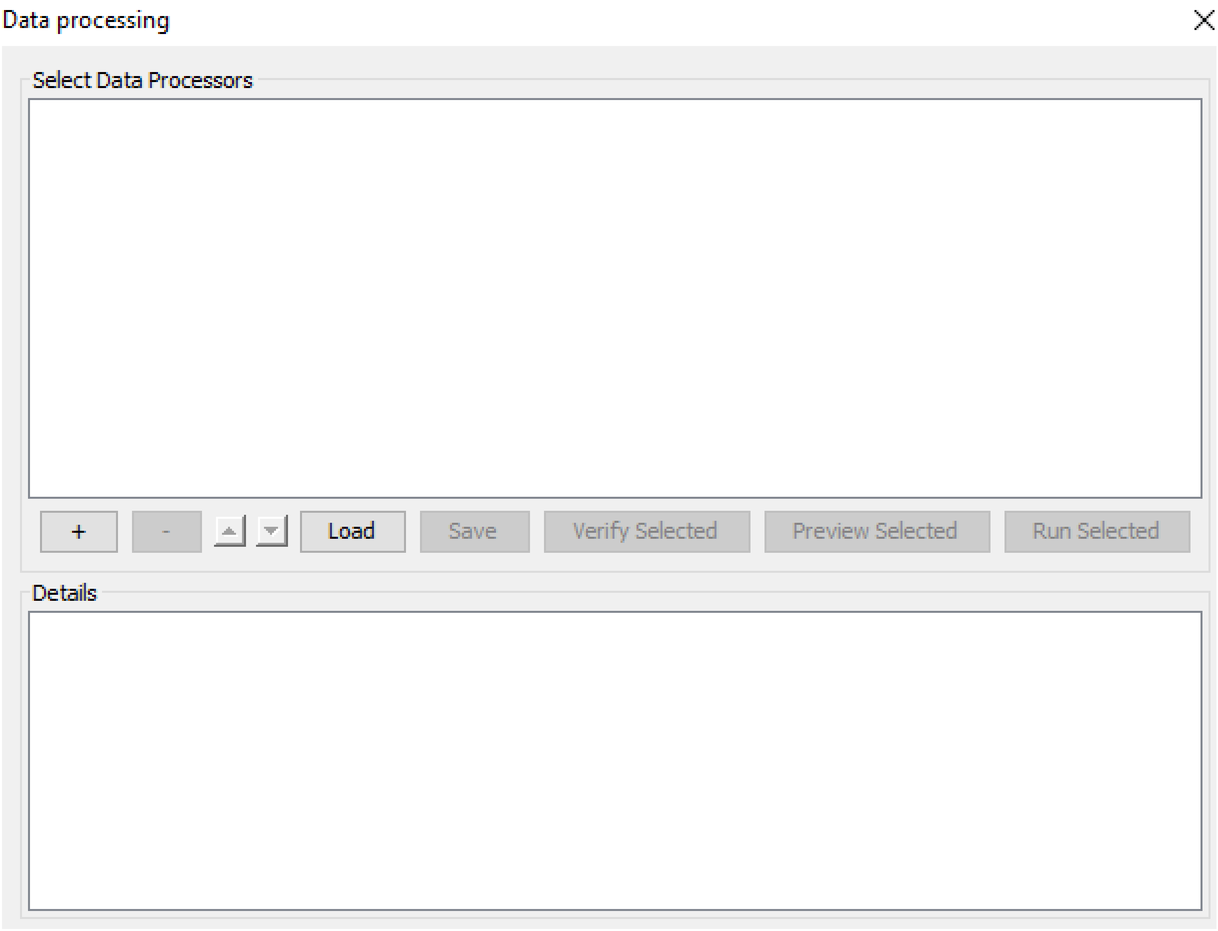


Figure 19: The Empty Data Processing Tool. Press '+' to create a new Data Processor. You can edit the Data Processor directly in the Details section in the dialog window below.

Assume you have a runtime model and a dataset with 3 time slices (i.e. T1, T2 and T3) and you have added a prediction on a variable in the future time slice T4. To align the true future data with the predictions in T4 you wish to copy the data in the T1 columns and shift them 4 cells up. The following 2 Data Processor Descriptions can obtain this when run in sequence:

To clone all columns in time slice T1 (T1.A, T1.B,. . .) and name them with prefix *"eval-T4-"* (eval-T4-T1.A, eval-T4-T1.B, . . .) use the following data processor description:

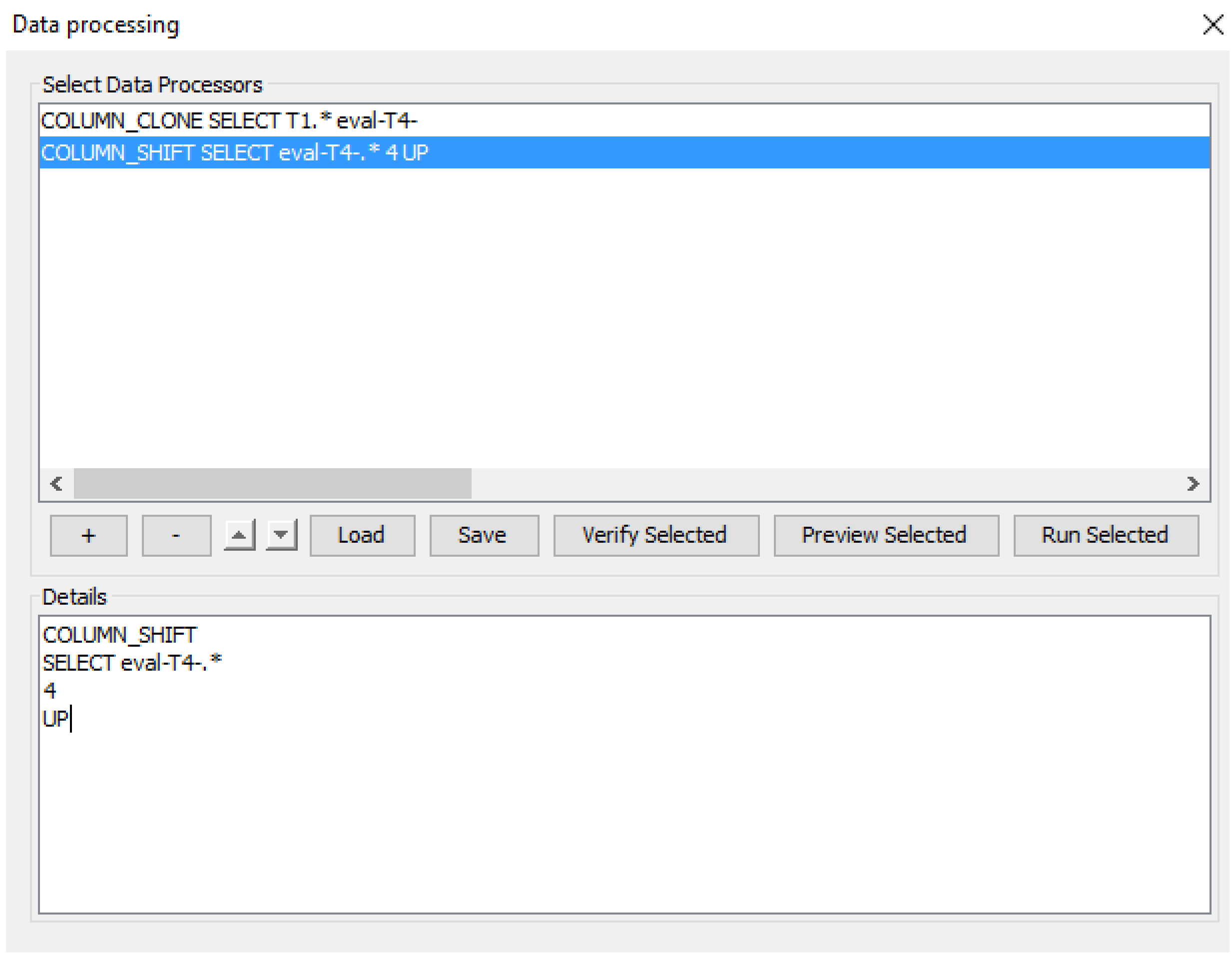
```
COLUMN_CLONE
SELECT T1.*
eval-T4-
```

To obtain this, press '+', select '<Empty Data Processor Template>' and enter the above text in the Details section. To shift all columns with prefix *"eval-T4-"* 4 cells up use the following data processor description:

```
COLUMN_SHIFT
SELECT eval-T4-.*
4
UP
```

To obtain this, press '+', select '<Empty Data Processor Template>' and enter the above text in the Details section.

When you have entered both data processing descriptions you should see something like Figure 20 where the 2 new data processors are listed.

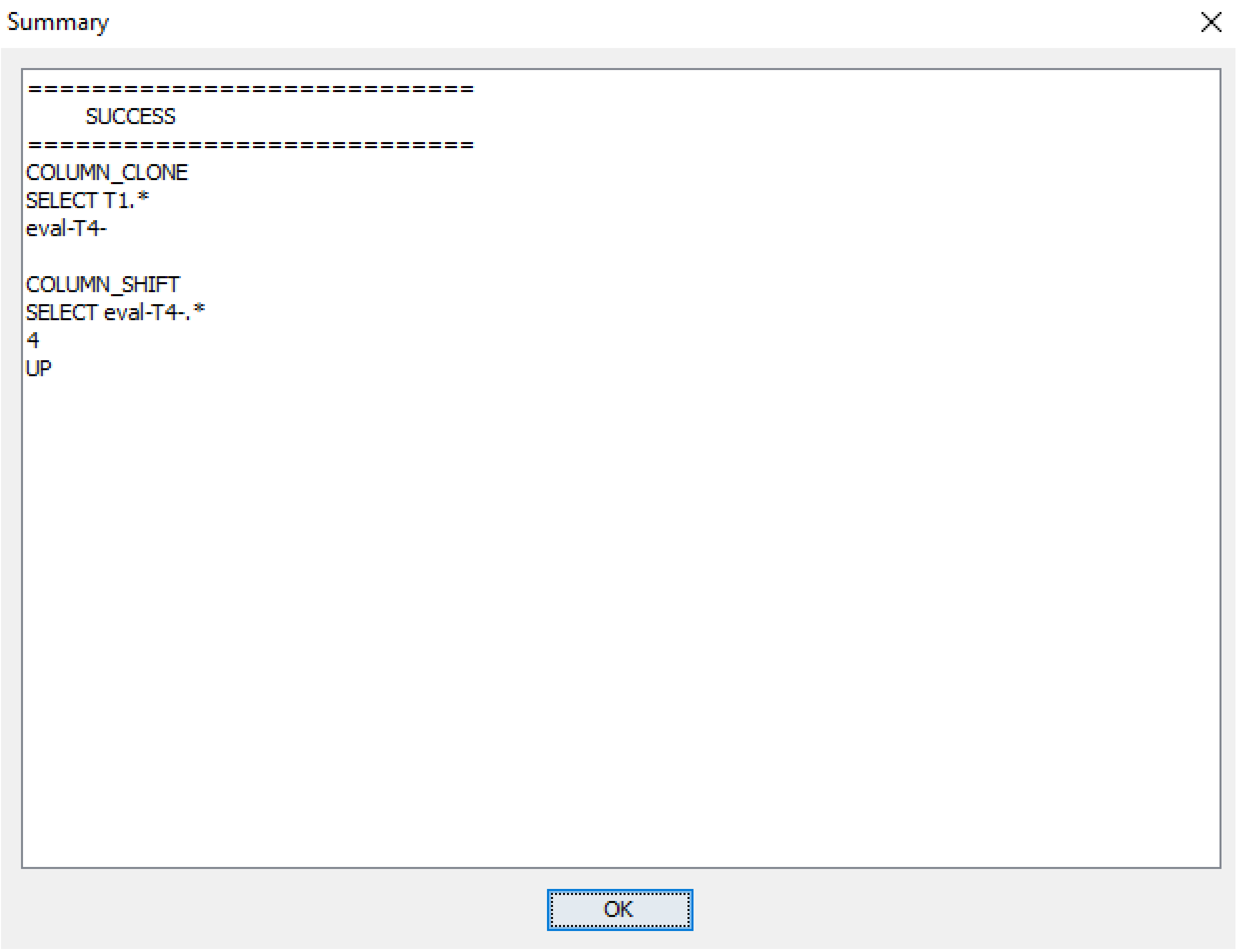


COLUMN_CLONE SELECT T1.* eval-T4-
COLUMN_SHIFT SELECT eval-T4-.* 4 UP

```
COLUMN_SHIFT
SELECT eval-T4-.*
4
UP
```

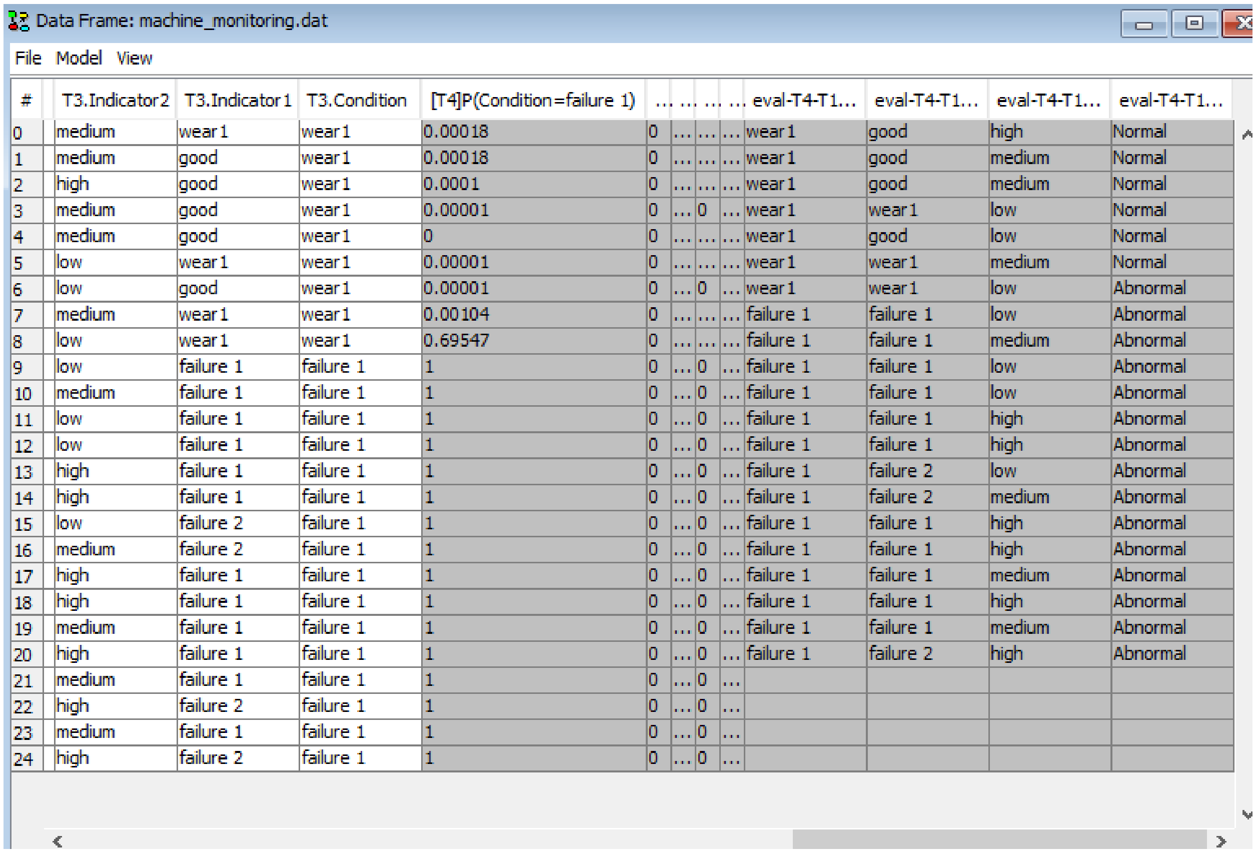
Figure 20: The Data Processing Tool with the 2 above processors entered.

To run the data processors select the two processors by holding down the CTRL key while selecting them in the list and press 'Run Selected'. If it ran successfully you should obtain the dialog in Figure 21. You can now close the Data Processing Tool and verify that the columns should have been added and shifted as requested (See Figure 22).



```
============================
     SUCCESS
============================
COLUMN_CLONE
SELECT T1.*
eval-T4-

COLUMN_SHIFT
SELECT eval-T4-.*
4
UP
```

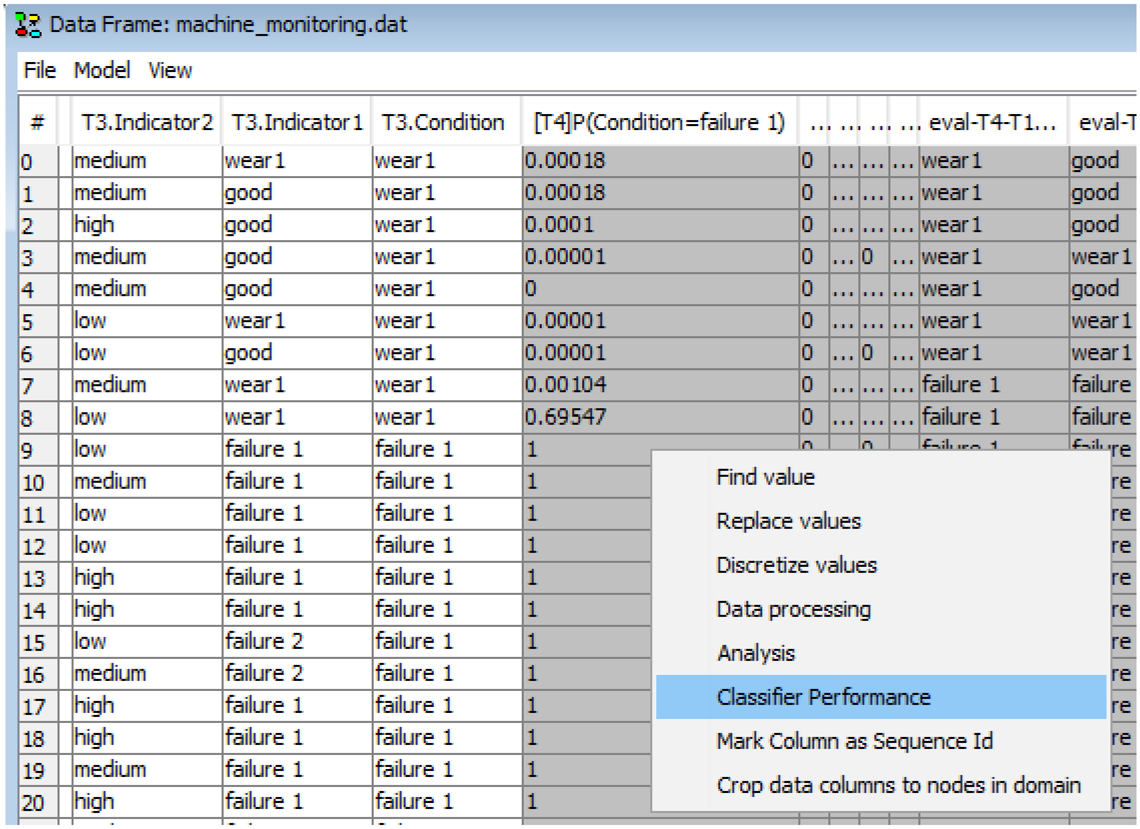
Figure 21: You should see this information .



#	T3.Indicator2	T3.Indicator1	T3.Condition	[T4]P(Condition=failure 1)	…	…	…	…	eval-T4-T1…	eval-T4-T1…	eval-T4-T1…	eval-T4-T1…
0	medium	wear1	wear1	0.00018	0	…	…	…	wear1	good	high	Normal
1	medium	good	wear1	0.00018	0	…	…	…	wear1	good	medium	Normal
2	high	good	wear1	0.0001	0	…	…	…	wear1	good	medium	Normal
3	medium	good	wear1	0.00001	0	…	0	…	wear1	wear1	low	Normal
4	medium	good	wear1	0	0	…	…	…	wear1	good	low	Normal
5	low	wear1	wear1	0.00001	0	…	…	…	wear1	wear1	medium	Normal
6	low	good	wear1	0.00001	0	…	0	…	wear1	wear1	low	Abnormal
7	medium	wear1	wear1	0.00104	0	…	…	…	failure 1	failure 1	low	Abnormal
8	low	wear1	wear1	0.69547	0	…	…	…	failure 1	failure 1	medium	Abnormal
9	low	failure 1	failure 1	1	0	…	0	…	failure 1	failure 1	low	Abnormal
10	medium	failure 1	failure 1	1	0	…	0	…	failure 1	failure 1	low	Abnormal
11	low	failure 1	failure 1	1	0	…	0	…	failure 1	failure 1	high	Abnormal
12	low	failure 1	failure 1	1	0	…	0	…	failure 1	failure 1	high	Abnormal
13	high	failure 1	failure 1	1	0	…	0	…	failure 1	failure 2	low	Abnormal
14	high	failure 1	failure 1	1	0	…	0	…	failure 1	failure 2	medium	Abnormal
15	low	failure 2	failure 1	1	0	…	0	…	failure 1	failure 1	high	Abnormal
16	medium	failure 2	failure 1	1	0	…	0	…	failure 1	failure 1	high	Abnormal
17	high	failure 1	failure 1	1	0	…	0	…	failure 1	failure 1	medium	Abnormal
18	high	failure 1	failure 1	1	0	…	0	…	failure 1	failure 1	high	Abnormal
19	medium	failure 1	failure 1	1	0	…	0	…	failure 1	failure 1	medium	Abnormal
20	high	failure 1	failure 1	1	0	…	0	…	failure 1	failure 2	high	Abnormal
21	medium	failure 1	failure 1	1	0	…	0	…				
22	high	failure 2	failure 1	1	0	…	0	…				
23	medium	failure 1	failure 1	1	0	…	0	…				
24	high	failure 2	failure 1	1	0	…	0	…				

Figure 22: The columns have been cloned and shifted such that the predicted values are aligned with the actual future values.

The predictions can be evaluated using the HUGIN's *Data Classifier Performance Tool* (page 422). To open this tool right click one of the cells in the Data Frame and select ‘Classifier Performance’ as shown in Figure 23. This shoul open the Classifier Performance Tool shown in Figure 24.



#	T3.Indicator2	T3.Indicator1	T3.Condition	[T4]P(Condition=failure 1)	…	…	…	…	eval-T4-T1…	eval-T
0	medium	wear1	wear1	0.00018	0	…	…	…	wear1	good
1	medium	good	wear1	0.00018	0	…	…	…	wear1	good
2	high	good	wear1	0.0001	0	…	…	…	wear1	good
3	medium	good	wear1	0.00001	0	…	0	…	wear1	wear1
4	medium	good	wear1	0	0	…	…	…	wear1	good
5	low	wear1	wear1	0.00001	0	…	…	…	wear1	wear1
6	low	good	wear1	0.00001	0	…	0	…	wear1	wear1
7	medium	wear1	wear1	0.00104	0	…	…	…	failure 1	failure
8	low	wear1	wear1	0.69547	0	…	…	…	failure 1	failure
9	low	failure 1	failure 1	1	0		0		failure 1	failure
10	medium	failure 1	failure 1	1						re
11	low	failure 1	failure 1	1						re
12	low	failure 1	failure 1	1						re
13	high	failure 1	failure 1	1						re
14	high	failure 1	failure 1	1						re
15	low	failure 2	failure 1	1						re
16	medium	failure 2	failure 1	1						re
17	high	failure 1	failure 1	1						re
18	high	failure 1	failure 1	1						re
19	medium	failure 1	failure 1	1						re
20	high	failure 1	failure 1	1						re

Figure 23: Right click on a data cell and select ‘Classifier Performance’ to evaluate the prediction.

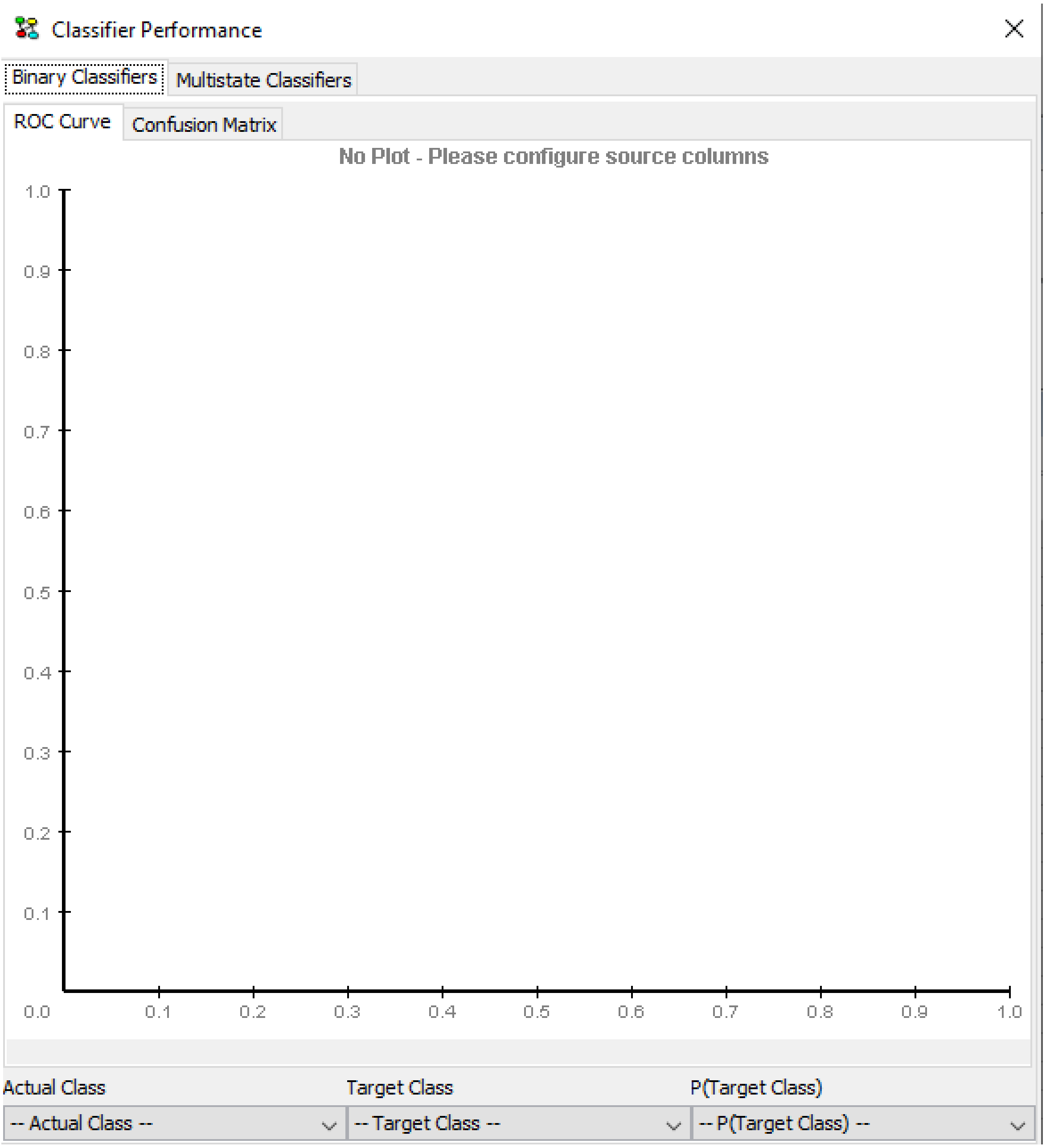


Figure 24: The Classifier Performance Tool. The tool is configured by selecting the three evaluation parameters in the bottom of the window.

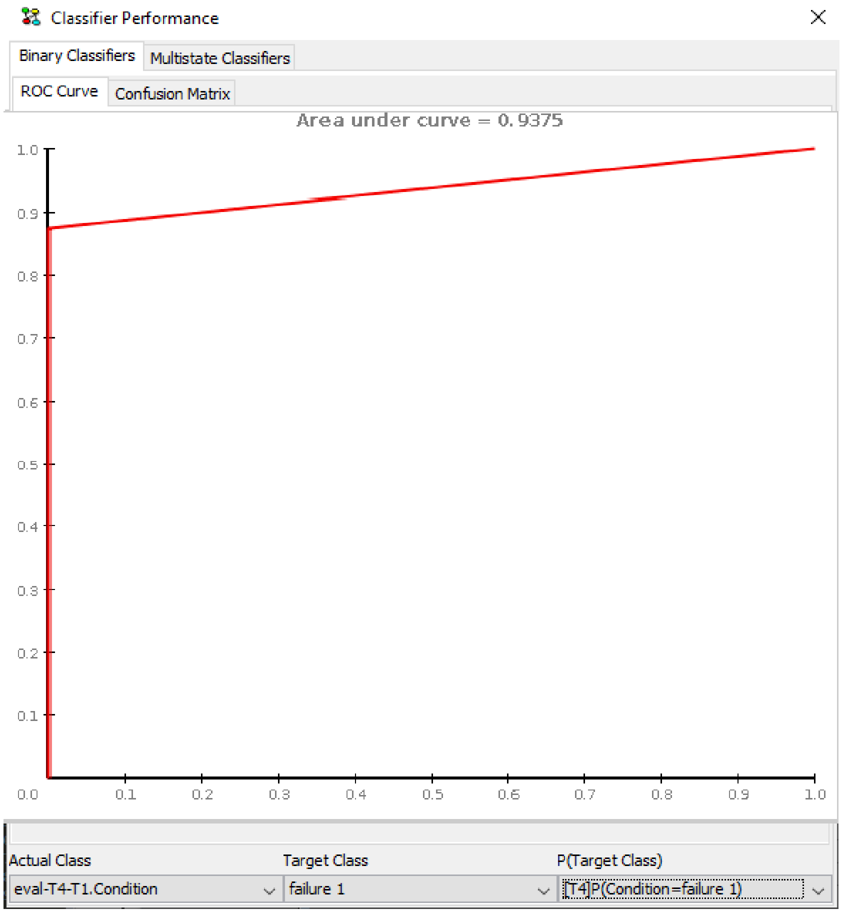
Configure the classification by selecting values for the three parameters at the bottom of the Classifier Performance window:

- **Actual Class**: Select the column containing the actual future value. Assume we wish to evaluate the performance of the prediction labelled "[T4]P(Condition=failure1)" and that we have used the data processors described above. Then the column labelled "eval-T4-T1.Condition" contains the actual values that "[T4]P(Condition=failure1)" predicts.
- **Target Class**: Select the state that the prediction we are evaluating predicts. If we are evaluating "[T4]P(Condition=failure1)" then select "failure1".

- **P(Target Class)**: Select the column containing the prediction. In this case select “[T4]P(Condition=failure1)”.

The result of this configuratin is shown in Figure 25. The ROC curve appears as soon as the last value is selected.

If you need to see more details about the prediction, navigate to the “Confusion Matrix” tab to see a count of false positives and false negatives and other relevant parameters (See Figure 26).

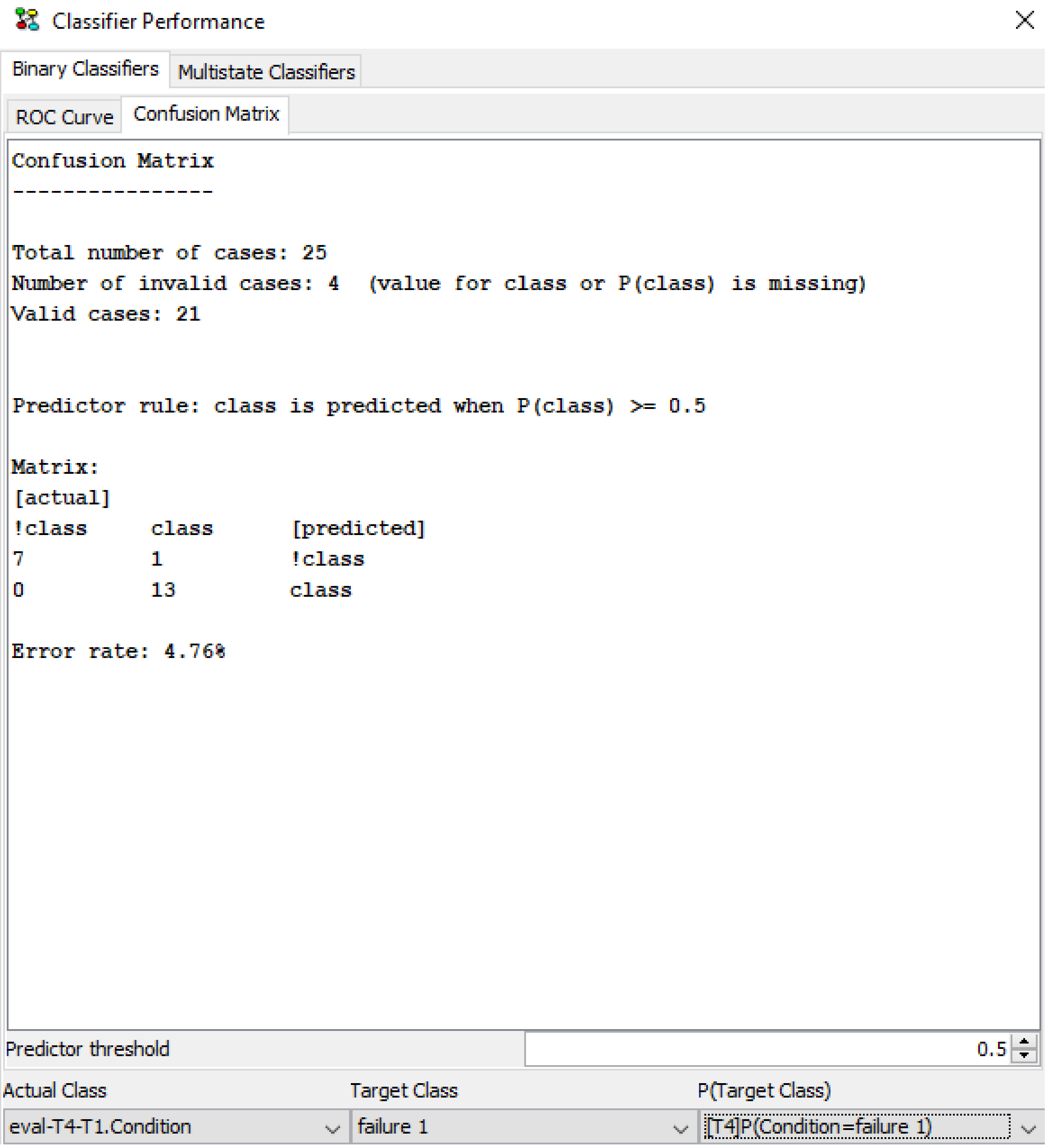


Actual Class: eval-T4-T1.Condition

Target Class: failure 1

P(Target Class): [T4]P(Condition=failure 1)

Figure 25: The resulting ROC Curve indicating the performance of the prediction.



```
Confusion Matrix
----------------

Total number of cases: 25
Number of invalid cases: 4  (value for class or P(class) is missing)
Valid cases: 21


Predictor rule: class is predicted when P(class) >= 0.5

Matrix:
[actual]
!class     class      [predicted]
7          1          !class
0          13         class

Error rate: 4.76%
```

Predictor threshold: 0.5

Actual Class: eval-T4-T1.Condition

Target Class: failure 1

P(Target Class): [T4]P(Condition=failure 1)

Figure 26: The confusion matrix for the prediction.

4.8.5 Leave One Out Propagation

Leave-One-Out Propagation is propagation provided within the the *Data Matrix* (page 431) tool that allows to perform N-fold Cross Validation where N is the number of cases in the data set used for evaluation of a model.

Requirements

Before the functionality is available the data set must be associated with a compiled network with at least one node with experience tables.

Concepts

Leave-One-Out Propagation works in the following way:

- A data set **D** with cases r_1, r_2, . . . , r_n.
- A Bayesian Network *M* over the nodes **C**. Let **C'** $\subseteq$ **C** be a set of nodes which have experience tables.
- For each row *i*, HUGIN uses the EM-algorithm to learn a model *M_i* in which the CPTs for C' are learned from the cases r_1, r_2,. . . ,r_{i-1}, r_{i+1},. . . .,r_n
- Enter the evidence of row r_i into *M_i* and propagate to obtain the desired beliefs.

The desired beliefs are specified in the 'Configure Batch Propagation' dialog window.

The rest of this manual page will be a walk through of how to obtain this functionality.

Loading the Bayesian network and Data file

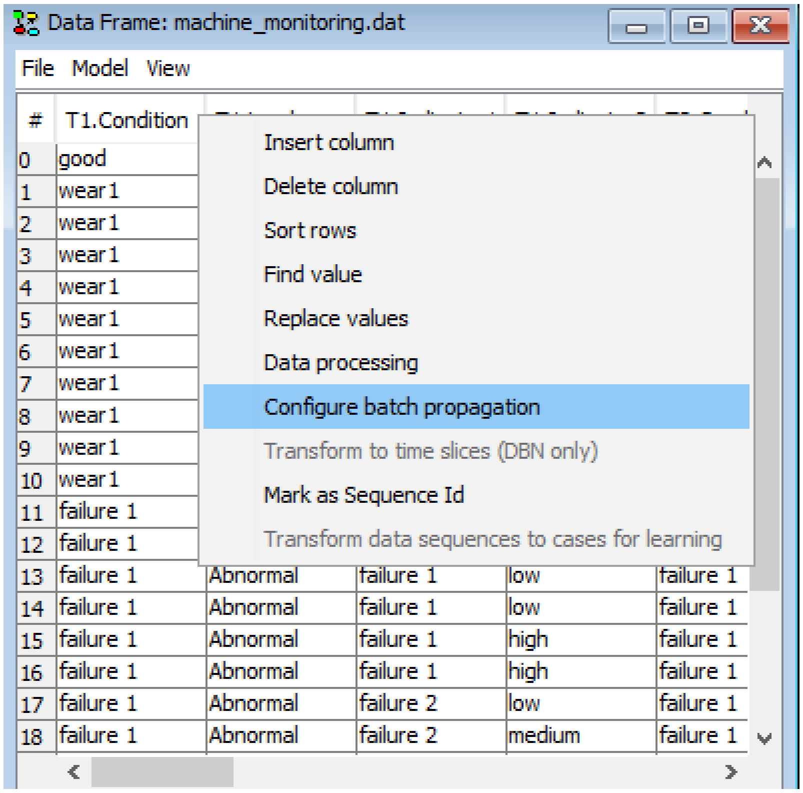
Press Ctrl + O, navigate to the file containing your model and press 'Open'. If you have not already added Experience tables to the nodes you wish to learn, see *Adding experience tables* (page 385). Compile the model to enter *run-mode* (page 123).

The next step is to open a data file in the Data Frame. By following these links you can read how to:

- *open your data file* (page 361)
- *associate you data file with a run-mode model, and* (page 362)
- *convert the data to time windows (if using a DBN Model)* (page 363).

Configuring Leave-One-Out propagation

After having performed the above steps, right-click on a column header in the data frame window and select 'Configure batch propagation' (See Figure 1).



Data Frame: machine_monitoring.dat
File
Model
View
Insert column
Delete column
Sort rows
Find value
Replace values
Data processing
Configure batch propagation
Transform to time slices (DBN only)
Mark as Sequence Id
Transform data sequences to cases for learning

#	T1.Condition				
0	good				
1	wear 1				
2	wear 1				
3	wear 1				
4	wear 1				
5	wear 1				
6	wear 1				
7	wear 1				
8	wear 1				
9	wear 1				
10	wear 1				
11	failure 1				
12	failure 1				
13	failure 1	Abnormal	failure 1	low	failure 1
14	failure 1	Abnormal	failure 1	low	failure 1
15	failure 1	Abnormal	failure 1	high	failure 1
16	failure 1	Abnormal	failure 1	high	failure 1
17	failure 1	Abnormal	failure 2	low	failure 1
18	failure 1	Abnormal	failure 2	medium	failure 1

Figure 1: Right-click on a column header in the data frame window and select ‘Configure batch propagation’ to enter the Batch Propagation Dialog.

Mark nodes for classification

Navigate to the ‘Classify/Learn’ tab, click ‘Classify Node’ to configure a node as target for learning and evaluation. The effect of doing this is that the column representing the selected variable will obtain the prefix “[C]” which indicates to the propagation algorithm that data from this coulumn will be included while learning *M_i* but will *not* be included while performing the propagation of the data in row *i*.

If the class variable is not marked, the class variable will be included in the propagation of *M_i* with row *i* which results in a wrong evaluation of the classifier.

Figure 2 and Figure 3 shows the dialog for marking a node for classification. Any number of nodes can be marked.

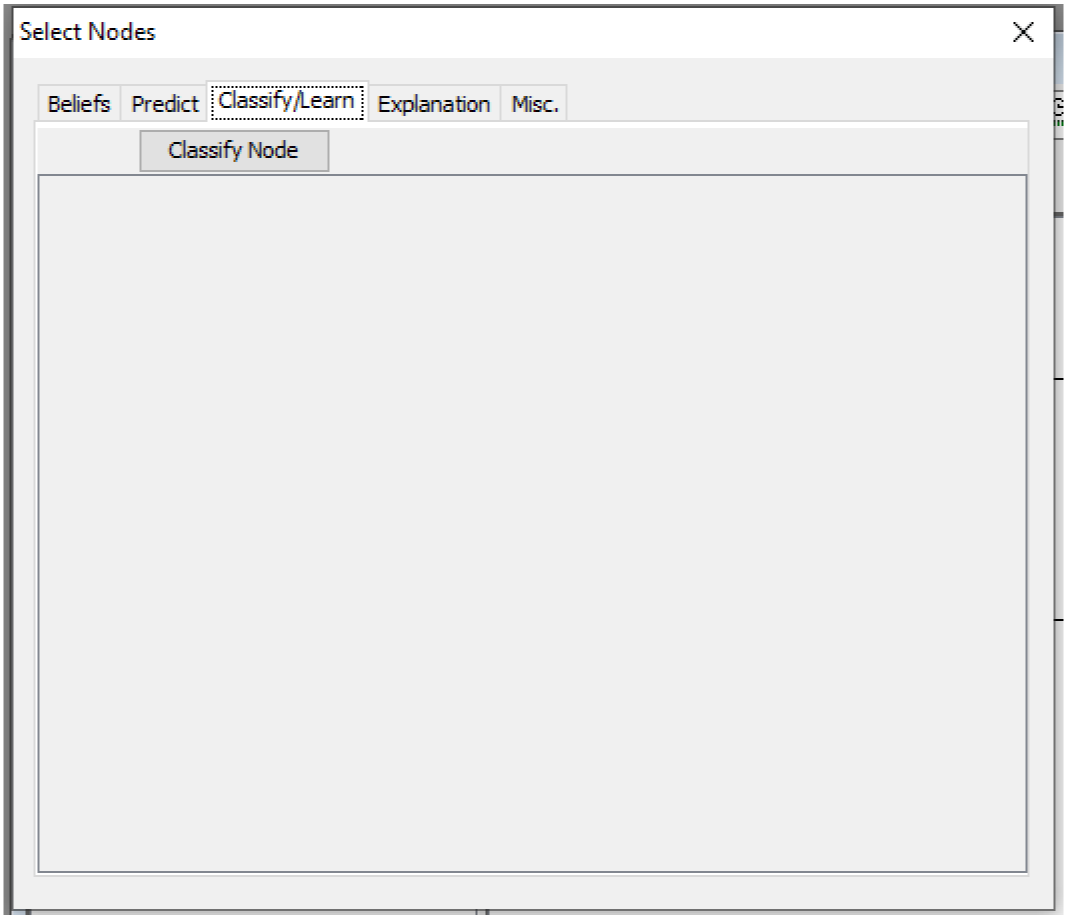
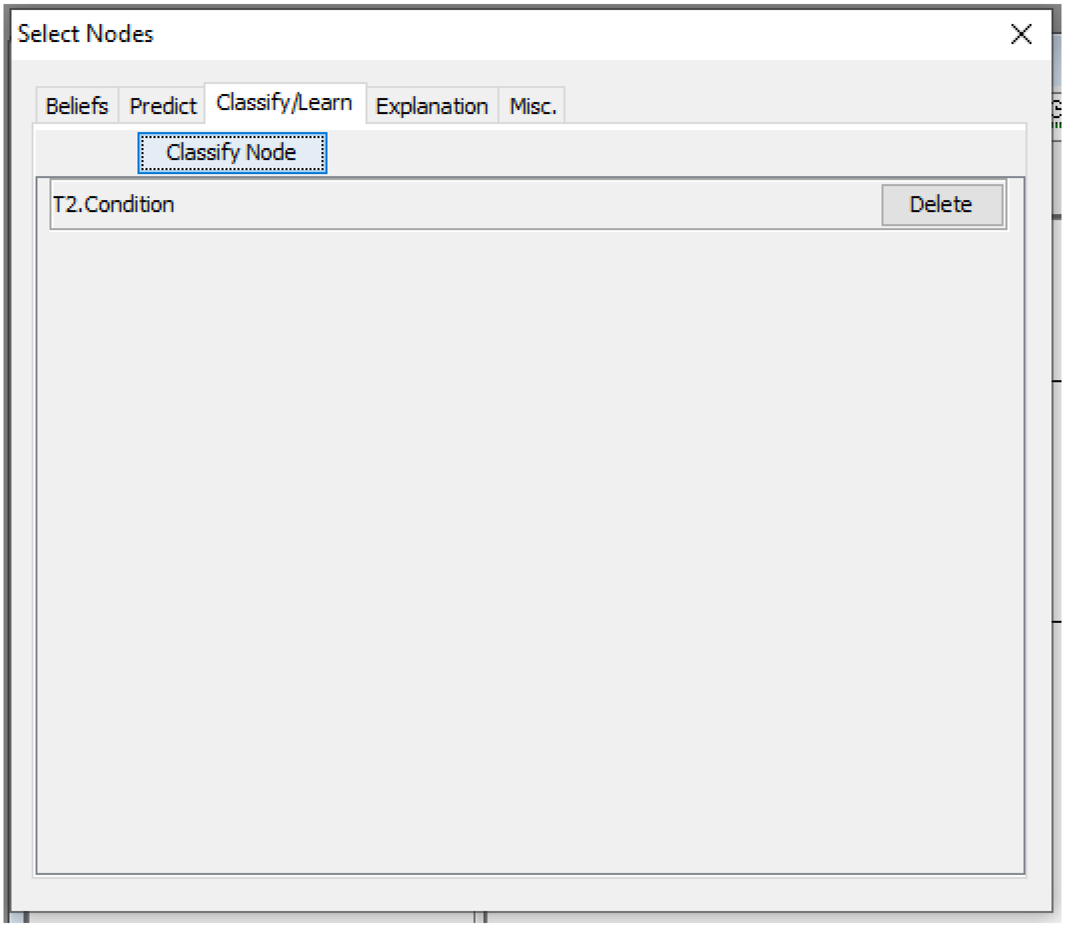


Figure 2: Navigate to the ‘Classify/Learn’ tab.



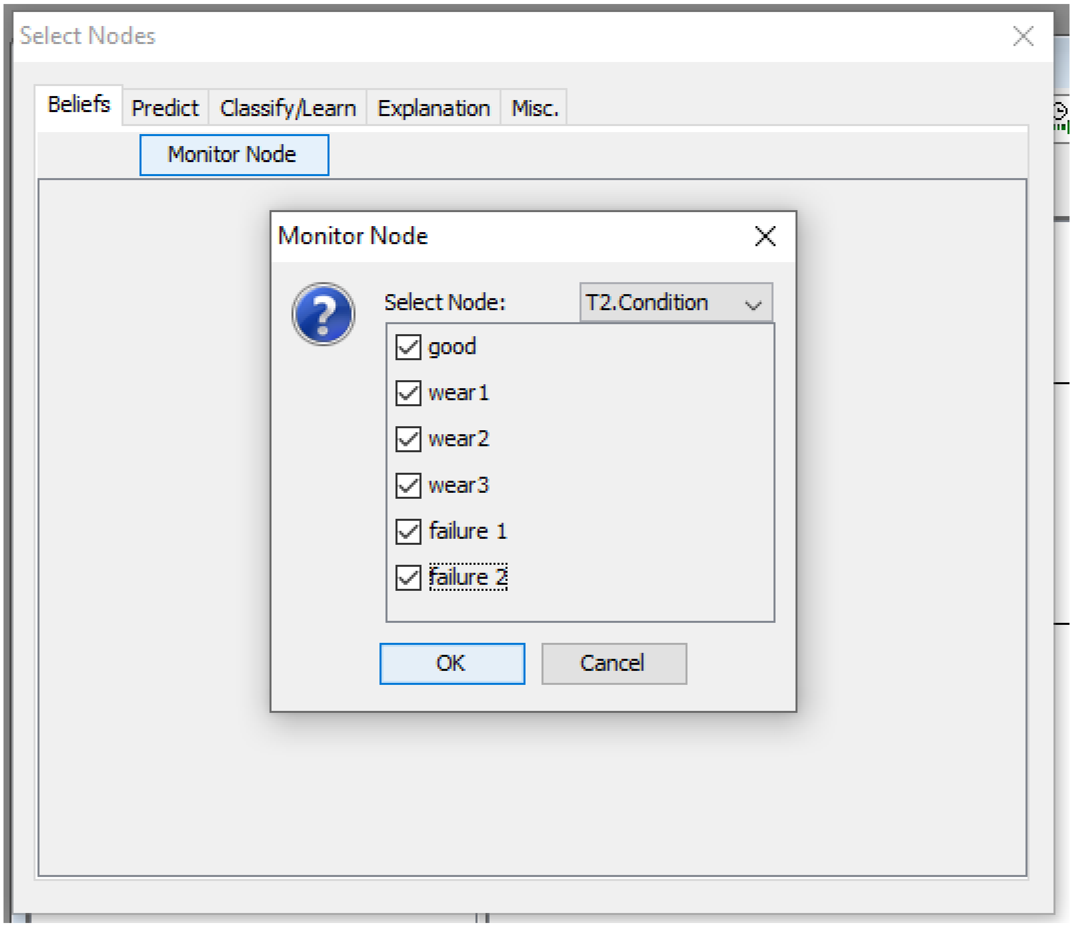
T2.Condition

Figure 3: Click ‘Classify Node’ to mark a node as target for learning and evaluation.

Configure the class variable for monitoring

Navigate to the ‘Beliefs’ tab, click ‘Monitor Node’ to configure a node for monitoring of its beliefs during the propagation.

The effect of doing this is that a number of columns (one column for each selected state) will be added to the data set which will contain the beliefs for the variable being in that state.

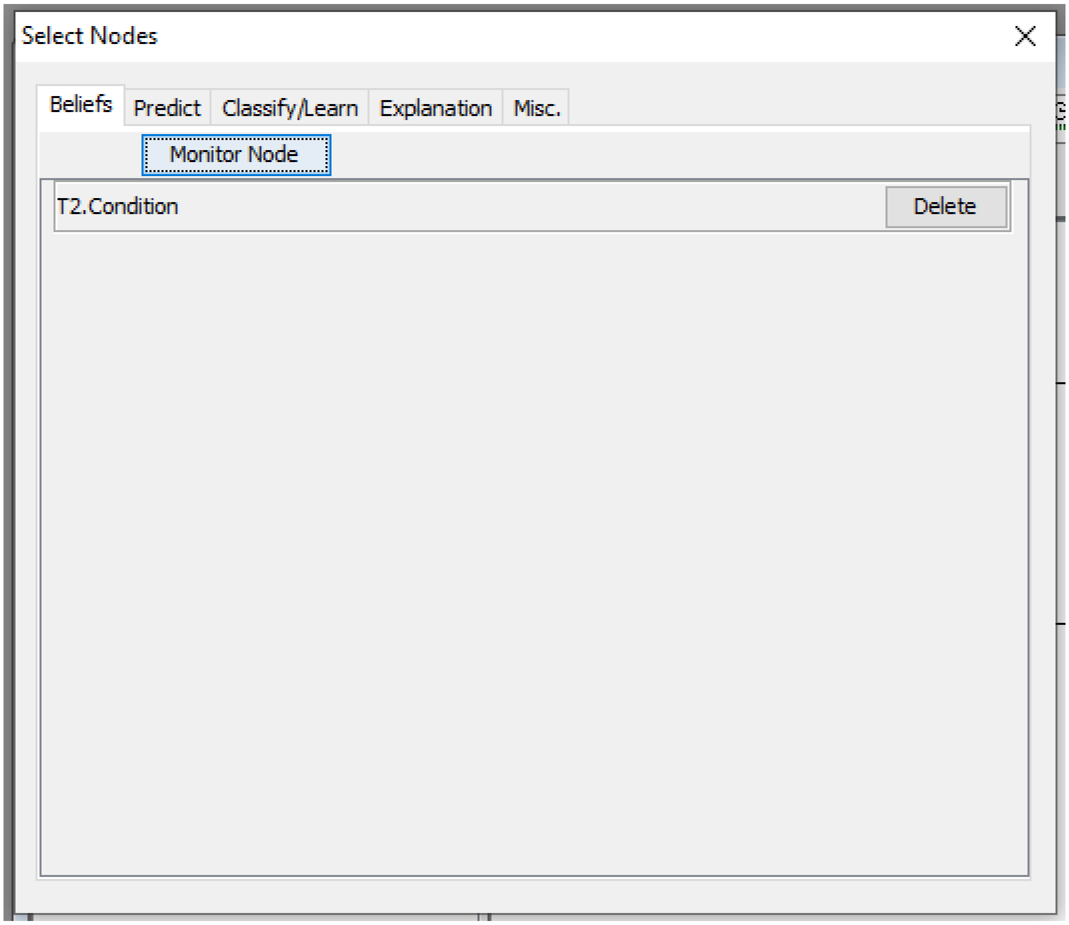


Select Node: T2.Condition

☑ good
☑ wear1
☑ wear2
☑ wear3
☑ failure 1
☑ failure 2

Figure 4: Navigate to the 'Beliefs' tab, click 'Monitor Node' and select the state for monitoring. Select the node twice in the 'Select Node' dropdown bar in order to mark all states for monitoring.

The nodes selected for monitoring will be listed in the pane. Add as many nodes for monitoring as you wish.



T2.Condition

Figure 5: The nodes marked for monitoring are listed. Add as many as you like.

Adding Experience Tables

Experience tables can be added to nodes when the network is in edit mode.

Right-click on the node, select ‘Experience/Fading table operations’ and select ‘Add Experience Table’. (See Figure 6.)

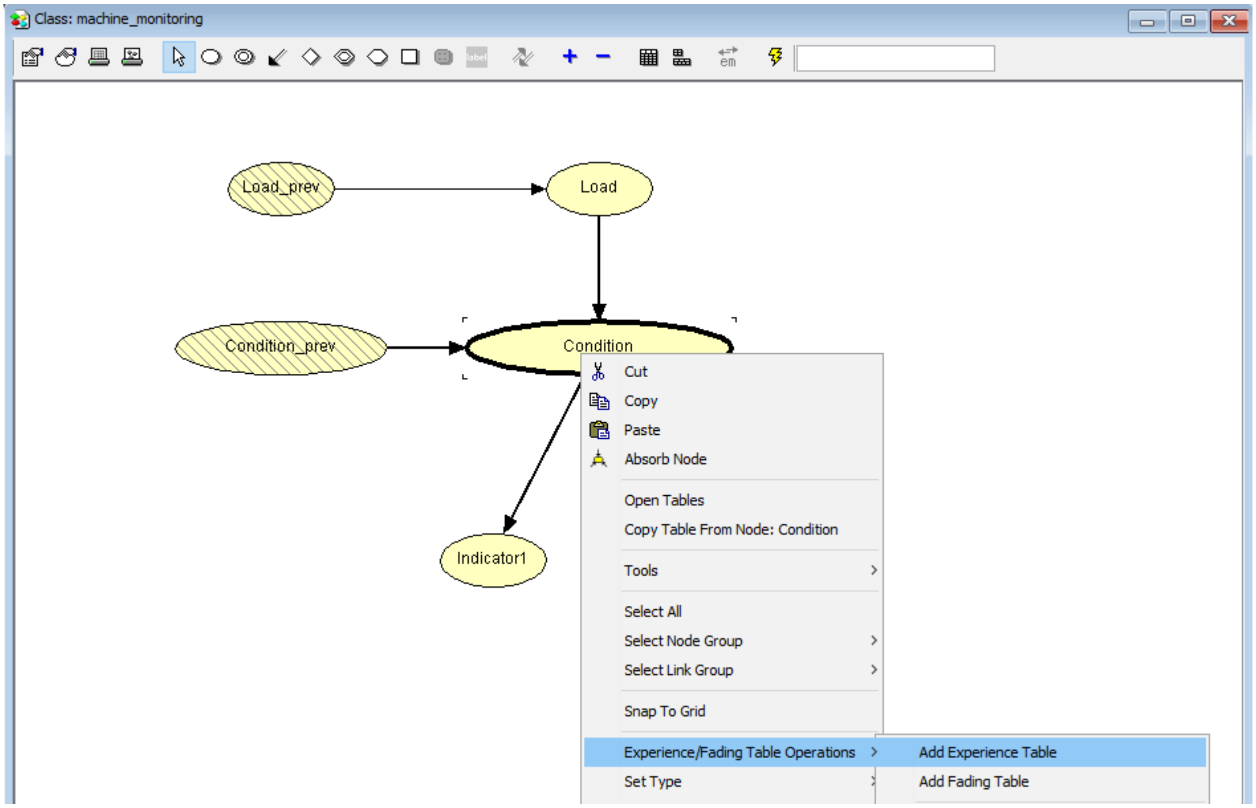
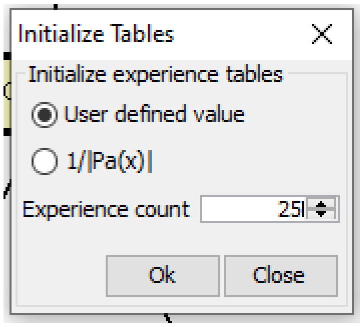


Figure 6: Adding an experience Table.

As initial parameters for the experience table we have here selected 'User defined value' and experience count 25. That means that the initial distribution on the nodes will have a weight as if it was based on 25 observations.


Initialize Tables

Initialize experience tables

● User defined value

○ 1/|Pa(x)|

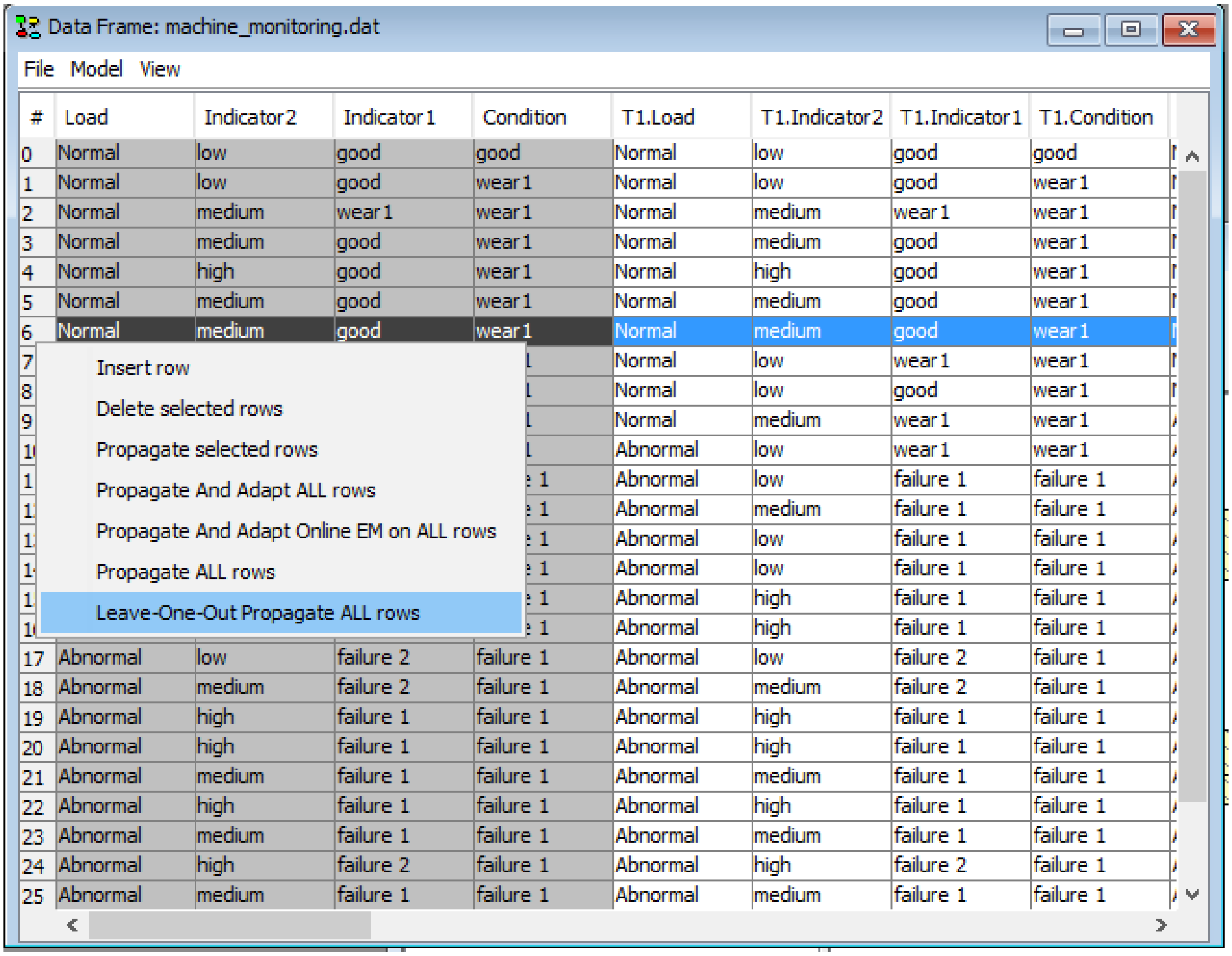
Experience count 25

Figure 7: Set initial parameters for experience table.

Add experience tables to all nodes for which you wish to learn the CPT. In order to perform Leave-One-Out cross validation propagation at least one node must have experience table.

Perform Leave-One-Out-Propagation

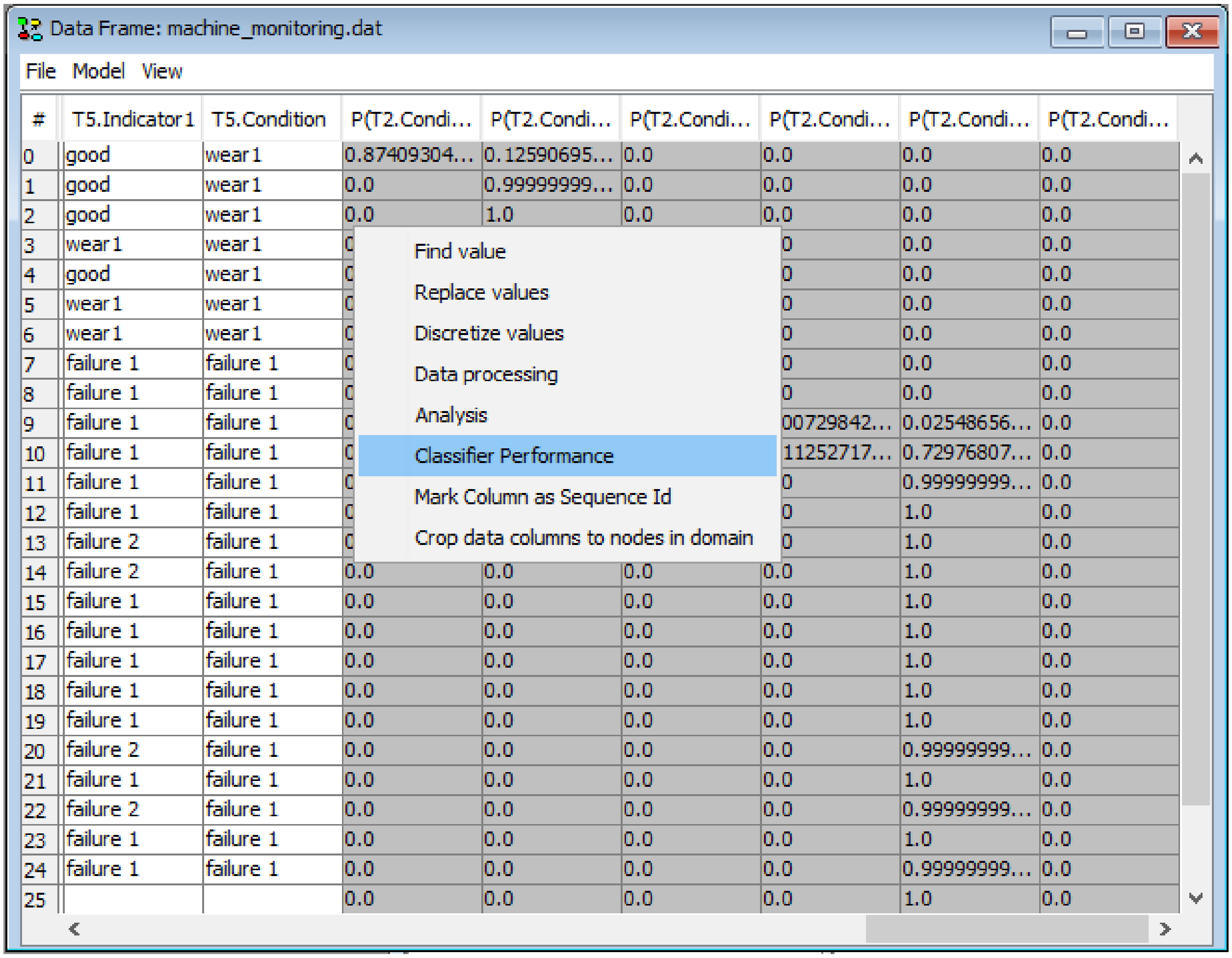
Close the ‘Select Nodes’ window, right-click on a row-number and select ‘Leave-One-Out Propagate ALL rows’ in order to start the propagation.


Data Frame: machine_monitoring.dat
File
Model
View
Insert row
Delete selected rows
Propagate selected rows
Propagate And Adapt ALL rows
Propagate And Adapt Online EM on ALL rows
Propagate ALL rows
Leave-One-Out Propagate ALL rows

#	Load	Indicator2	Indicator1	Condition	T1.Load	T1.Indicator2	T1.Indicator1	T1.Condition
0	Normal	low	good	good	Normal	low	good	good
1	Normal	low	good	wear1	Normal	low	good	wear1
2	Normal	medium	wear1	wear1	Normal	medium	wear1	wear1
3	Normal	medium	good	wear1	Normal	medium	good	wear1
4	Normal	high	good	wear1	Normal	high	good	wear1
5	Normal	medium	good	wear1	Normal	medium	good	wear1
6	Normal	medium	good	wear1	Normal	medium	good	wear1
7				l	Normal	low	wear1	wear1
8				l	Normal	low	good	wear1
9				l	Normal	medium	wear1	wear1
1				l	Abnormal	low	wear1	wear1
1				e 1	Abnormal	low	failure 1	failure 1
1				e 1	Abnormal	medium	failure 1	failure 1
1				e 1	Abnormal	low	failure 1	failure 1
1				e 1	Abnormal	low	failure 1	failure 1
1				e 1	Abnormal	high	failure 1	failure 1
1				e 1	Abnormal	high	failure 1	failure 1
17	Abnormal	low	failure 2	failure 1	Abnormal	low	failure 2	failure 1
18	Abnormal	medium	failure 2	failure 1	Abnormal	medium	failure 2	failure 1
19	Abnormal	high	failure 1	failure 1	Abnormal	high	failure 1	failure 1
20	Abnormal	high	failure 1	failure 1	Abnormal	high	failure 1	failure 1
21	Abnormal	medium	failure 1	failure 1	Abnormal	medium	failure 1	failure 1
22	Abnormal	high	failure 1	failure 1	Abnormal	high	failure 1	failure 1
23	Abnormal	medium	failure 1	failure 1	Abnormal	medium	failure 1	failure 1
24	Abnormal	high	failure 2	failure 1	Abnormal	high	failure 2	failure 1
25	Abnormal	medium	failure 1	failure 1	Abnormal	medium	failure 1	failure 1

Figure 8: Start the propagation.

Inspect and evaluate the propagated result The propagated beliefs should now be placed in the data matrix. The data can be saved to a file or the *Classifier Performance tool* (page 361) can be used to evaluate the classifier performance.



#	T5.Indicator1	T5.Condition	P(T2.Condi...	P(T2.Condi...	P(T2.Condi...	P(T2.Condi...	P(T2.Condi...	P(T2.Condi...
0	good	wear1	0.87409304...	0.12590695...	0.0	0.0	0.0	0.0
1	good	wear1	0.0	0.99999999...	0.0	0.0	0.0	0.0
2	good	wear1	0.0	1.0	0.0	0.0	0.0	0.0
3	wear1	wear1	0			0	0.0	0.0
4	good	wear1	0			0	0.0	0.0
5	wear1	wear1	0			0	0.0	0.0
6	wear1	wear1	0			0	0.0	0.0
7	failure 1	failure 1	0			0	0.0	0.0
8	failure 1	failure 1	0			0	0.0	0.0
9	failure 1	failure 1	0			00729842...	0.02548656...	0.0
10	failure 1	failure 1	0			11252717...	0.72976807...	0.0
11	failure 1	failure 1	0			0	0.99999999...	0.0
12	failure 1	failure 1	0			0	1.0	0.0
13	failure 2	failure 1	0			0	1.0	0.0
14	failure 2	failure 1	0.0	0.0	0.0	0.0	1.0	0.0
15	failure 1	failure 1	0.0	0.0	0.0	0.0	1.0	0.0
16	failure 1	failure 1	0.0	0.0	0.0	0.0	1.0	0.0
17	failure 1	failure 1	0.0	0.0	0.0	0.0	1.0	0.0
18	failure 1	failure 1	0.0	0.0	0.0	0.0	1.0	0.0
19	failure 1	failure 1	0.0	0.0	0.0	0.0	1.0	0.0
20	failure 2	failure 1	0.0	0.0	0.0	0.0	0.99999999...	0.0
21	failure 1	failure 1	0.0	0.0	0.0	0.0	1.0	0.0
22	failure 2	failure 1	0.0	0.0	0.0	0.0	0.99999999...	0.0
23	failure 1	failure 1	0.0	0.0	0.0	0.0	1.0	0.0
24	failure 1	failure 1	0.0	0.0	0.0	0.0	0.99999999...	0.0
25			0.0	0.0	0.0	0.0	1.0	0.0

Figure 9: Inspect the propagated beliefs in the data matrix, and use the Classifier Performance tool to evaluate the classifier performance.

4.8.6 Case Files

The HUGIN Graphical User Interface supports saving and loading of case files. A case file contains a description of a single evidence scenario. Figure 1 shows a sample case file.

```
B: "stable"     % Burning regimen
F: "defect"     % Filter state
D: 5.6  % Dust emission
C: -1.7 % CO2 concentration
L: 0.3  % Light penetrability
```

Figure 1: A sample case file.

As indicated in Figure 1, each line of a case file contains information about the evidence for a single variable. In general, each line of a case file must obey the following syntax:

```
<variable name>: <value> [% <comment>]
```

where

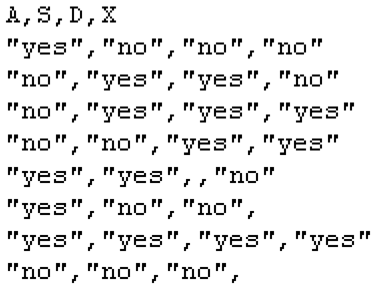
```
<value> : \:= "<state label>" | true | false | <number> | #<state index>
```

The choice of syntax for <value> depends on the *category* (page 209) and *type* (page 216) of the variable/node. For labeled nodes <value> must be specified using the "<state label>" syntax. For Boolean nodes <value> must be either 'true' or 'false'. For numbered nodes or *continuous chance nodes* (page 224) <value> must be a number (e.g., 1, -4.2). For interval nodes <value> must be specified as #i, where i is the index of the state in question (the first state has index 0).

A case file can be generated via the "Save Case…" item of the *File Menu* (page 191) whenever the HUGIN Graphical User Interface is in *Run Mode* (page 123) and evidence has been entered. Similarly, in *Run Mode* (page 123), a case can be loaded via the "Load Case…" item. Please note that possible evidence already entered is removed when a case is loaded.

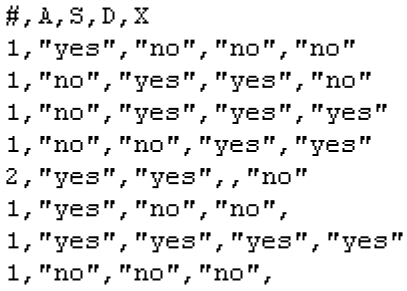
4.8.7 Data Files

The HUGIN Graphical User Interface supports saving and loading of data files. A data file contains a set of cases as this



```
A,S,D,X
"yes","no","no","no"
"no","yes","yes","no"
"no","yes","yes","yes"
"no","no","yes","yes"
"yes","yes",,"no"
"yes","no","no",
"yes","yes","yes","yes"
"no","no","no",
```

Figure 1: A sample data file.



```
#,A,S,D,X
1,"yes","no","no","no"
1,"no","yes","yes","no"
1,"no","yes","yes","yes"
1,"no","no","yes","yes"
2,"yes","yes",,"no"
1,"yes","no","no",
1,"yes","yes","yes","yes"
1,"no","no","no",
```

Figure 2: Another sample data file.

As indicated in Figure 1, the first line is the name of the nodes as in the network whereas each of the following lines specify a case in which each variable is assigned a value, i.e., a set of observations with one observation for each node. An observation may missing in a case. A missing value is specified using N/A.

Notice that the first line of the data file always contains the names of a (subset) of the nodes in the network. It is important to specify the name and not the label of nodes. The name is a unique identifier for each node. Thus, the data file in Figure 1 assumes that the network has nodes with names E, T, L, S, A , D, B and X. This could be the node names of the network shown in Figure 3 (where node labels are shown in the nodes.)

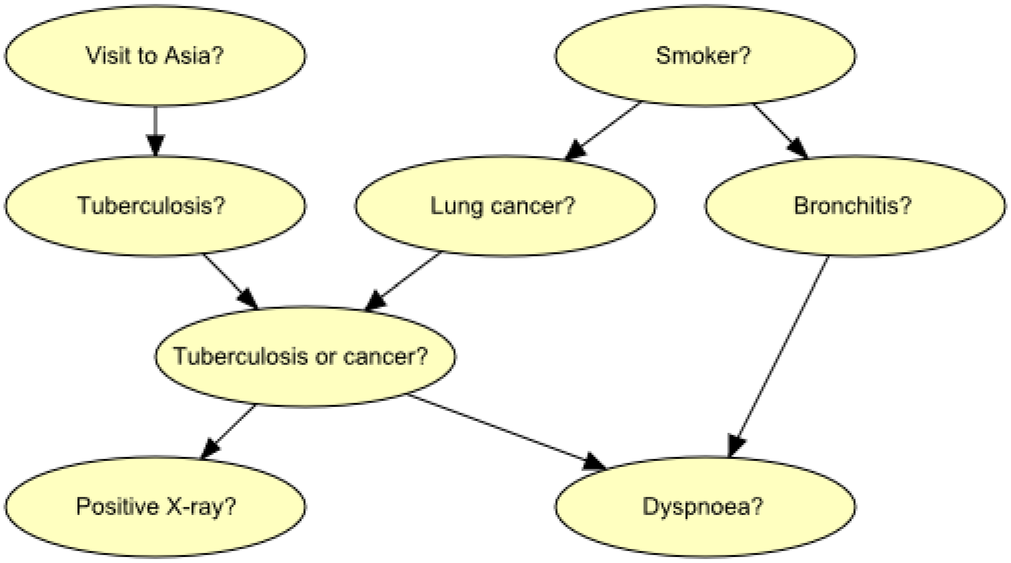


Figure 3: Bayesian-network representation of “Chest Clinic”.

See also notes on data files for *OOBN EM* (page 348).

The format of a data file can be described by the following grammar:

```
<Data file> ::= <Header> <Case>*
<Header> ::= # <Separator> <Node list> | <Node list>
<Separator> ::= , | <Empty>
<Node list> ::= <Node name> | <Node list> <Separator> <Node name>
<Case> ::= <Case count> <Separator> <Data list> | <Data list>
<Case count> ::= <Nonnegative real number>
<Data list> ::= <Data> | <Data list> <Separator> <Data>
<Data> ::= <Value> | N/A | <Empty>
<Value> ::= <State index> | <Label> | <Real number> | true | false
<State index> ::= #<Integer>
```

Where:

- **The <Header>** must occupy a single line in the file. Likewise, each <Case> must occupy a single line.
- **If #** is the first element of <Header>, then each <Case> must include a <Case count>.
- **Each <Case>** must contain a <Data> item for each node specified in the <Header>. The ith <Data> item (if it is a <Value>) in the <Data list> must be valid for the ith node in the <Node list> of the <Header>.
- **If <Data>** is *, ?, or <Empty>, then the data is taken as ‘missing’.
- **If <Separator>** is <Empty>, then none of the separated items is allowed to be <Empty>.
- **<Value>** is as defined in Section 8.8, with the exception that <Likelihood> is not allowed.

- **<Real number>** is a valid specification for CG, numbered, and interval nodes. For numbered and interval nodes, the acceptable values are defined by the state values of the named node.
- **<Label>** is a valid specification for labeled nodes. The label (a doublequoted string) must match a unique state label of the named node. Quotes can be omitted if the label is a single word.
- ***true* and *false*** are valid specifications for boolean nodes.

4.8.8 Case Generator

Based on the current conditional probability distribution the HUGIN Graphical User Interface can generate cases and save these into a file.

Cases can be generated in two different ways: MCAR (Missing Completely at Random) or MAR (Missing at Random).

MCAR sets some values to N/A by removing values of some nodes in the generated case set randomly (i.e., MCAR considers all the values in the generated cases and not one case at a time).

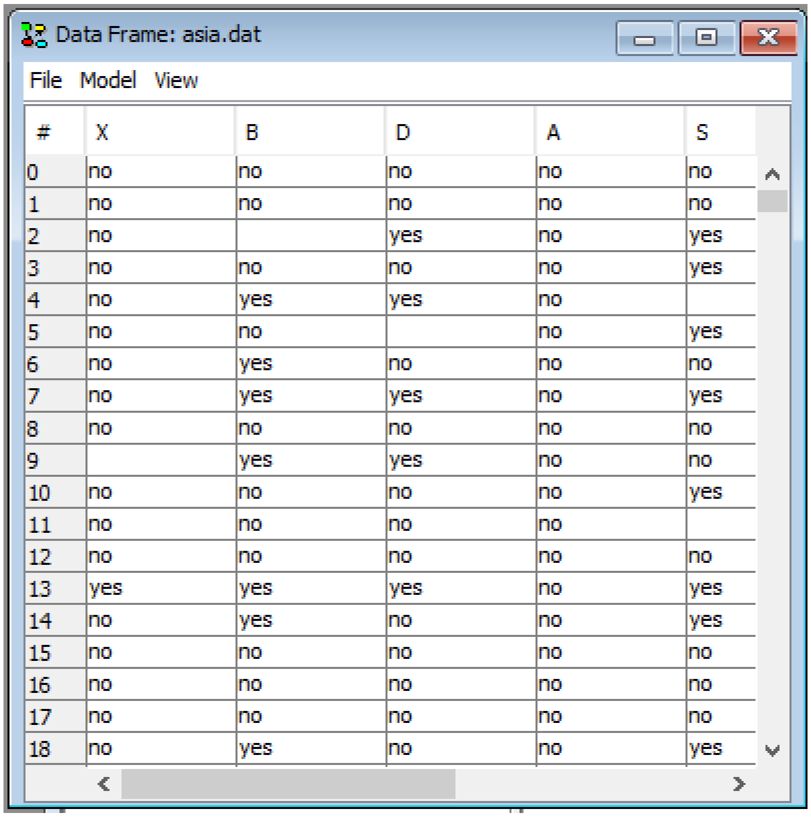
MAR randomly sets some values in a case to N/A based on auto-generated templates. These templates specify that if some nodes have a specific value, then the values of a subset of the nodes are randomly set to N/A. Note that the specification of the templates is generated randomly.

In both cases the conditional probability distribution is considered as a factor to the number of cases generated with a specific set of observations. Also in both MCAR and MAR it is possible to indicate the percentage of missing values. To activate the case generator, the *Case Generator Button* (page 157) may be used.

4.8.9 Data Frame

HUGIN can load a data file as a project frame, the data frame. Data files can be inspected, manipulated and analyzed as the contents of the data frame is a *data matrix* (page 431) with additional functionallity for performing batch propagation in *run mode* (page 123) models.

To open a new data frame and **load a data file**, click in the menu of the main HUGIN frame *File* -> *Open Data File* and select a data file.

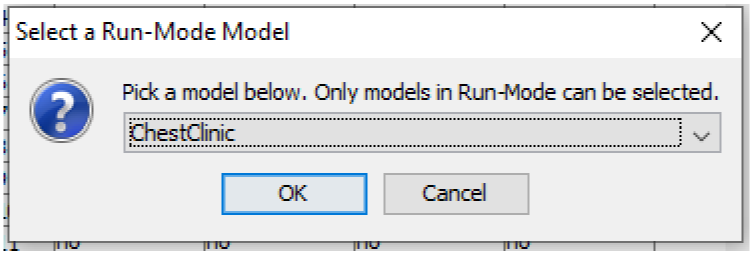


#	X	B	D	A	S
0	no	no	no	no	no
1	no	no	no	no	no
2	no		yes	no	yes
3	no	no	no	no	yes
4	no	yes	yes	no	
5	no	no		no	yes
6	no	yes	no	no	no
7	no	yes	yes	no	yes
8	no	no	no	no	no
9		yes	yes	no	no
10	no	no	no	no	yes
11	no	no	no	no	
12	no	no	no	no	no
13	yes	yes	yes	no	yes
14	no	yes	no	no	yes
15	no	no	no	no	no
16	no	no	no	no	no
17	no	no	no	no	no
18	no	yes	no	no	yes

Figure 1: Typical data frame - A data frame containing a data file for the chest clinic model with only symptom observations.

To configure a *data frame* for batch propagation, we must first load a network model and set the model to *run mode* (page 123) *(Note. batch propagation functionallity is only available for models in run-mode!)*.

Then in the data frame, click the *Model* menu and then click *Select a Run-Mode Model*. A dialog appears, containing all the current project frames to pick from.

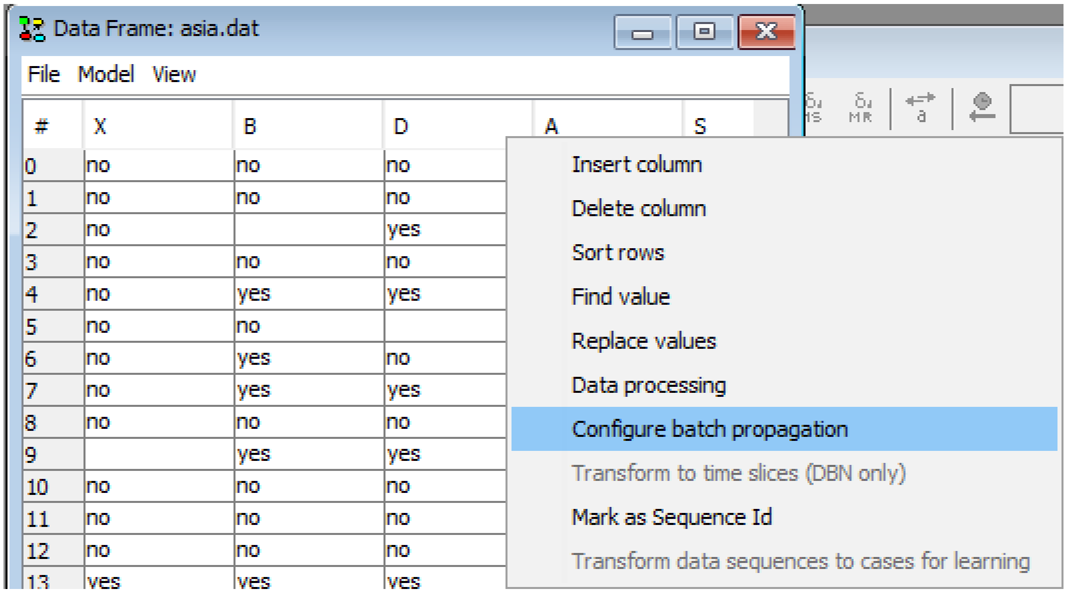


Pick a model below. Only models in Run-Mode can be selected.

ChestClinic

Figure 2: Pick a project for doing batch propagation.

When a *run mode* (page 123) model has been configured for the data frame, we can configure the batch propagation options. To do this, right click a column header in the data matrix and select the **Configure batch propagation** option.



#	X	B	D
0	no	no	no
1	no	no	no
2	no		yes
3	no	no	no
4	no	yes	yes
5	no	no	
6	no	yes	no
7	no	yes	yes
8	no	no	no
9		yes	yes
10	no	no	no
11	no	no	no
12	no	no	no
13	yes	yes	yes

Figure 3: Select configure batch propagation *(Note: if this option is dimmed out, then switch the associated network model to run-mode first)*.

This produces a dialog where we can configure what information to record during batch propagation. The information recorded depends on the node type:

- Discrete nodes - posterior beliefs for each state
- Continuous nodes - the mean or the variance.
- Function nodes - the function value
- Utility nodes - the expected utility

To add a node for recording, click the ‘Monitor Node’ button and the ‘Monitor Node’ dialog appears.

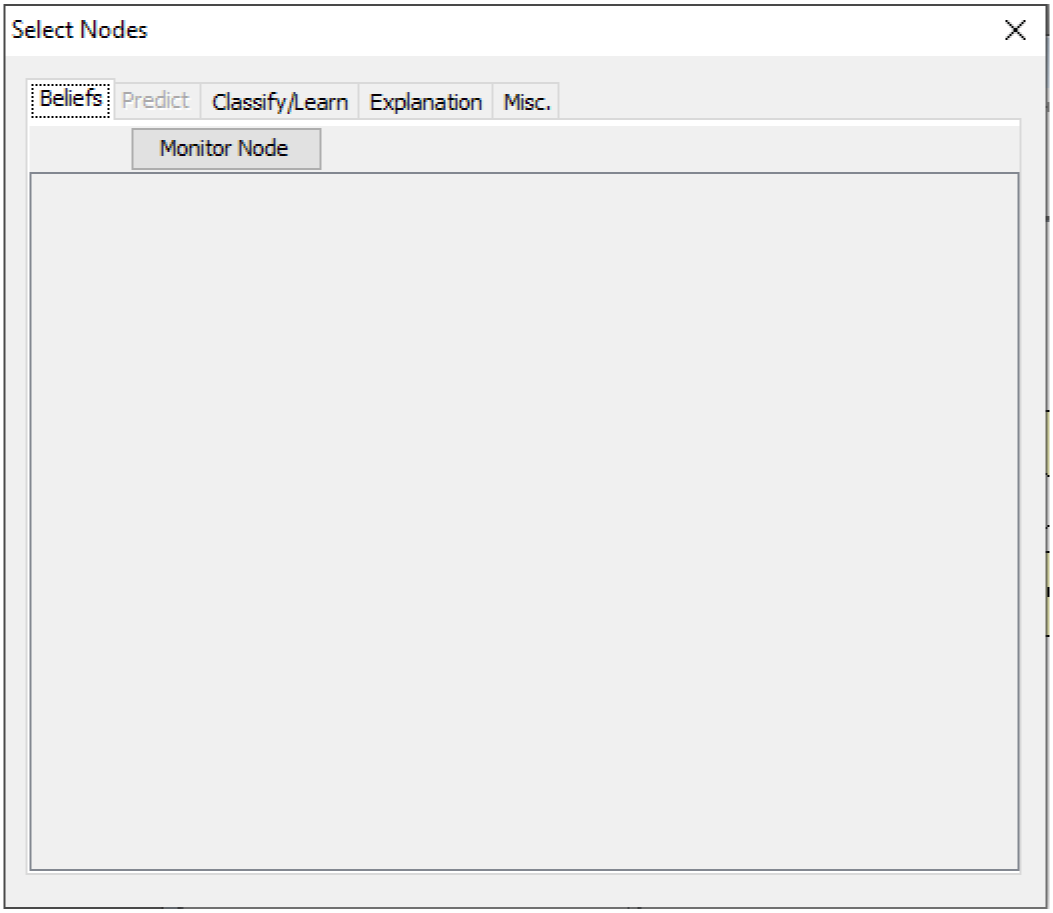
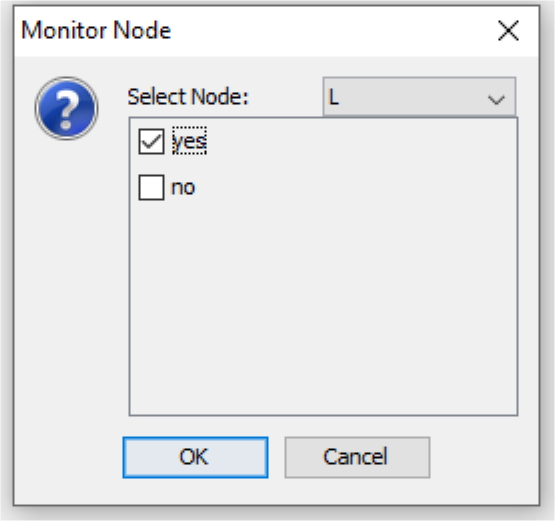


Figure 4: Pick any nodes we wish to record beliefs for and then close the dialog - in this example the nodes from the chest clinic model that represents the diagnosis.

In the 'Monitor Node' dialog select a node from the drop down menu. If the selected node is a Discrete Chance Node, select the states you wish to monitor. If the selected node is a Continuous Chance Node, select mean(μ) and/or variance (σ^2). Click the 'OK' button to add the node to the list of nodes selected for monitoring.



Monitor Node

Select Node: L

☑ yes

□ no

Figure 5: Pick any nodes we wish to record beliefs for and then close the dialog - in this example the nodes from the chest clinic model that represents the diagnosis.

The nodes selected for monitoring will be listed. Note that if the same node is selected twice it will only appear once in the list as it would otherwise result in duplicate columns in the data set. If a node is selected twice a state which was not previously selected is selected, it will be added to the existing row representing that node.

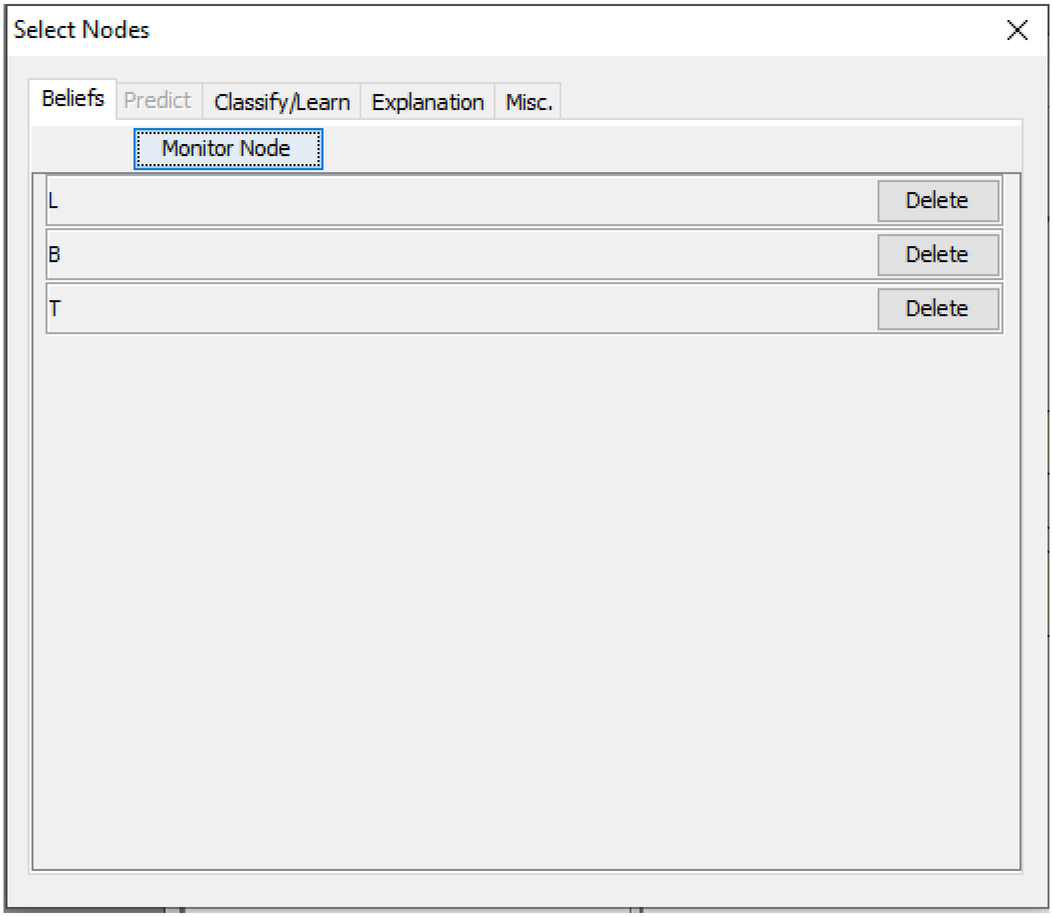
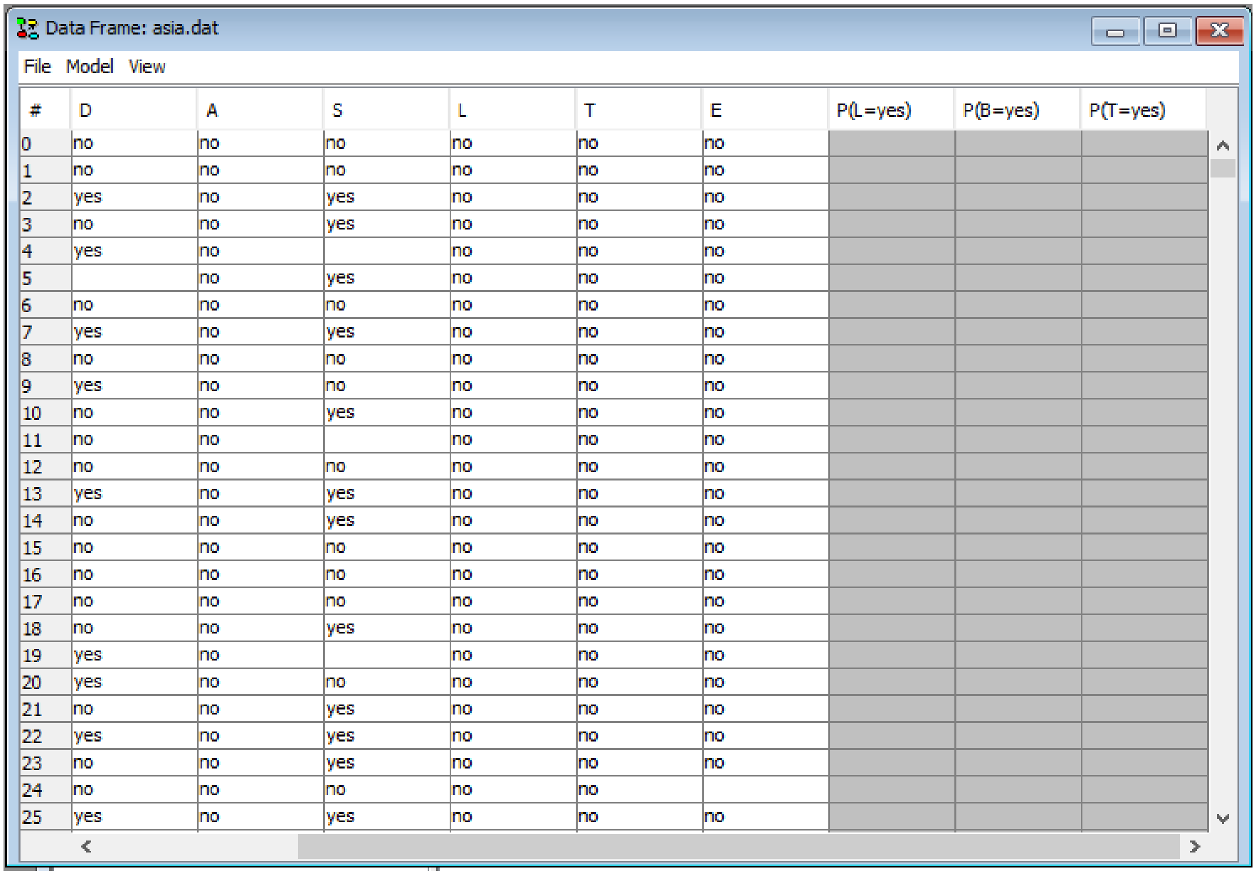


Figure 6: Pick any nodes we wish to record beliefs for and then close the dialog - in this example the nodes from the chest clinic model that represents the diagnosis.

Depending on the nodes selected, a number of extra columns appear to the data matrix. The information recorded in the column depends on the column name.

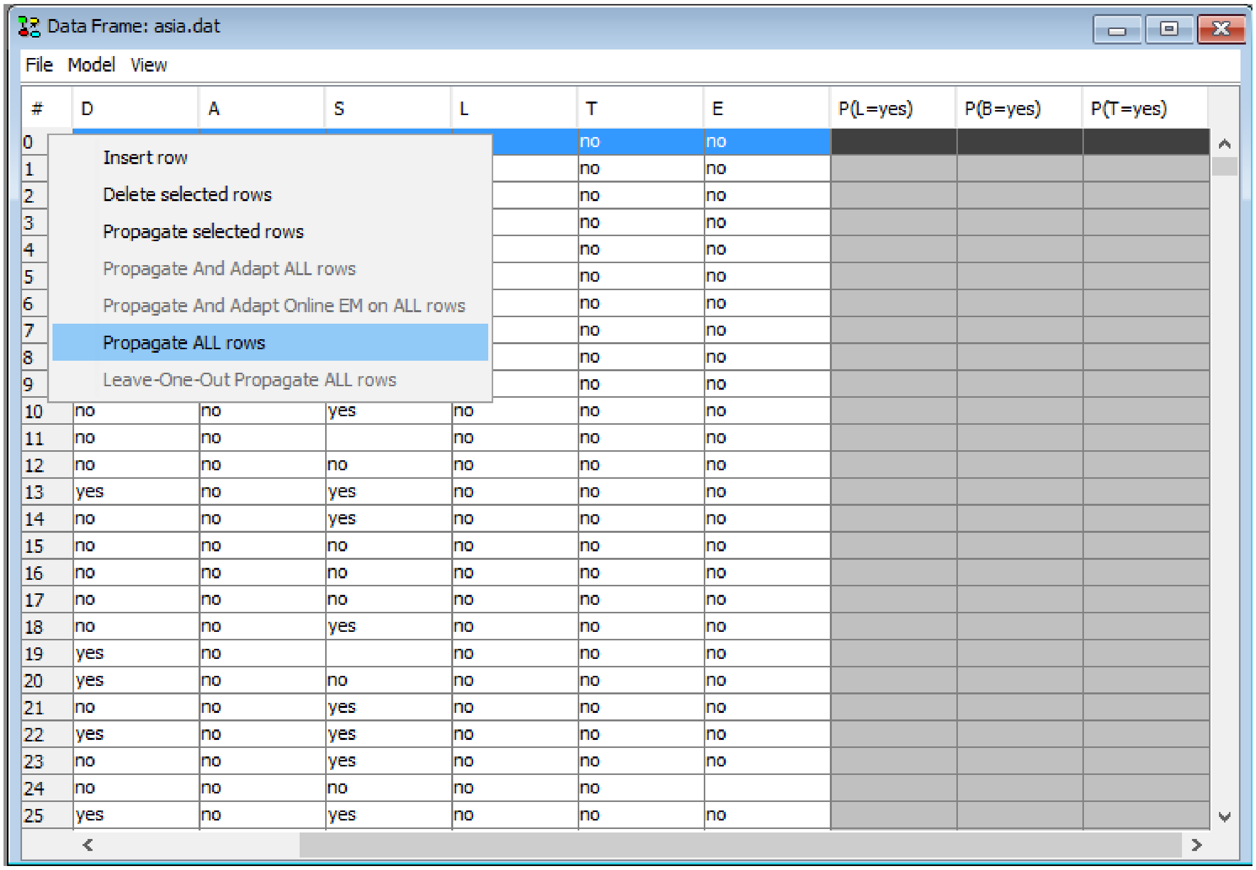


#	D	A	S	L	T	E	P(L=yes)	P(B=yes)	P(T=yes)
0	no	no	no	no	no	no			
1	no	no	no	no	no	no			
2	yes	no	yes	no	no	no			
3	no	no	yes	no	no	no			
4	yes	no		no	no	no			
5		no	yes	no	no	no			
6	no	no	no	no	no	no			
7	yes	no	yes	no	no	no			
8	no	no	no	no	no	no			
9	yes	no	no	no	no	no			
10	no	no	yes	no	no	no			
11	no	no		no	no	no			
12	no	no	no	no	no	no			
13	yes	no	yes	no	no	no			
14	no	no	yes	no	no	no			
15	no	no	no	no	no	no			
16	no	no	no	no	no	no			
17	no	no	no	no	no	no			
18	no	no	yes	no	no	no			
19	yes	no		no	no	no			
20	yes	no	no	no	no	no			
21	no	no	yes	no	no	no			
22	yes	no	yes	no	no	no			
23	no	no	yes	no	no	no			
24	no	no	no	no	no				
25	yes	no	yes	no	no	no			

Figure 7: Extra columns appear for recording information when a row (a case) is propagated.

In addition to using the above dialog, it is possible to add the columns by hand and specify the appropriate column header to obtain the desired results. There are options to manually specify that the propagate operation should report the probability of a state with maximum probability ([MAX](P(X))) for a discrete chance node, a state with maximum probability ([ARGMAX](P(X))), mean ([MEAN(X)), median ([MEDIAN](X)), quantile ([Qy](X) where 0 < y < 100) and coefficient of variance ([CoV](X)) of a numeric discrete node where the format of the column header is shown in parentheses for node X. For instance, to report the mean of X, the column header should be [MEAN](X), to report the coefficient of variance the header should be [CoV](X) and to report the 95 percent quantile the header should be [Q95](X).

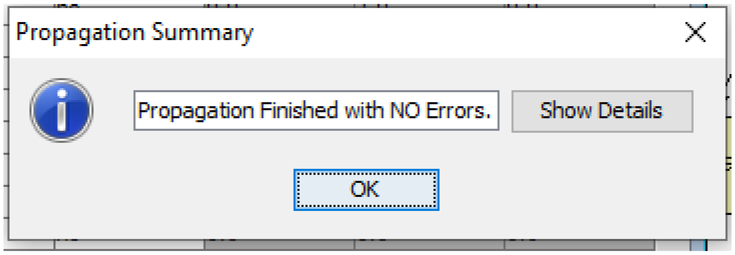
To propagate a single row (a case), double click on the row header for the particular row. Alternatively, right click on the row header and select 'Propagate selected rows' in the row header menu.



#	D	A	S	L	T	E	P(L=yes)	P(B=yes)	P(T=yes)
0					no	no			
1					no	no			
2					no	no			
3					no	no			
4					no	no			
5					no	no			
6					no	no			
7					no	no			
8					no	no			
9					no	no			
10	no	no	yes	no	no	no			
11	no	no		no	no	no			
12	no	no	no	no	no	no			
13	yes	no	yes	no	no	no			
14	no	no	yes	no	no	no			
15	no	no	no	no	no	no			
16	no	no	no	no	no	no			
17	no	no	no	no	no	no			
18	no	no	yes	no	no	no			
19	yes	no		no	no	no			
20	yes	no	no	no	no	no			
21	no	no	yes	no	no	no			
22	yes	no	yes	no	no	no			
23	no	no	yes	no	no	no			
24	no	no	no	no	no				
25	yes	no	yes	no	no	no			

Figure 8: Right-click the row header to enter the row header menu where it is possible to propagate a selected set of rows. Alternatively, to propagate a particular row, double click on the row header for that row.

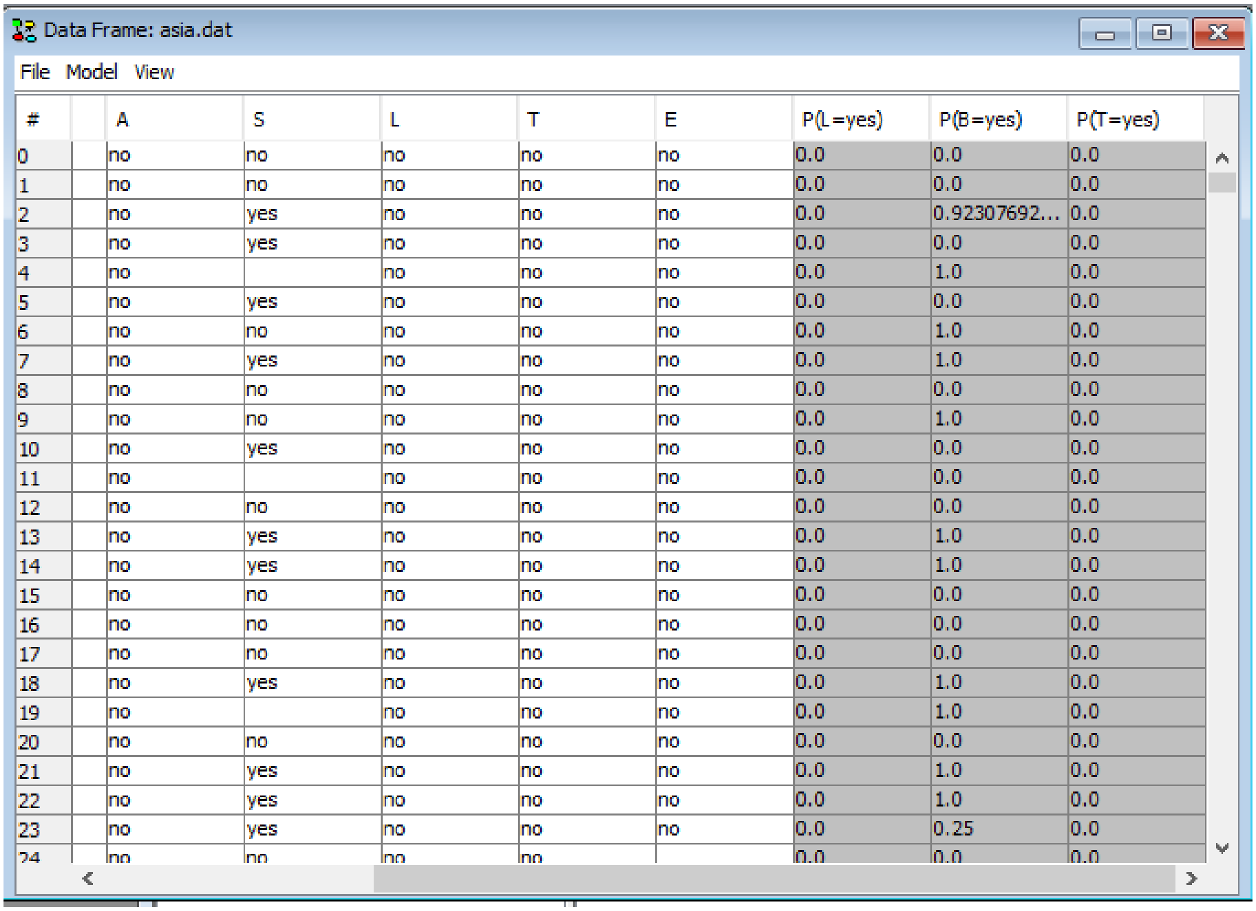
Rows can also be propagated in batch by selecting a number or rows, right-clicking the row header and choose propagate selected rows. Or just right-clicking the row header and choosing propagate ALL rows. When rows are being propagated in batch, a dialog appears displaying the status of the batch propagation job.


Propagation Summary

Propagation Finished with NO Errors.

Figure 9: Batch propagation job status.

When the batch propagation job completes, information have been recorded for all rows that could be inserted and propagated in the network model.



#		A	S	L	T	E	P(L=yes)	P(B=yes)	P(T=yes)
0		no	no	no	no	no	0.0	0.0	0.0
1		no	no	no	no	no	0.0	0.0	0.0
2		no	yes	no	no	no	0.0	0.92307692...	0.0
3		no	yes	no	no	no	0.0	0.0	0.0
4		no		no	no	no	0.0	1.0	0.0
5		no	yes	no	no	no	0.0	0.0	0.0
6		no	no	no	no	no	0.0	1.0	0.0
7		no	yes	no	no	no	0.0	1.0	0.0
8		no	no	no	no	no	0.0	0.0	0.0
9		no	no	no	no	no	0.0	1.0	0.0
10		no	yes	no	no	no	0.0	0.0	0.0
11		no		no	no	no	0.0	0.0	0.0
12		no	no	no	no	no	0.0	0.0	0.0
13		no	yes	no	no	no	0.0	1.0	0.0
14		no	yes	no	no	no	0.0	1.0	0.0
15		no	no	no	no	no	0.0	0.0	0.0
16		no	no	no	no	no	0.0	0.0	0.0
17		no	no	no	no	no	0.0	0.0	0.0
18		no	yes	no	no	no	0.0	1.0	0.0
19		no		no	no	no	0.0	1.0	0.0
20		no	no	no	no	no	0.0	0.0	0.0
21		no	yes	no	no	no	0.0	1.0	0.0
22		no	yes	no	no	no	0.0	1.0	0.0
23		no	yes	no	no	no	0.0	0.25	0.0
24		no	no	no	no		0.0	0.0	0.0

Figure 10: All rows has been propagated and information recorded.

4.8.10 Data Sequences in HUGIN

Since HUGIN version 8.3 the data frame HUGIN has had support for data in sequences (or time series). When the associated runtime domain is a DBN-model HUGIN already assumes that the Data set is a sequence of observations. However, if the data file contains multiple sequences then each sequence must have a sequence id (SID). The SID is a column in the data set with with text identifiers that are the same for all observations belonging to the same sequence. For SIDs you must be aware of the following properties:

- All observations belonging to the same sequence must be organized in sequence in the data set. As soon as a new SID is observed, it HUGIN assumes that a new sequence begins.
- SIDs are not required to be unique throughout the data set. If two sequences share a SID they will be treated as different sequences.

The use of a Sequence Identifier has impact on how many things work throughout the data frame. Here is a list of some of the functions it influences:

- **Data Propagation**: In batch propagation mode, HUGIN resets the time window back to 0 and reinitializes the domain each time a SID is not similar to the SID of the previous case.
- **Transform to time slices**: Transformation of a dataset into time slices transforms sequence by sequence in stead of entire data set.

- **Transform data sequences to cases for learning**: Possibility to transform a set of sequences into a dataset containing one row for each sequence. The number of columns will be the number of cases in the longest sequence times the original number of columns.

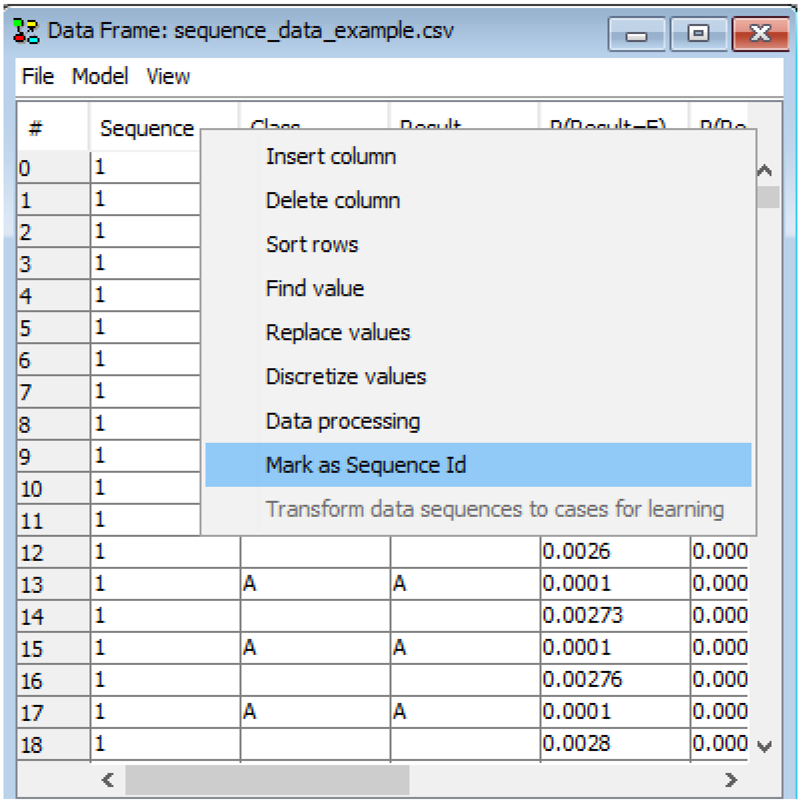
Get ready

By following these links you can read how to:

- *open your data file* (page 361),
- *associate your data file with a run-mode DBN model, and* (page 362),
- *convert the data to time windows* (page 363).

Pointing out the SID column

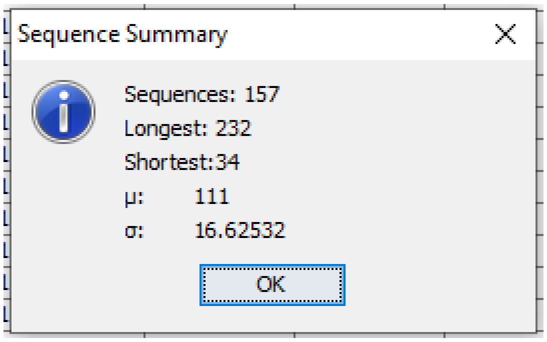
To point out the SID column in a data set with multiple sequences right click on the column containing the sequence identifiers (See Figure 1).



#	Sequence	Class	Result	P(Result=F)	P(Re
0	1				
1	1				
2	1				
3	1				
4	1				
5	1				
6	1				
7	1				
8	1				
9	1				
10	1				
11	1				
12	1			0.0026	0.000
13	1	A	A	0.0001	0.000
14	1			0.00273	0.000
15	1	A	A	0.0001	0.000
16	1			0.00276	0.000
17	1	A	A	0.0001	0.000
18	1			0.0028	0.000

Figure 1: Right-click on the column and select "Mark as Sequence id" to mark a column as SID.

Once the SID column has been pointed out, or whenever an new SID column is choosen a dialog box appears with statistics on the sequences found in the data set (See Figure 2).

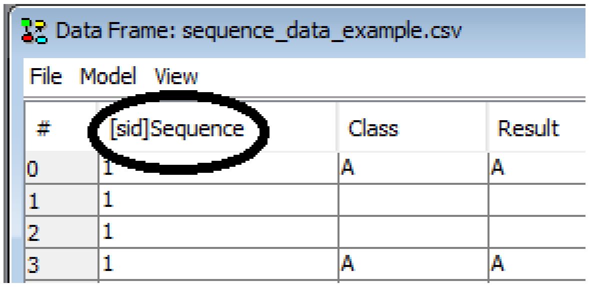


Sequence Summary

Sequences: 157
Longest: 232
Shortest:34
μ: 111
σ: 16.62532

Figure 2: Once a new SID has been selected, a dialog appears with some statistics on the observed sequences.

Once the SID column obtains a "[sid]" tag which indicates that the column is the SID (See Figure 3). The data set can be saved with the SID choosen and once it is re-loaded HUGIN will read the tag and know that it is the SID.



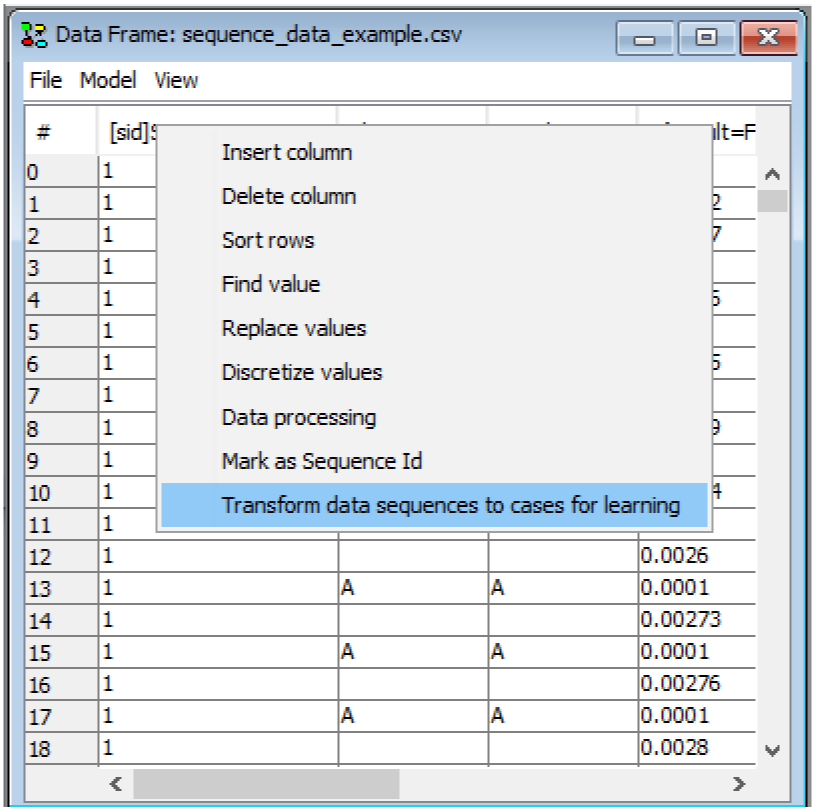
Data Frame: sequence_data_example.csv

#	[sid]Sequence	Class	Result
0	1	A	A
1	1		
2	1		
3	1	A	A

Figure 3: The appended "[sid]" tag on the column name tells HUGIN that the column is the SID.

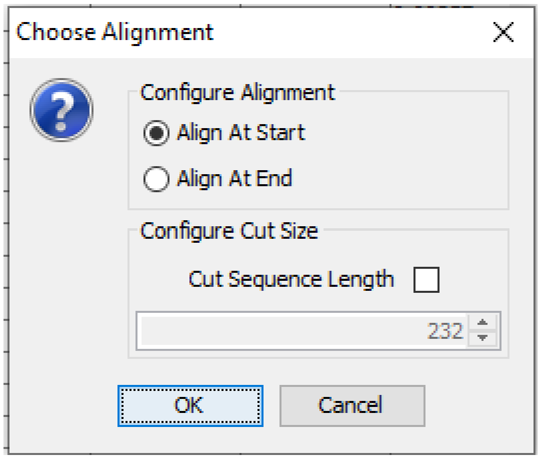
Transform Sequences to cases for learning

Once the SID column has been pointed out, the option "Transform dataset to cases for learning" becomes available (See Figure 3). This option transforms the dataset into a new dataset in which each sequence is represented as individual cases. The number of columns will correspond to the original number of columns times the number of observations in the longest sequence.



#	[sid]S			lt=F
0	1			
1	1			2
2	1			7
3	1			
4	1			5
5	1			
6	1			5
7	1			
8	1			9
9	1			
10	1			4
11	1			
12	1			0.0026
13	1	A	A	0.0001
14	1			0.00273
15	1	A	A	0.0001
16	1			0.00276
17	1	A	A	0.0001
18	1			0.0028

Figure 4: As soon as a column has a "[sid]" tag it is possible to transform the dataset to a dataset containing a row for each sequence.


Choose Alignment

Configure Alignment

● Align At Start

○ Align At End

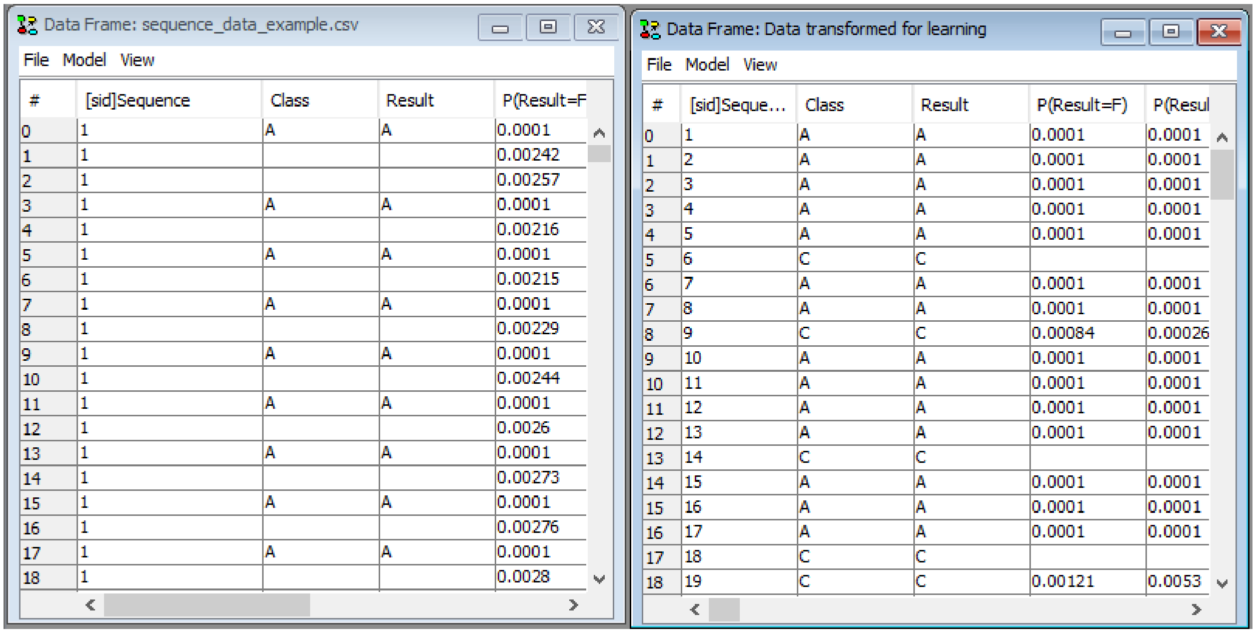
Configure Cut Size

Cut Sequence Length □

232

Figure 5: Choose whether to align the data to the right or to the left and whether to cut away columns with lowest or highest time label respectively.

The transformed dataset opens in a new data frame (See Figure 6). The data set can now be inspected, propagated or saved and opened using the *Learning Wizard* (page 267).


Data Frame: sequence_data_example.csv
File Model View

#	[sid]Sequence	Class	Result	P(Result=F
0	1	A	A	0.0001
1	1			0.00242
2	1			0.00257
3	1	A	A	0.0001
4	1			0.00216
5	1	A	A	0.0001
6	1			0.00215
7	1	A	A	0.0001
8	1			0.00229
9	1	A	A	0.0001
10	1			0.00244
11	1	A	A	0.0001
12	1			0.0026
13	1	A	A	0.0001
14	1			0.00273
15	1	A	A	0.0001
16	1			0.00276
17	1	A	A	0.0001
18	1			0.0028

Data Frame: Data transformed for learning
File Model View

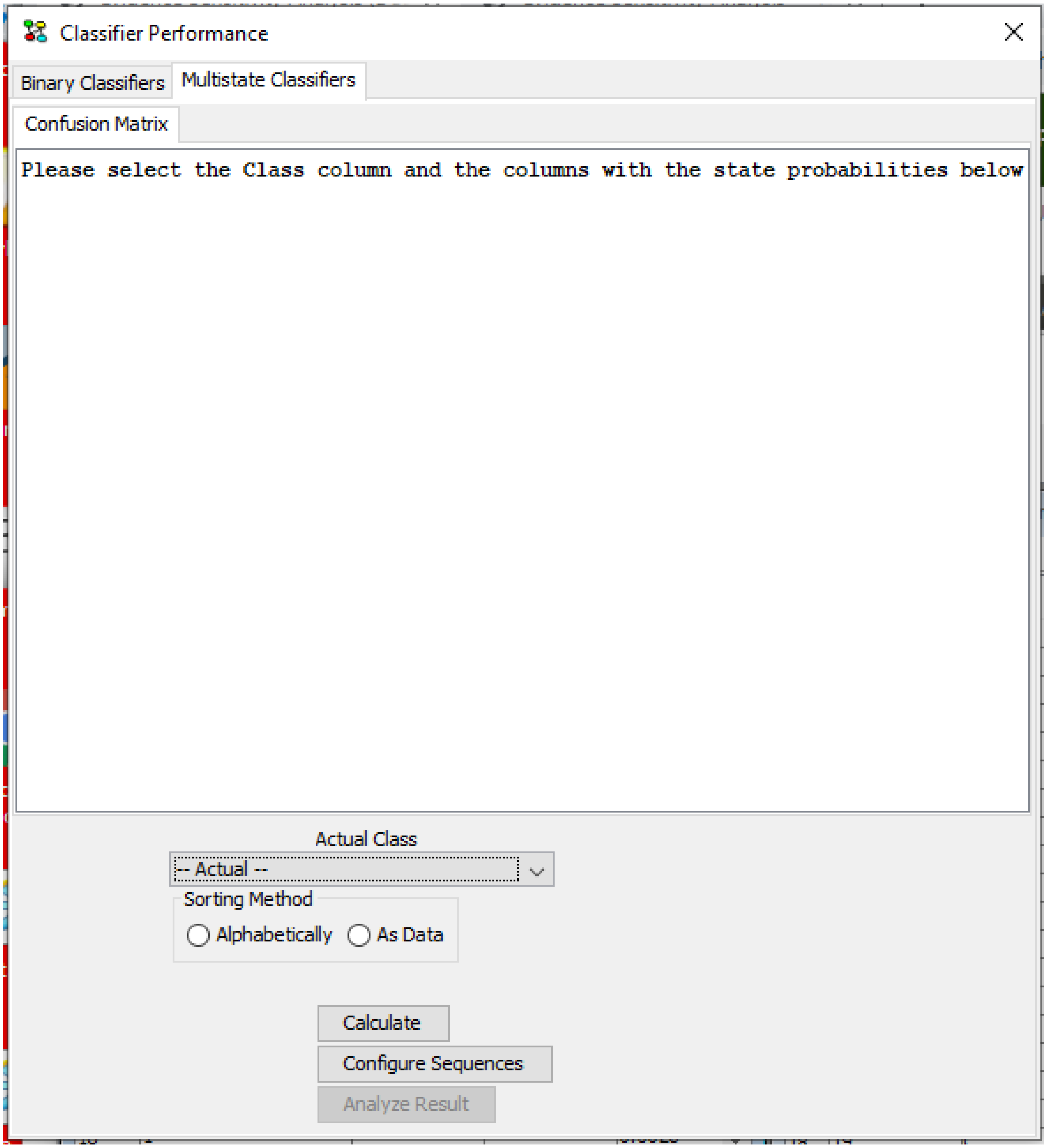
#	[sid]Seque...	Class	Result	P(Result=F)	P(Resul
0	1	A	A	0.0001	0.0001
1	2	A	A	0.0001	0.0001
2	3	A	A	0.0001	0.0001
3	4	A	A	0.0001	0.0001
4	5	A	A	0.0001	0.0001
5	6	C	C		
6	7	A	A	0.0001	0.0001
7	8	A	A	0.0001	0.0001
8	9	C	C	0.00084	0.00026
9	10	A	A	0.0001	0.0001
10	11	A	A	0.0001	0.0001
11	12	A	A	0.0001	0.0001
12	13	A	A	0.0001	0.0001
13	14	C	C		
14	15	A	A	0.0001	0.0001
15	16	A	A	0.0001	0.0001
16	17	A	A	0.0001	0.0001
17	18	C	C		
18	19	C	C	0.00121	0.0053

Figure 6: The transforme dataset will open in a new frame.

Classifier Performance on Sequences

With a SID column, the Classifier Performance tool has been extended with ability to evaluate classification of sequences. In this tool, sequences are assumed to appear in sequence and not as individual cases as above. The Classifier Performance tool is accessed by right-clicking on a cell in the data matrix and then selecting “Classifier Performance”.

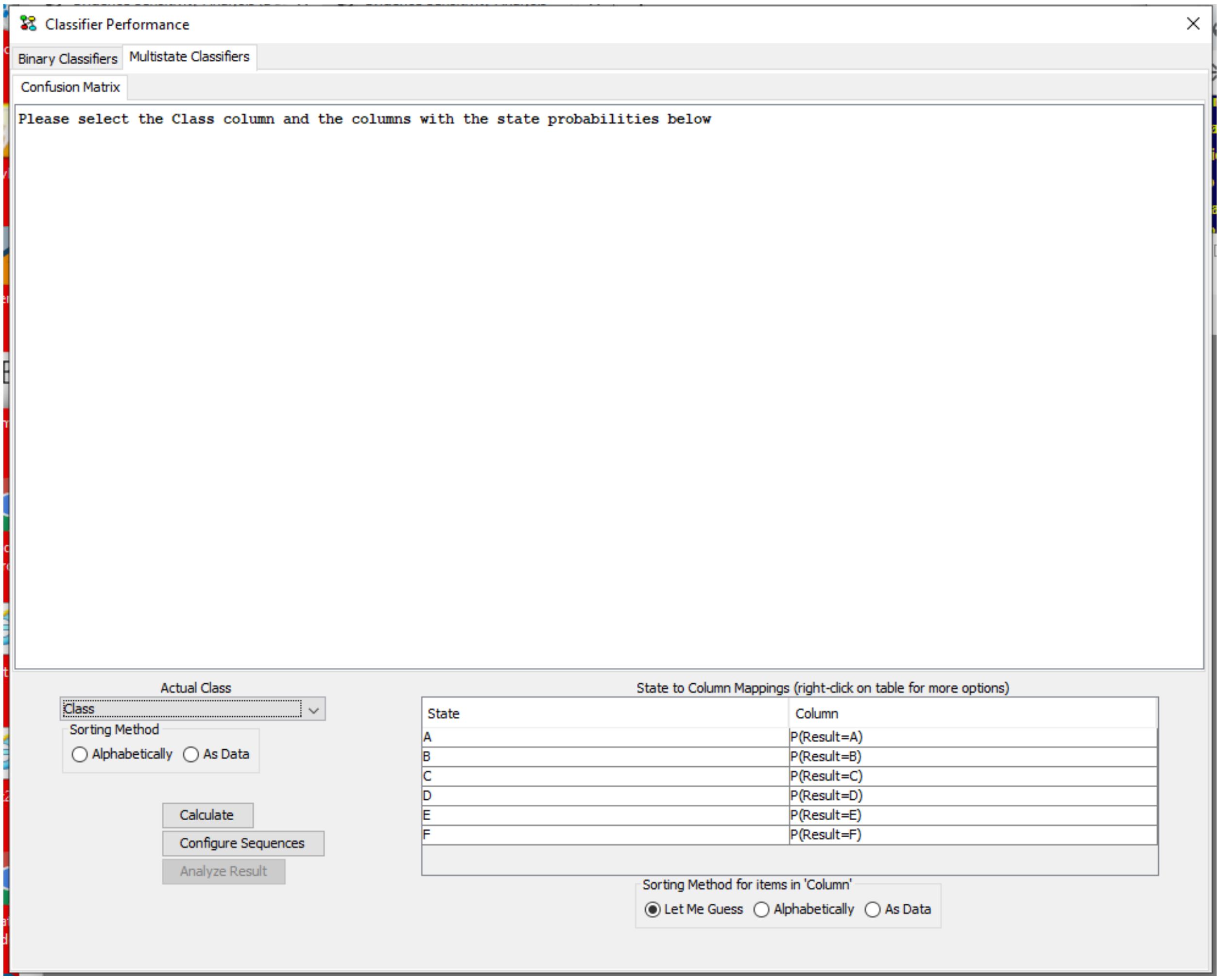
In the “Multistate Classifiers” tab notice that the “Configure Sequences” button is now visible. It becomes visible when a SID column is selected on the dataset (See Figure 7).



```
Please select the Class column and the columns with the state probabilities below
```

Figure 7: In the Classifier Performance tool, Multistate Classifiers tab, the "Configure Sequences" button is visible when operating with data sequences.

Under "Actual Class" select the column containing the actual class in the data set (See Figure 8). As soon as a column is selected HUGIN tries to "guess" the columns containing the probabilities of each states. HUGIN's guess can be reviewed and modified in the "Columns with State probabilities" panel. The guess is based on a penalizing algorithm that assigns lowest penalty to columns with names matching "P([.]?=StateName from Actual Class)", where [.]? is any pattern containing at least 1 character. Arbitrary text can be appended, but it will increase the penalty.



```
Please select the Class column and the columns with the state probabilities below
```

Actual Class: Class

Sorting Method: ○ Alphabetically ○ As Data

State to Column Mappings (right-click on table for more options)

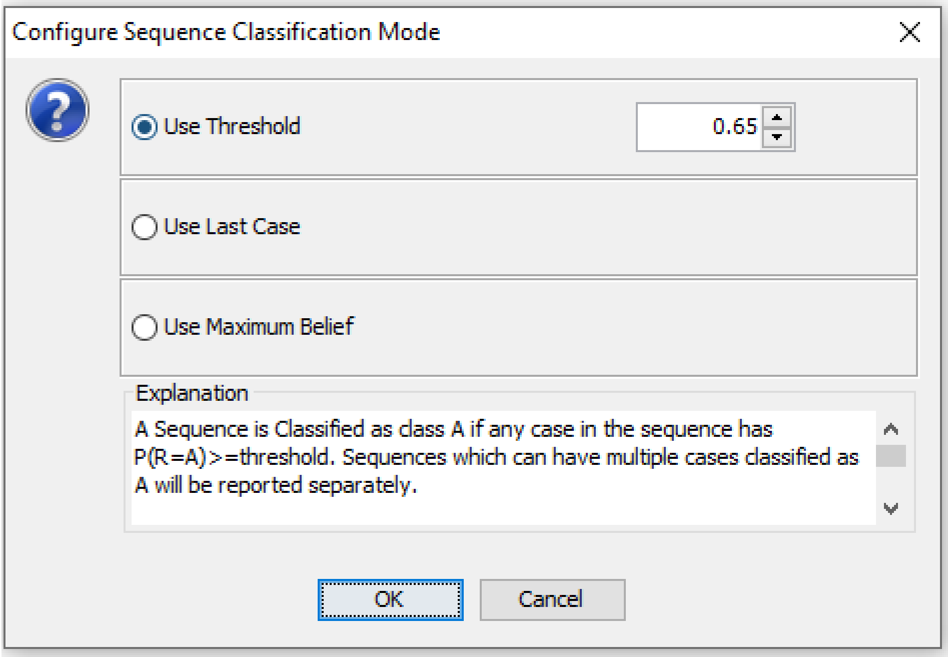
State	Column
A	P(Result=A)
B	P(Result=B)
C	P(Result=C)
D	P(Result=D)
E	P(Result=E)
F	P(Result=F)

Sorting Method for items in 'Column': ● Let Me Guess ○ Alphabetically ○ As Data

Figure 8: When a class has been selected, HUGIN attempts to auto fill out the columns in "Columns with state probabilities".

The classification evaluation method can be configured under "Configure Sequences". Here there are 2 options:

- **Use Threshold**: Classification mode that assigns the sequence to a class as soon as the probability of one class anywhere in the sequence is above a selected threshold (See Figure 9).
- **Use Last Case**: Classification mode that considers the state probabilities of the last case in the sequence only (See Figure 10).



Configure Sequence Classification Mode

● Use Threshold 0.65

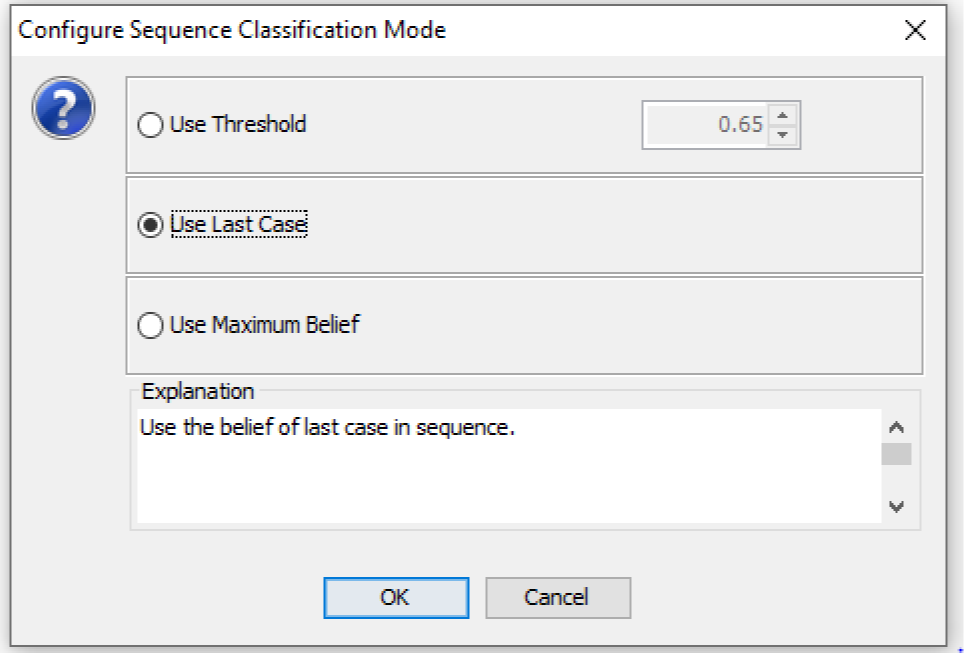
○ Use Last Case

○ Use Maximum Belief

Explanation

A Sequence is Classified as class A if any case in the sequence has P(R=A)>=threshold. Sequences which can have multiple cases classified as A will be reported separately.

Figure 9: The transforme dataset will open in a new frame.



Configure Sequence Classification Mode

○ Use Threshold 0.65

● Use Last Case

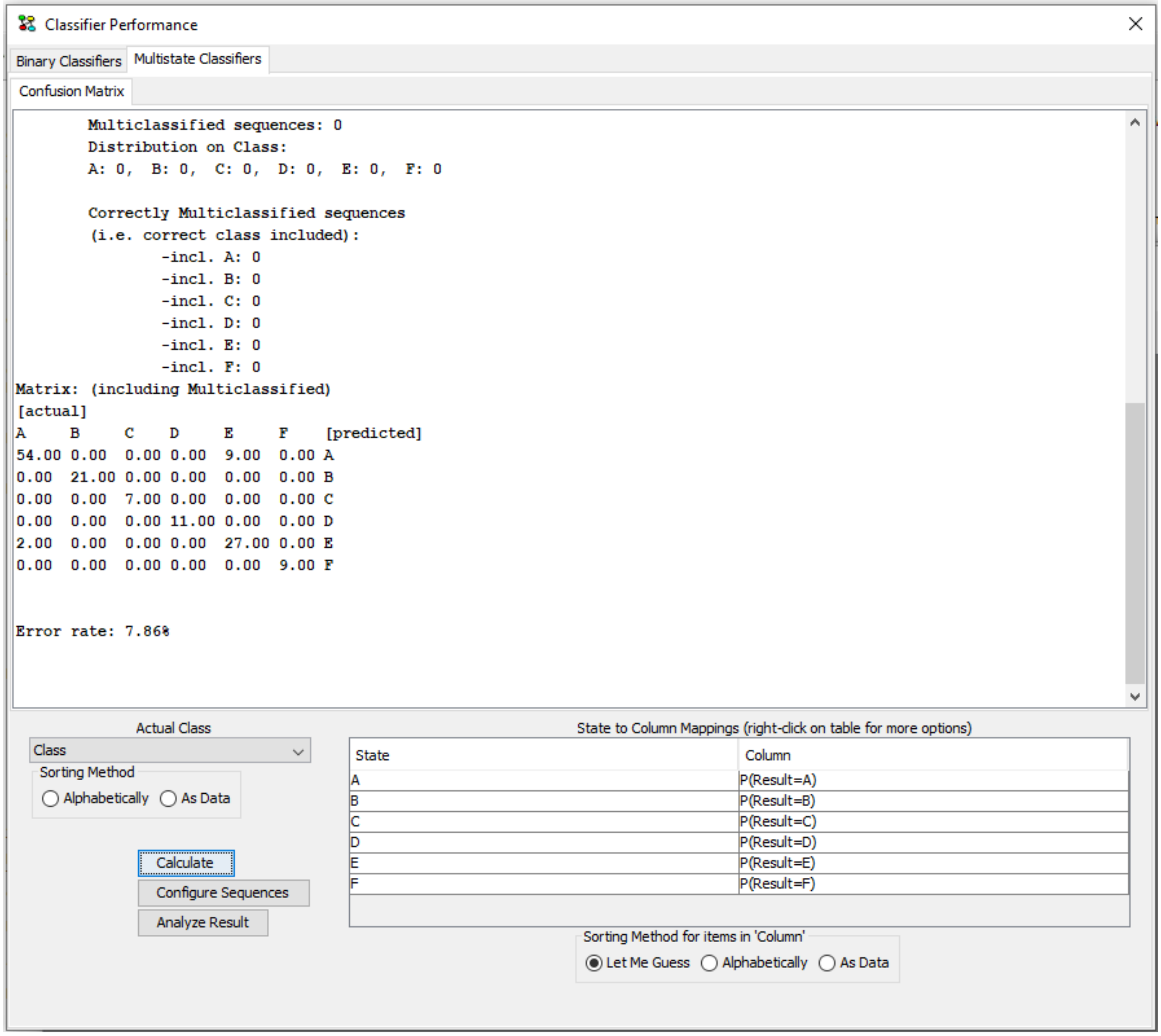
○ Use Maximum Belief

Explanation

Use the belief of last case in sequence.

Figure 10: The transforme dataset will open in a new frame.

Click on “Calculate” to calculate the result. The text in the frame explains what the evaluation has counted. Notice that you may need to scroll in order to read all results and be aware that not all data may be included in the confusion matrix. The latter is due the the possibility of a sequence belonging to multiple classes. Such cases are summed up in the results below the confusion matrix.



```
        Multiclassified sequences: 0
        Distribution on Class:
        A: 0,  B: 0,  C: 0,  D: 0,  E: 0,  F: 0

        Correctly Multiclassified sequences
        (i.e. correct class included):
                -incl. A: 0
                -incl. B: 0
                -incl. C: 0
                -incl. D: 0
                -incl. E: 0
                -incl. F: 0
Matrix: (including Multiclassified)
[actual]
A     B     C    D     E     F    [predicted]
54.00 0.00  0.00 0.00  9.00  0.00 A
0.00  21.00 0.00 0.00  0.00  0.00 B
0.00  0.00  7.00 0.00  0.00  0.00 C
0.00  0.00  0.00 11.00 0.00  0.00 D
2.00  0.00  0.00 0.00  27.00 0.00 E
0.00  0.00  0.00 0.00  0.00  9.00 F


Error rate: 7.86%
```

Actual Class: Class

Sorting Method: ○ Alphabetically ○ As Data

State to Column Mappings (right-click on table for more options)

State	Column
A	P(Result=A)
B	P(Result=B)
C	P(Result=C)
D	P(Result=D)
E	P(Result=E)
F	P(Result=F)

Sorting Method for items in 'Column': ● Let Me Guess ○ Alphabetically ○ As Data

Figure 11: The transforme dataset will open in a new frame.

A click on the button "Analyze Result" will open a new dataframe containing the data used for the construction of the confusion matrix and the count of multi-classifications. This dataset is opened in a new frame. It contains the following columns:

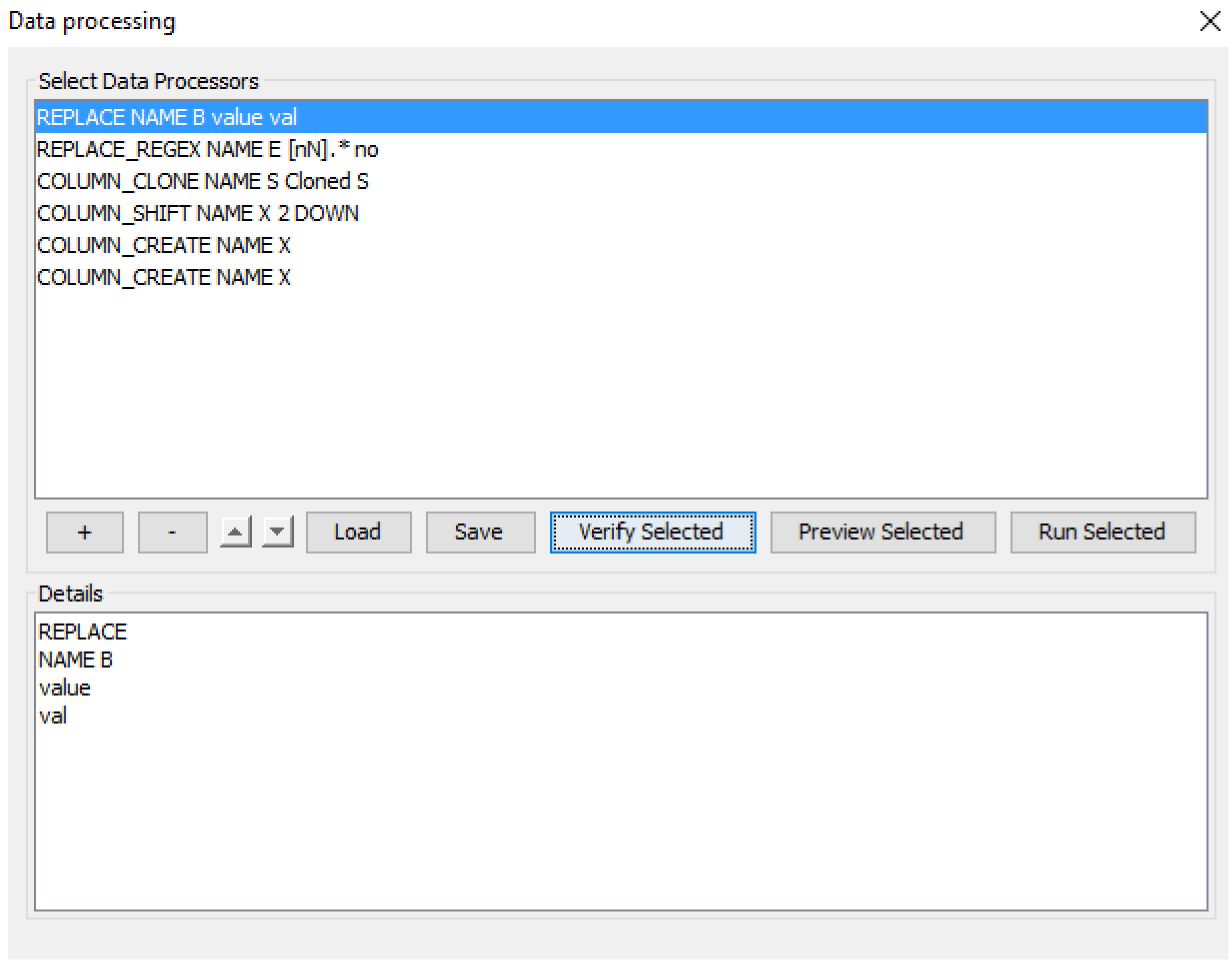
- Sequence id
- Sequence index in original data set
- Sequence's true class
- List of classes the particular sequence was assigned to
- For each class C, the relative index in the sequence where the sequence was recognezed as belonging to C.

4.8.11 Data Processing

Data Processing is the task of applying a set of transformations to the data items in a data set. Data Processing functionallity is available using the data processing tool, which can be launched from the right-click context menu in the data matrix.

The information used to perform a transformation is called a data processor description. Using the data processing tool, one can define data processors as well as applying them. The tool supports loading and saving data processor descriptions, which is useful for dealing with different data sets that require the same transformations to be usable in HUGIN.

The data processing tool is displayed

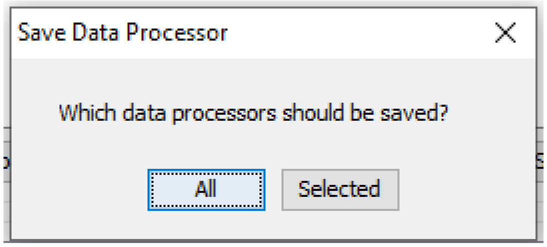


REPLACE NAME B value val
REPLACE_REGEX NAME E [nN].* no
COLUMN_CLONE NAME S Cloned S
COLUMN_SHIFT NAME X 2 DOWN
COLUMN_CREATE NAME X
COLUMN_CREATE NAME X

REPLACE
NAME B
value
val

Figure 1: The data processor tool - a number of data processor descriptions has been loaded. The top half of the window contains a list of data processors. When a data processor is selected, the description can be edited in the details text field located in the bottom half of the window.

Use the function buttons '+' and '-' to create and delete data processors and the arrow buttons (▲ and ▼) to rearrange currently selected data processors. Use the load and save buttons to store data processor descriptions in files. One has the option to save all data processors or only selected when clicking the save button, see Figure 2.



Save Data Processor

Which data processors should be saved?

Figure 2: Save all or selected data processors

Use the 'Verify Selected' button to get an overview of possible problems with the selected processors and use the 'Preveiw Selected' button to see the result of the currently selected processors on the first 200 rows. It is recommended to use these tools before applying a data processor, to see how it performs. Click the 'Run Selected' button to apply a data processor.

Data Processor Descriptions

A data processor description is a list of text lines, containing three pieces of information: the data processor type, a column specifier and any parameters for the data processor:

```
<data processor type>
<column specifier>
<argument>*
```

The data processor type can be one of:

```
REPLACE
REPLACE_REGEX
TO_UPPER
TO_LOWER
DISCRETIZE_MANUAL
DISCRETIZE_EQUAL_DISTRIBUTION
DISCRETIZE_IEM
COLUMN_DELETE
COLUMN_CREATE
COLUMN_CLONE
COLUMN_SHIFT
```

The column specifier can select a single column based on the column name:

```
NAME <column name>
```

Or a set of columns where the column names matches a regular expression *(regular expressions described further below)*:

```
SELECT <regular expression>
```

The arguments depend on the chosen data processor type:

- REPLACE:

```
<match text string>
<replacement text string>
```

- REPLACE_REGEX:

```
<regular expression>
<replacement text string>
```

- TO_UPPER: none
- TO_LOWER: none
- DISCRETIZE_MANUAL: The intervals, specified as a list of interval boundaries:

```
<lower bound first interval>
<upper bound previous interval/lower bound next interval>*
<upper bound last interval>
```

- DISCRETIZE_EQUAL_DISTRIBUTION: The number of target intervals:

```
<number of states>
```

- DISCRETIZE_EQUAL_DISTRIBUTION: The number of target intervals:

```
<number of states>
```

- DISCRETIZE_IEM

```
<column name>
```

- COLUMN_DELETE: none
- COLUMN_CREATE: none
- COLUMN_CLONE

```
<new column name or prefixes>
```

- COLUMN_SHIFT

```
<rows to shift>
<direction to shift>
```

Regular expressions

A regular expression is a pattern used to match a sequence of characters. The pattern matching rules used in HUGIN follow the java regular expressions pattern matching from java.util.regex.Pattern[12].

A summary of selected regular-expression constructs:

Characters

```
x       The character x
\\      The backslash character
```

Character classes

[12] http://docs.oracle.com/javase/1.5.0/docs/api/java/util/regex/Pattern.html

```
[abc]           a, b, or c (simple class)
[^abc]          Any character except a, b, or c (negation)
[a-zA-Z]        a through z or A through Z, inclusive (range)
[a-d[m-p]]      a through d, or m through p: [a-dm-p] (union)
```

Predefined character classes

```
.       Any character
\d      A digit: [0-9]
\D      A non-digit: [^0-9]
\s      A whitespace character: [ \t\n\x0B\f\r]
\S      A non-whitespace character: [^\s]
\w      A word character: [a-zA-Z_0-9]
\W      A non-word character: [^\w]
```

Greedy quantifiers

```
X?      X, once or not at all
X*      X, zero or more times
X+      X, one or more times
```

Logical operators

```
XY      X followed by Y
X|Y     Either X or Y
(X)     X, as a capturing group
```

The backslash character (’') serves to introduce escaped constructs, as defined in the table above, as well as to quote characters that otherwise would be interpreted as unescaped constructs. Thus the expression \ matches a single backslash and { matches a left brace.

Examples of regular expressions:

```
[n].*           Match any string that begins with the character 'n', e.g. 'not', 'nothing
↪' etc.
\d+(\.\d*)?     Match any string that is a number, on the form 1 or 1.234 etc.
```

Creating a Data Processor

Click the '+' button to create a new data processor, the dialog in Figure 3 appears.

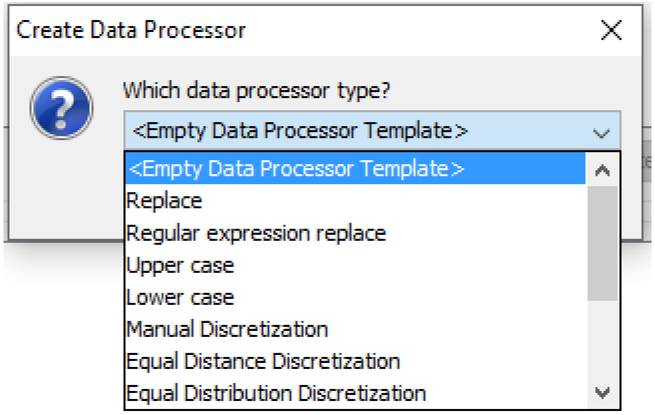

Create Data Processor
Which data processor type?
<Empty Data Processor Template>
<Empty Data Processor Template>
Replace
Regular expression replace
Upper case
Lower case
Manual Discretization
Equal Distance Discretization
Equal Distribution Discretization

Figure 3: Select a data processor type

- To manually enter a data processor description, choose the empty template.
- To create a new data processor using a guided approach, select any of the data processor types.

The guided approach consists of a sequence of questions and finally automatic generation of the resulting data processor description. After choosing a data processor type, one must select a target column in the data set. See Figure 4.

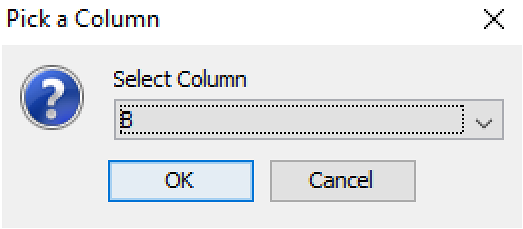


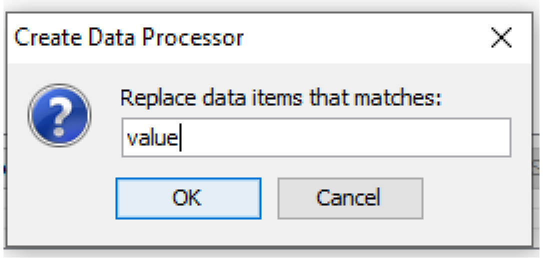
Figure 4: Select target column for the new data processor

Data Processor: Replace

This data processor replaces any data items that matches a specific text string, with a replacement text string. The REPLACE data processor takes two parameters, the text string to match and the replacement text string.

```
REPLACE
NAME <column-name>
<match text string>
<replacement text string>
```

When creating the data processor using the guided approach, specify parameters in the dialogs in Figures 5 and 6.

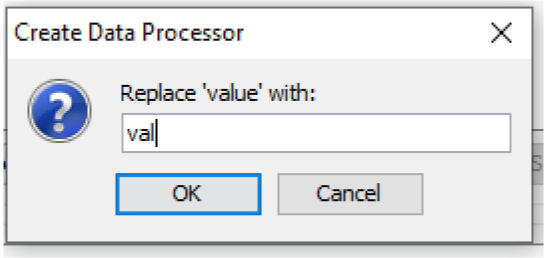


Create Data Processor

Replace data items that matches:

value

Figure 5: Enter the text string that should be replaced - any data items that matches the string 'value' will be replaced



Create Data Processor

Replace 'value' with:

val

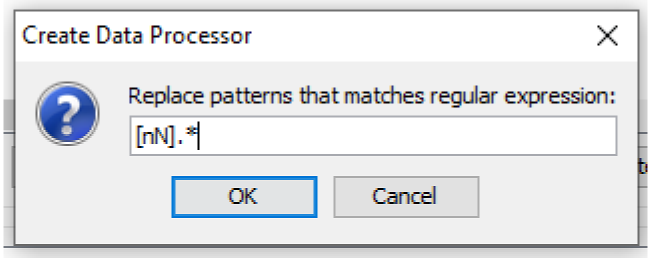
Figure 6: Enter the replacement text string - replace any data items that matches 'value' with 'val'

Data Processor: Regular expression replace

This data processor is similar to the normal REPLACE data processor, except that data items are matched using a regular expression instead of a fixed text string. The REPLACE_REGEX data processor takes two parameters, the regular expression used to select matching data items and the replacement text string.

```
REPLACE_REGEX
NAME <column-name>
<regular expression>
<replacement text string>
```

When creating the data processor using the guided approach, specify parameters in the dialogs in Figures 7 and 8.

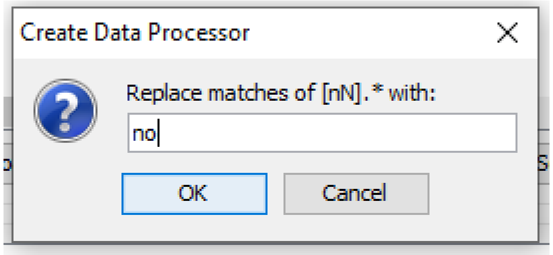


Create Data Processor

Replace patterns that matches regular expression:

[nN].*

Figure 7: Enter a regular expression that matches the desired data items - this regular expression matches any data item that begin with a lower- or uppercase n



Create Data Processor

Replace matches of [nN].* with:

no

Figure 8: Enter the replacement text string - replace any data items that matches the regular expression with 'no'

Data Processor: Upper case

This is a very simple data processor, which converts any lower case characters to upper case. The TO_UPPER data processor has no parameters.

```
TO_UPPER
NAME <column-name>
```

Data Processor: Lower case

This is a very simple data processor, which converts any upper case characters to lower case. The TO_LOWER data processor has no parameters.

```
TO_LOWER
NAME <column-name>
```

Data Processor: Manual Discretization

This data processor applies a discretization to all data items. The DISCRETIZE_MANUAL data processor takes a variable number of parameters, namely target intervals specified as a list of interval boundaries:

```
DISCRETIZE_MANUAL
NAME <column-name>
<lower bound first interval>
<upper bound previous interval/lower bound next interval>*
<upper bound last interval>
```

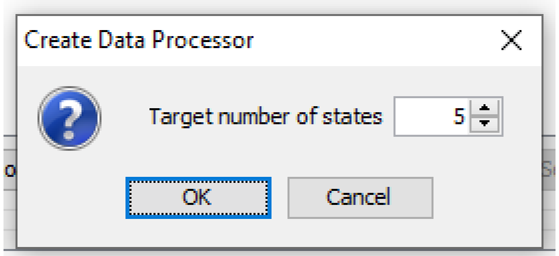
When creating the data processor using the guided approach, the *discretization tool* (page 434) is spawned to aid specifying the intervals.

Data Processor: Equal Distribution Discretization

This data processor applies a discretization to all data items. The target intervals are dynamically generated based on all the numeric data items in the column, such that each interval contain approximately the same number of data items. The DISCRETIZE_EQUAL_DISTRIBUTION data processor takes a single parameter, the number of target intervals.

```
DISCRETIZE_EQUAL_DISTRIBUTION
NAME <column-name>
<number of states>
```

When creating the data processor using the guided approach, specify parameters in the dialog in Figure 9.



Target number of states 5

Figure 9: Specify the number of intervals

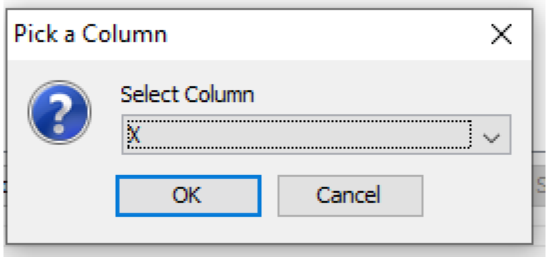
Depending on how well the data values are scattered, the number of intervals may be pruned in order to make the resulting intervals equally distributed.

Data Processor: Information Entropy Minmization Discretization

This data processor applies a discretization to all data items. Target intervals are generated that minimizes entropy in an other discrete column (the target column). The DISCRETIZE_IEM data processor takes a single parameter, the target column on which to minimize the entropy (discrete valued).

```
DISCRETIZE_IEM
NAME <column-name>
<target column>
```

When creating the data processor using the guided approach, the target column is selected in the dialog in Figure 10.



Select Column X

Figure 10: Specify the target column on which to base the entropy minimization discretization.

Data Processor: Delete Column

This data processor deletes one or more columns. The COLUMN_DELETE data processor deletes a single column when a named column is specified or it deletes multiple columns that matches a pattern if a SELECT clause is used.

```
COLUMN_DELETE
NAME <column-name>
```

Data Processor: Create Column

This data processor creates a new empty column. The COLUMN_CREATE data processor creates a single column with the specified name. The create data processor cannot be combined with a SELECT clause.

```
COLUMN_CREATE
NAME <column-name>
```

Data Processor: Clone Column

This data processor clones one or more columns. The COLUMN_CLONE data processor takes a single parameter, which determines the name of the clones. When COLUMN_CLONE is combined with at SELECT clause the parameter is used as a prefix on the new names of the cloned columns. Otherwise the parameter is the name of the cloned column.

```
COLUMN_CLONE
NAME <column-name>
<new column name or prefixes>
```

When creating the data processor using the guided approach, the column to clone is selected in the dialog in Figure 11 while its name is specified in the dialog in Figure 12.

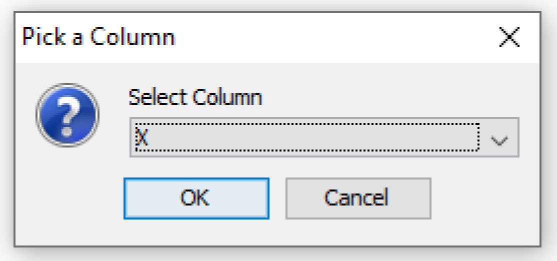


Figure 11: Specify the column to clone.

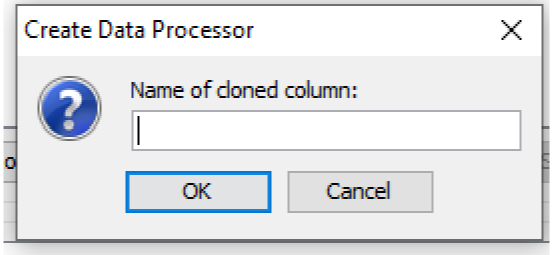


Figure 12: Specify the name of the cloned column.

© Copyright 2023, HUGIN EXPERT A/S

Data Processor: Shift Column Cells

This data processor shifts the cells of or more columns a specified number of cells up or down. A COLUMN_SHIFT data processor takes two parameters. The first determines the number of rows to shift the cells while the second determines whether to shift the cells up or down. When using the guided approach, the system asks how many rows to shift the cells down. However, by specifying a negative number the cells can be shifted up.

```
COLUMN_SHIFT
NAME <column-name>
<Number of rows (any integer in the interval ]-∞, ∞[) >
<UP or DOWN>
```

When creating the data processor using the guided approach, the number of rows to shift the column is selected in the dialog in Figure 13.

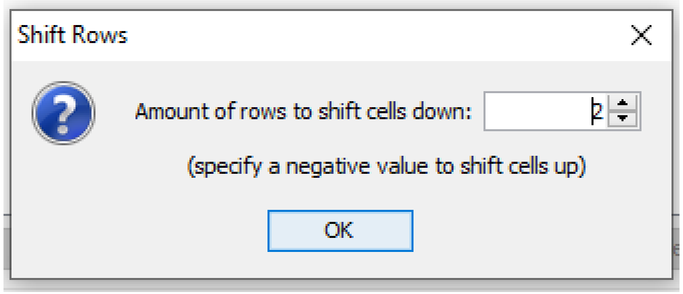

Shift Rows
Amount of rows to shift cells down:
2
(specify a negative value to shift cells up)
OK

Figure 13: Specify the number of rows to shift the selected column.

Selecting Data Processors

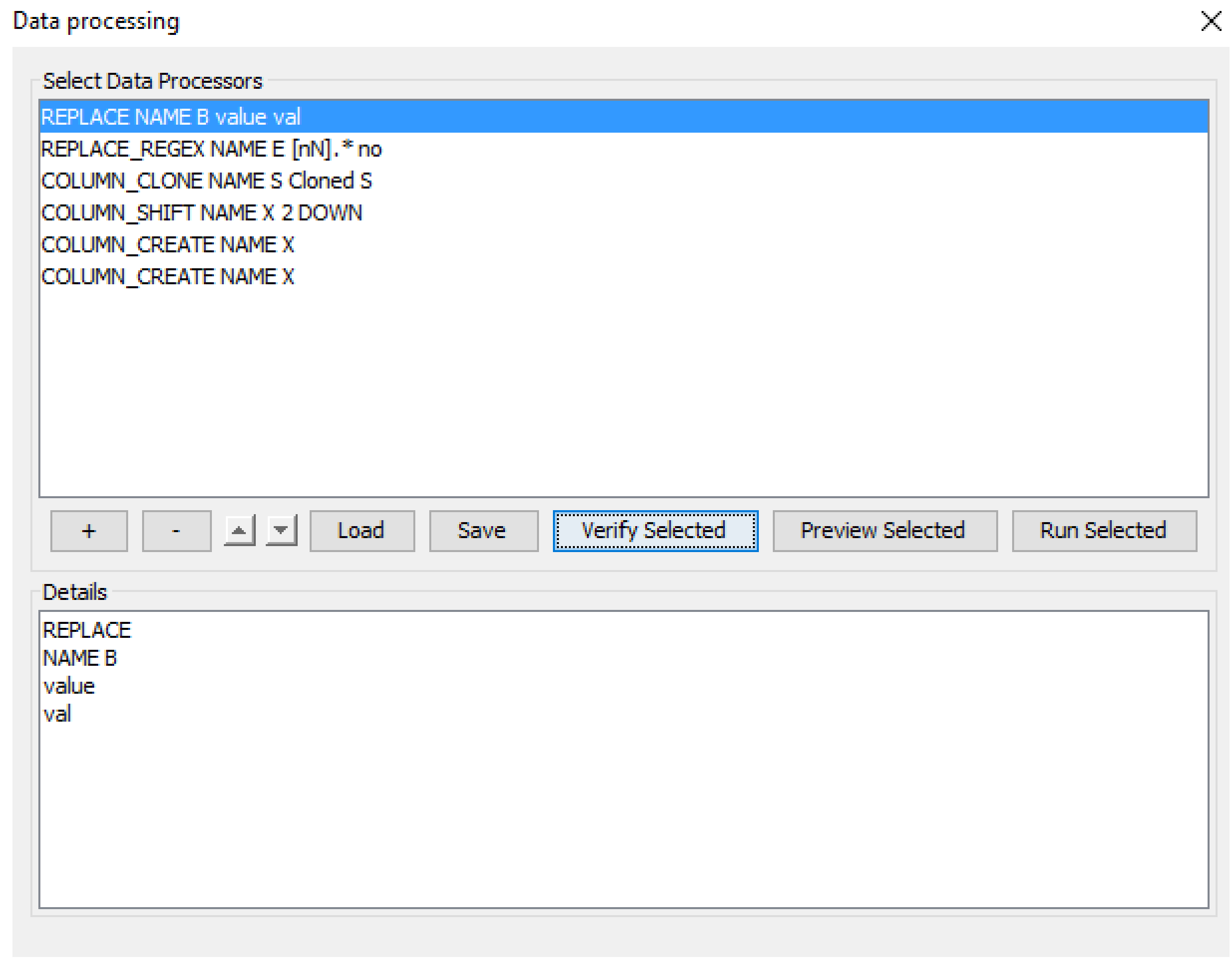
A Data Processor is selected by clicking on its description in the ‘Select Data Processors List. Multiple data processors can be selected by holding down the CTRL key while selecting data processors from the list. Note that when multiple data processors are selected, they will be processed in the order they appear in the list from top to bottom. Remember that the arrow buttons (▲ and ▼) allow you to rearrange data processors in the list.

Verify and Preview Data Processors

It is possible to verify and preview the consequences of running the selected preprocessors on your data before actually running them. This is a great help when writing and debugging a data processor description.

The verification tool gives you a summary of what the data processors will do to your data. This option will also give you warnings and error messages if there is anything you should be careful about or if there seems to be an error in your data processors.

To use the verification tool on your data processors, select the desired data processor(s) in the list (see Figure 14), and then click the ‘Verify Selected’ button.

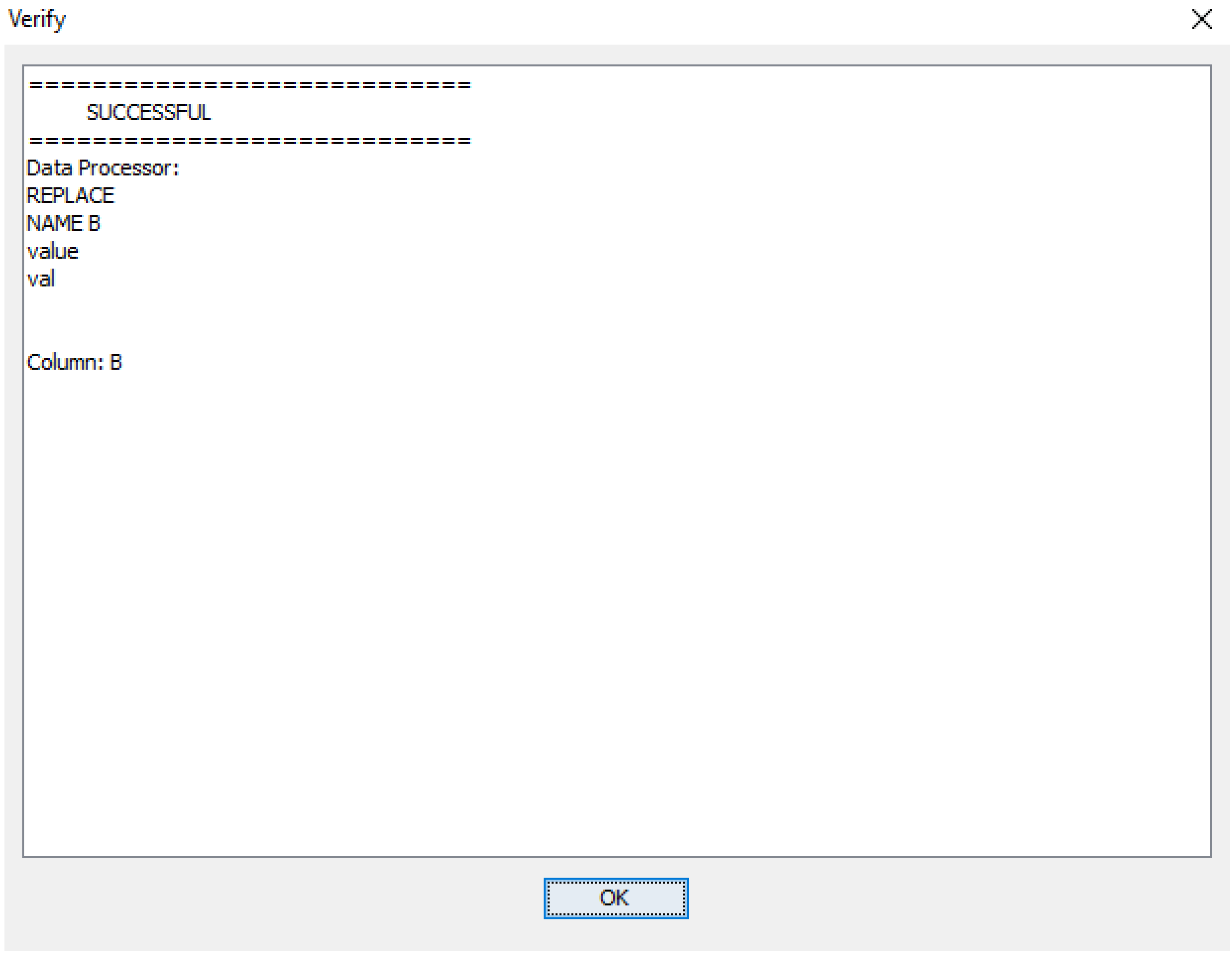


```
REPLACE NAME B value val
REPLACE_REGEX NAME E [nN].* no
COLUMN_CLONE NAME S Cloned S
COLUMN_SHIFT NAME X 2 DOWN
COLUMN_CREATE NAME X
COLUMN_CREATE NAME X
```

```
REPLACE
NAME B
value
val
```

Figure 14: Selecting a data processor - the selected data processor is of type REPLACE

The verification tool reports any errors/partial success and the transformations done by the selected data processor. The result of applying the verification tool can be seen in Figure 15.



```
============================
     SUCCESSFUL
============================
Data Processor:
REPLACE
NAME B
value
val


Column: B
```

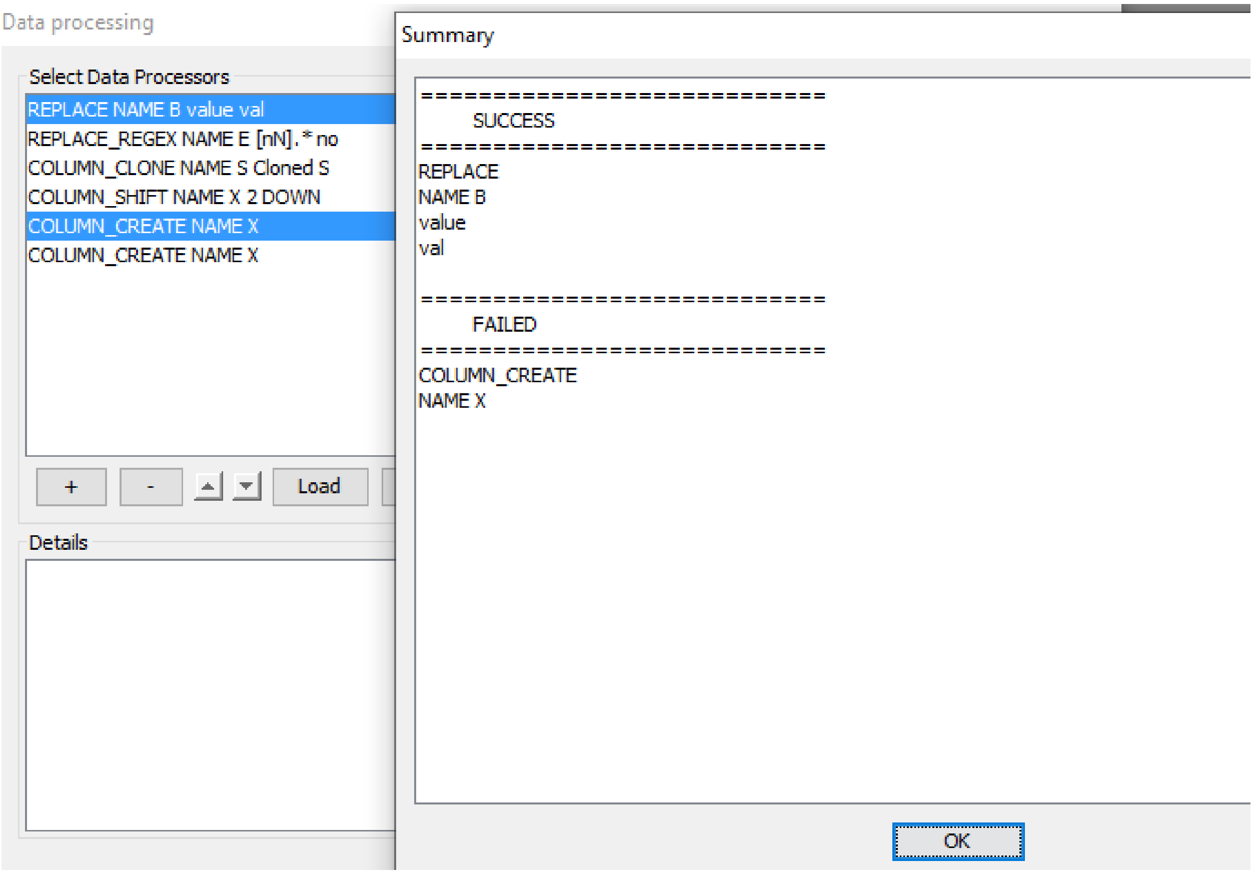
Figure 15: Verification tool result - see how selected data processors performs, inspect errors etc.

The preview tool copies the top 200 rows in the dataset and applies the selected preprocessors on them and displays the result in a new window which can be inspected.

An example of running the preview tool on 4 selected data processors is shown in Figure 16 and Figure 17. First, the window in Figure 16 indicates which processors succeeded and which processors failed.

In this example 3 of the data processors are listed under SUCCESS while 1 data processor is listed under FAILED. The result of running the data processors on the top 200 rows is outlined in the Preview table that follows when pressing ‘OK’ in the Summary window (Figure 16).

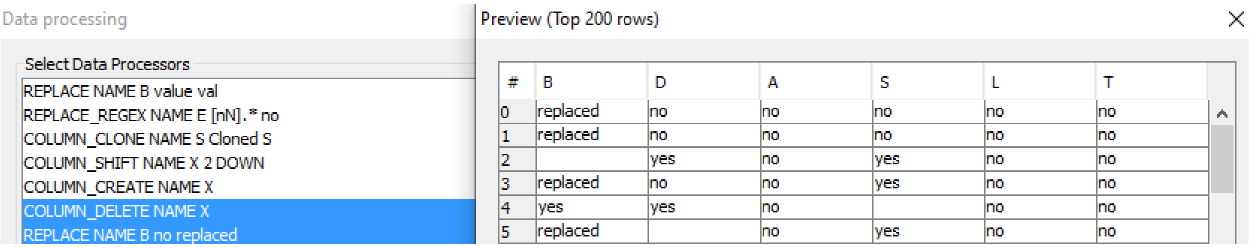
Using the verification tool on these 4 processors may indicate the cause of the error. Figure 18 shows that the COLUMN_CREATE processor failed since there were multiple columns named ‘X’.



REPLACE NAME B value val
REPLACE_REGEX NAME E [nN].* no
COLUMN_CLONE NAME S Cloned S
COLUMN_SHIFT NAME X 2 DOWN
COLUMN_CREATE NAME X
COLUMN_CREATE NAME X

```
============================
      SUCCESS
============================
REPLACE
NAME B
value
val

============================
      FAILED
============================
COLUMN_CREATE
NAME X
```

Figure 16: Summary of run of 2 selected data processors. The Summary indicates that the COLUMN_CREATE processor listed under FAILED did not succeed.



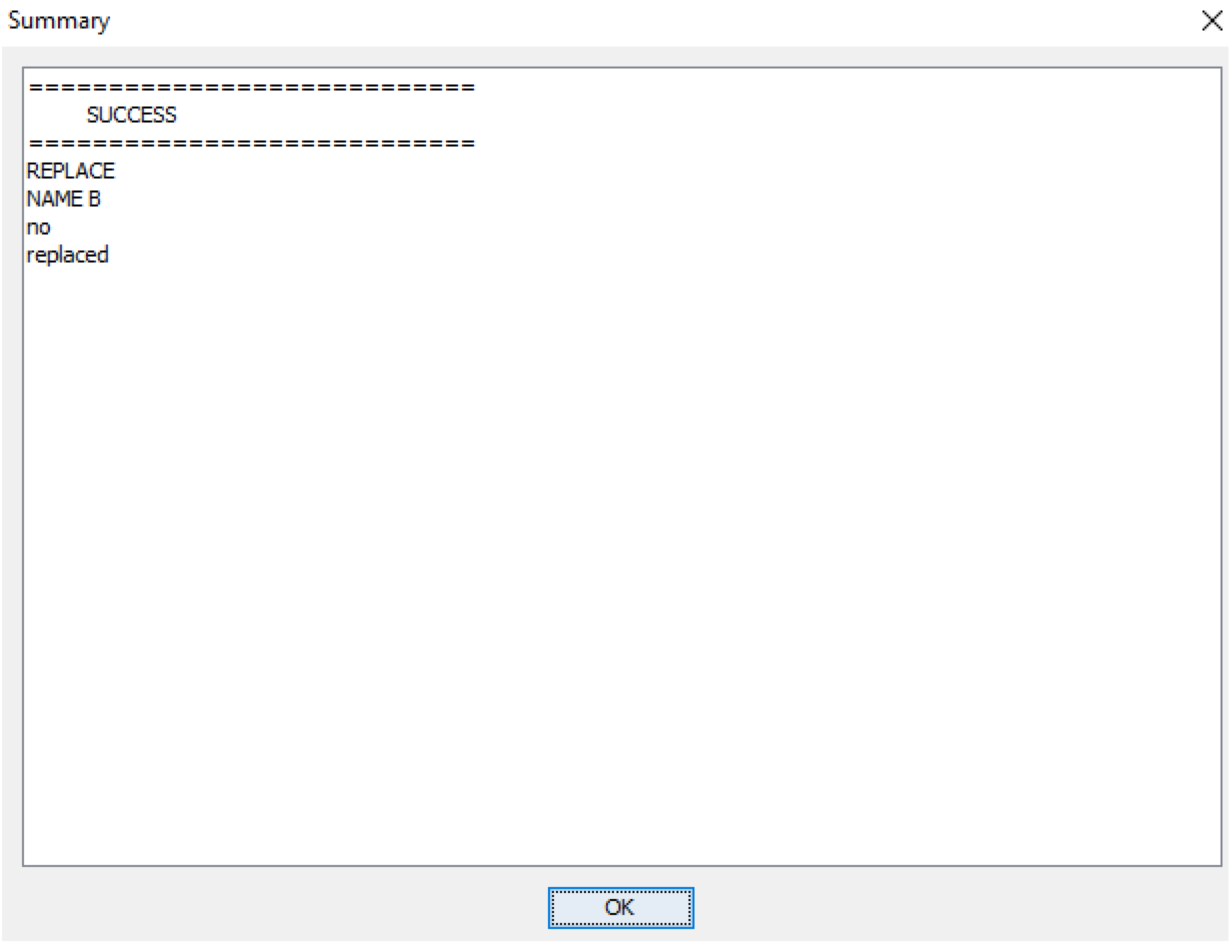
REPLACE NAME B value val
REPLACE_REGEX NAME E [nN].* no
COLUMN_CLONE NAME S Cloned S
COLUMN_SHIFT NAME X 2 DOWN
COLUMN_CREATE NAME X
COLUMN_DELETE NAME X
REPLACE NAME B no replaced

#	B	D	A	S	L	T
0	replaced	no	no	no	no	no
1	replaced	no	no	no	no	no
2		yes	no	yes	no	no
3	replaced	no	no	yes	no	no
4	yes	yes	no		no	no
5	replaced		no	yes	no	no

Figure 17: Preview of the result of running 2 selected data processors.

Applying Data Processors

To apply a data processor, select the data processor and click the run button. The data processor is applied to the data set, and a window appears with a summary of which data processors succeded and which failed, see Figure 18. If any errors appear, use the preview functionallity for further investigation and debugging.



```
============================
     SUCCESS
============================
REPLACE
NAME B
no
replaced
```

Figure 18: Summary of run data processors.

4.8.12 Data Classifier Performance

The data classifier performance analysis tool computes ROC curves for binary classifiers and confusion matrix for binary and multi class classifiers based on values in the data set. The data classifier performance tool is available for the *data matrix* (page 431).

Requirements and Assumptions

Requirement

Classifier performance tool require that probabillities be pre-computed and present in data set - batch propagation feature of the *data frame* (page 391) can be used to pre-compute required beliefs before running classifier performance analysis.

Assumptions about data

It is assumed that the beliefs present in the data are pre-computed by propagating in a network where the actual value of the target class has been held back.

Tips for pre-computing beliefs using the batch propagation feature:

- To hold back evidence on the target node: rename the column name to something different than the target node (e.g. prepend an underscore symbol '_') and then perform batch propagation.

ROC Curve

The ROC curve lets you inspect the performance of a given variable as a binary classifier for the data set.

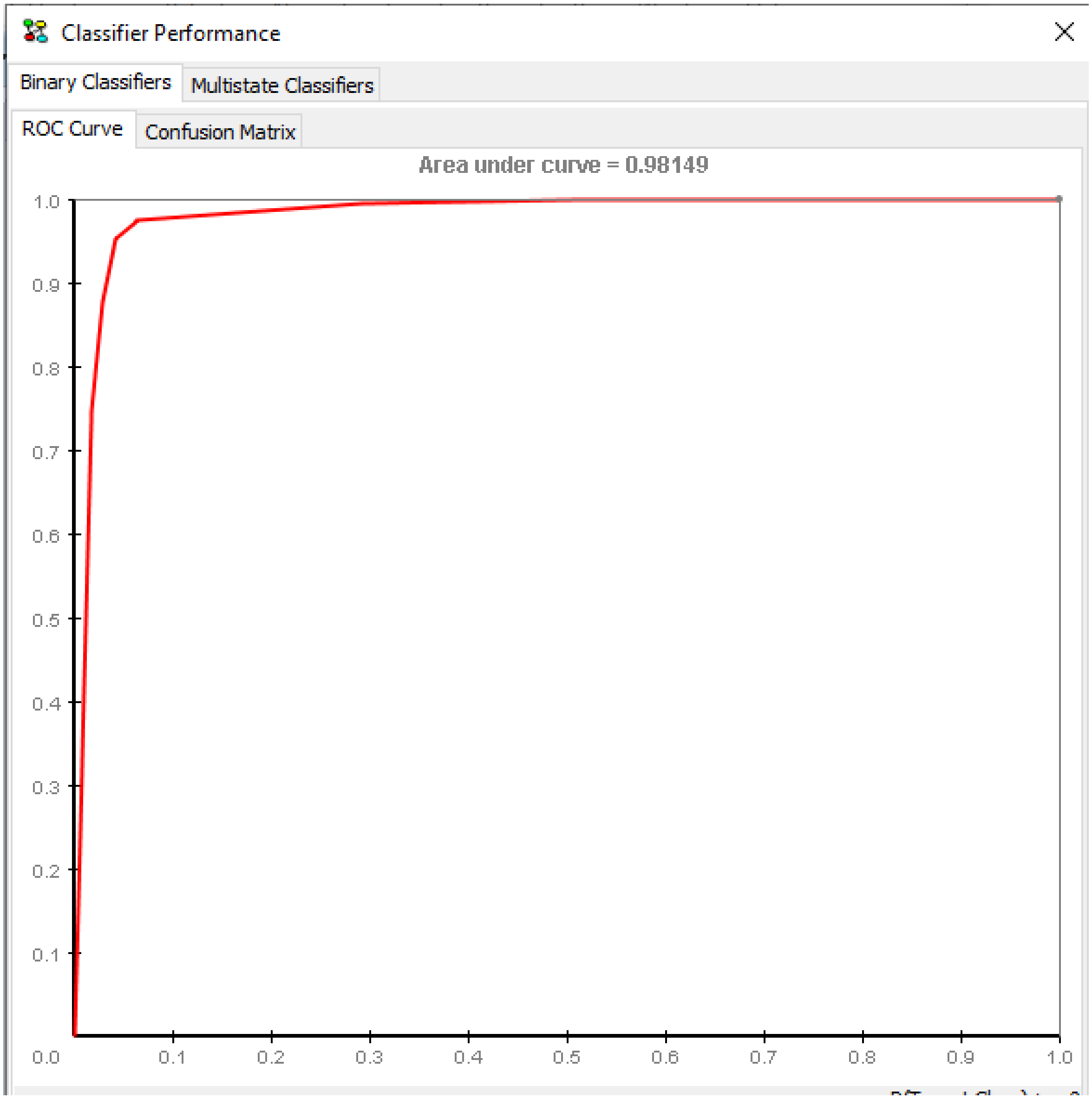


Figure 1: ROC curve

X-axis is the false positive rate, and the Y-axis is the true positive rate.

The ROC curve is based on the performance for predicting specific states.

The area under the curve can be used as a measure for the "goodness" of the network as a classifier for the given variable.

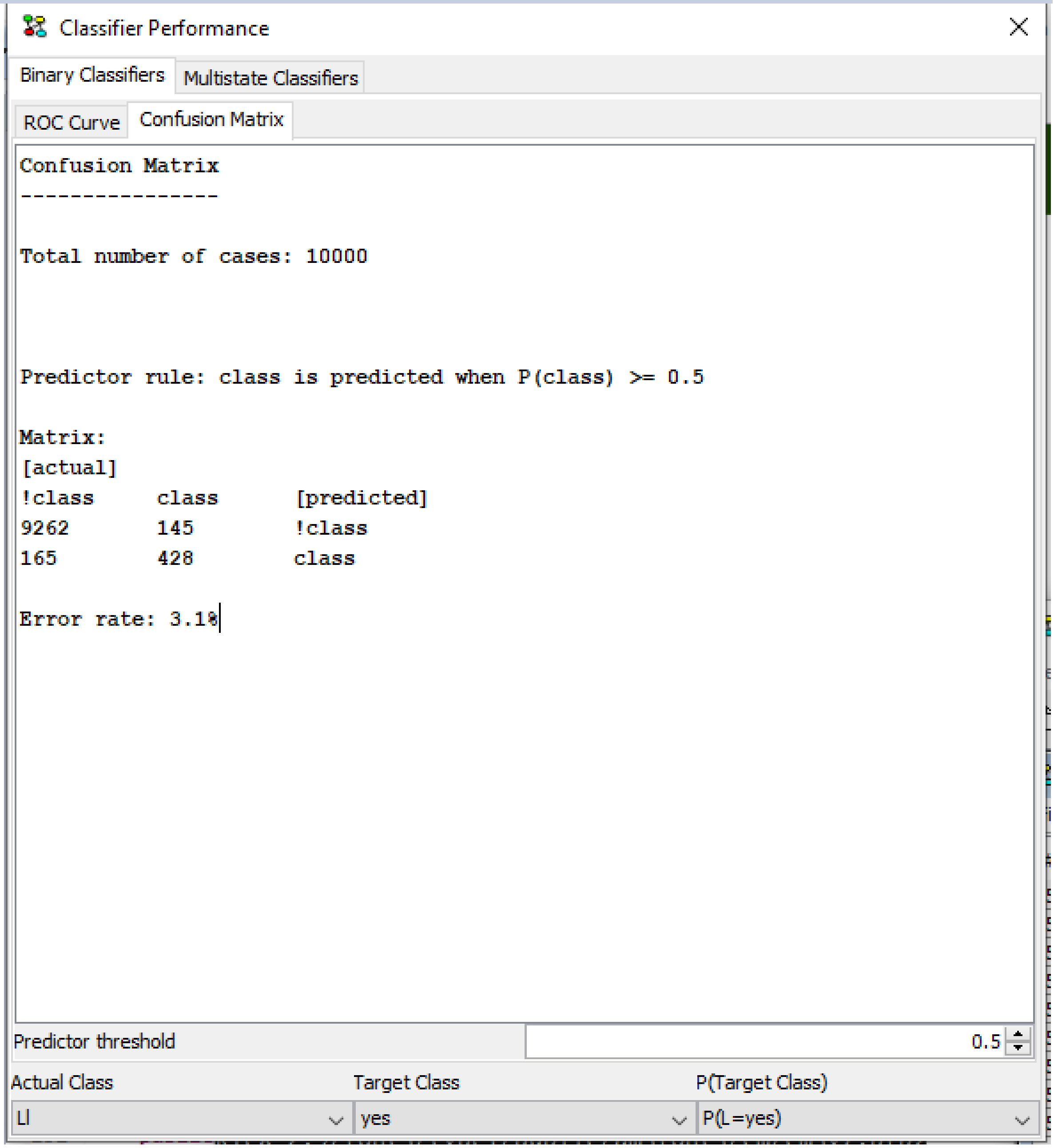
Click a point on the ROC curve to inspect the corresponding predictor threshold.

Confusion Matrix

A confusion matrix can be computed for binary and multi class classifiers.

Binary Classifier Confusion Matrix

A Binary confusion matrix can be computed based on a specific predictor threshold. The confusion matrix shows how well a given variable performs as a predictor of the actual class.



```
Confusion Matrix
----------------

Total number of cases: 10000



Predictor rule: class is predicted when P(class) >= 0.5

Matrix:
[actual]
!class     class      [predicted]
9262       145        !class
165        428        class

Error rate: 3.1%
```

Predictor threshold 0.5

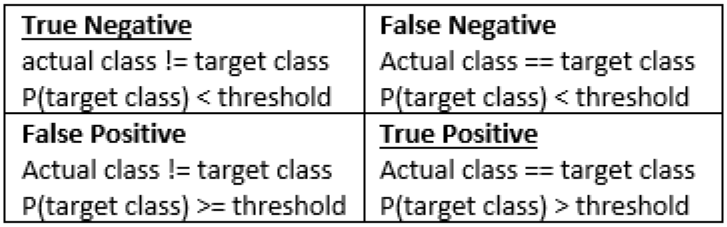
Actual Class: Ll

Target Class: yes

P(Target Class): P(L=yes)

Figure 2: Confusion matrix

The matrix is constructed like this:



True Negative actual class != target class P(target class) < threshold	**False Negative** Actual class == target class P(target class) < threshold
False Positive Actual class != target class P(target class) >= threshold	**True Positive** Actual class == target class P(target class) > threshold

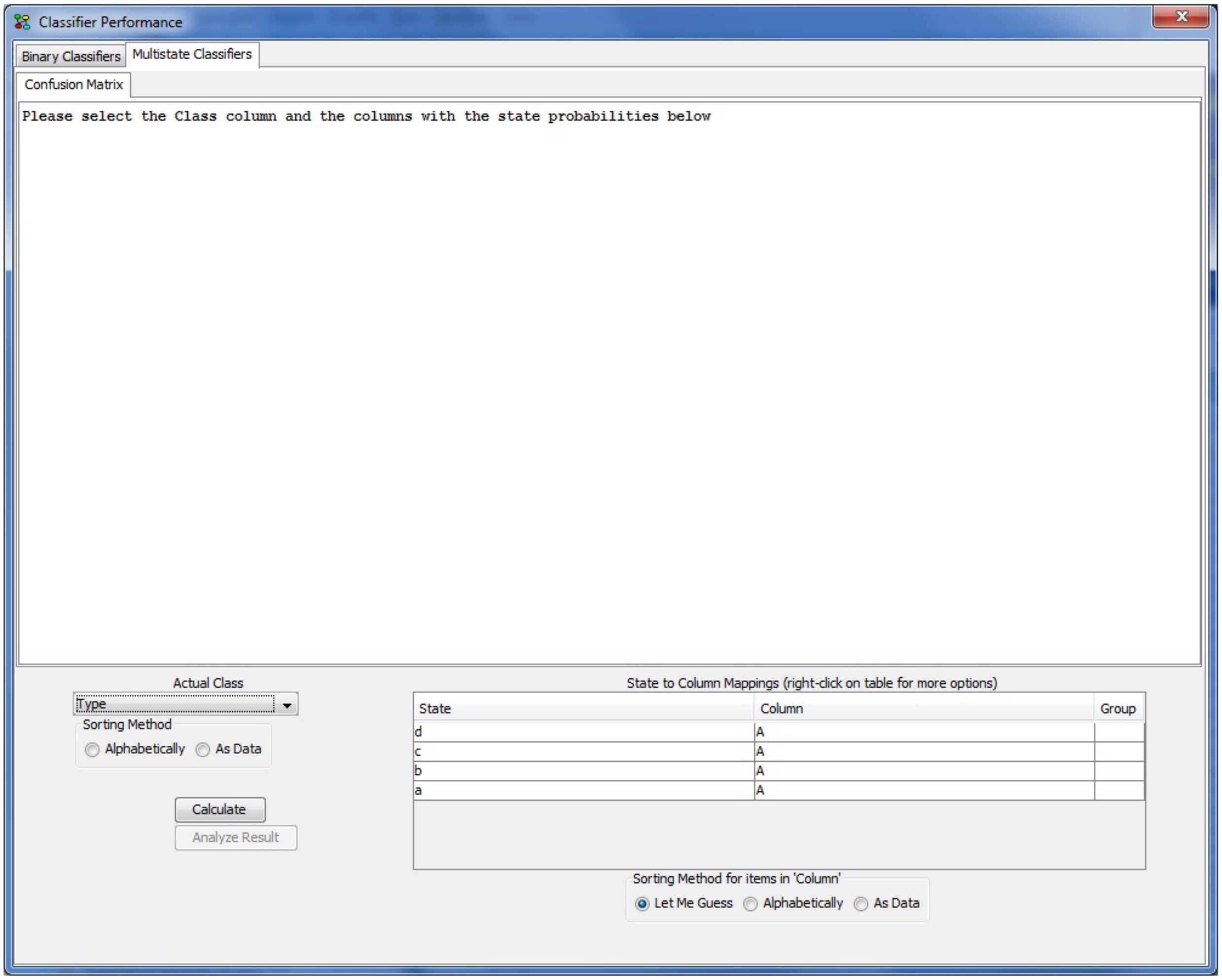
And finally the error rate is reported. It is a measure of the number of false classifications over number of true classifications.

Change the value in the threshold input box to inspect the corresponding confusion matrix.

Multi-State Classifier Confusion Matrix

A Confusion Matrix for Multistate classifiers can be calculated under the Multistate Classifiers tab.

As Actual Class select the name of the column containing the target class labels. Individual cases as well as Sequences can be classified. To classify sequences, make sure that a column has been marked as sequence identifier. When a column is selected under ‘Actual Class’, the table ‘State to Column Mappings’ is polulated with a row for each possible state. Each row int the table is a mapping from a state in the ‘Actual Class’ table to a column of probabilities in the data set (See Figure 3). As indicated in the parentheses a right-click menu is available. Additional mappings can be created if mappings are needed for states not observed in the ‘True Class’ column. Different options are available for sorting alloing convenient selection of the ‘Actual Class’ and State-Column mapping. In Sequence mode (A column has been labeled as sequence identifier) a Sequence Classification method can be configured. There are two sequence classification methods available. The simplest method assigns the entire sequence to the class that obtained the highest probability. The second classification method assigns the sequence S_i to class C if the $P(C|S_i) > t$ for some threshold t. That means that with this setting a sequence can potentially be assigned to multiple classes which will be reflected and summarized in the confusion matrix.



```
Please select the Class column and the columns with the state probabilities below
```

Actual Class: Type

Sorting Method: ○ Alphabetically ○ As Data

State to Column Mappings (right-click on table for more options)

State	Column	Group
d	A	
c	A	
b	A	
a	A	

Sorting Method for items in 'Column': ● Let Me Guess ○ Alphabetically ○ As Data

Figure 3: Multistate Confusion matrix.

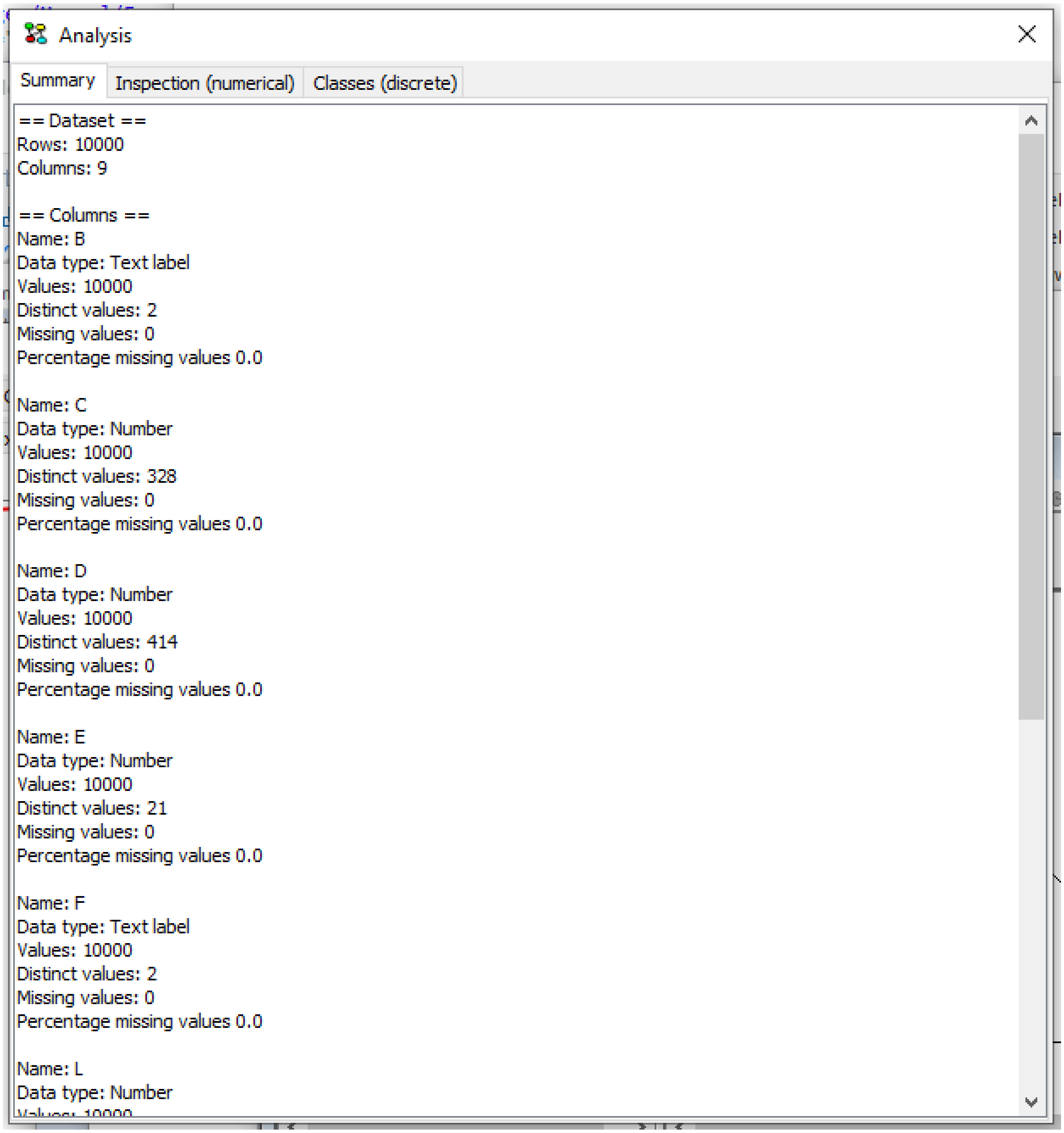
Click ‘Calculate’ to calculate the the Confusion matrix. Once the confusion matrix has been calculated, it is possible to look further at the classification result by clicking ‘Analyze Result’. This will open a new window displaying in details which class each case or sequence was assigned to in the calculation.

4.8.13 Data Analysis

The data analysis tool provides summary of data properties and visual inspection of data points. The data analysis tool is available for the *data matrix* (page 431).

Summary

A summary of data set properties is found in the summary panel, see Figure 1.



```
== Dataset ==
Rows: 10000
Columns: 9

== Columns ==
Name: B
Data type: Text label
Values: 10000
Distinct values: 2
Missing values: 0
Percentage missing values 0.0

Name: C
Data type: Number
Values: 10000
Distinct values: 328
Missing values: 0
Percentage missing values 0.0

Name: D
Data type: Number
Values: 10000
Distinct values: 414
Missing values: 0
Percentage missing values 0.0

Name: E
Data type: Number
Values: 10000
Distinct values: 21
Missing values: 0
Percentage missing values 0.0

Name: F
Data type: Text label
Values: 10000
Distinct values: 2
Missing values: 0
Percentage missing values 0.0

Name: L
Data type: Number
Values: 10000
```

Figure 1: Summary of data set properties

The generated summary reports:

- Number of rows in data set
- Number of columns in data set

As well as a per column summary that contain:

- Column name
- Inferred data type - *this is either Interval, Number or Text label*
- Values count (not counting missing values)
- Distinct values count
- Missing values count
- The percentage of missing values (missing/total)

Inspection

The inspection panel provide visual inspection of numerical data points by plotting all values from selected columns in a single chart. Depending on the data type of the selected columns, the chart may take the form of a set of dots (numerical data X numerical data), lines (numerical data X discrete data) or a grid (discrete data X discrete data).

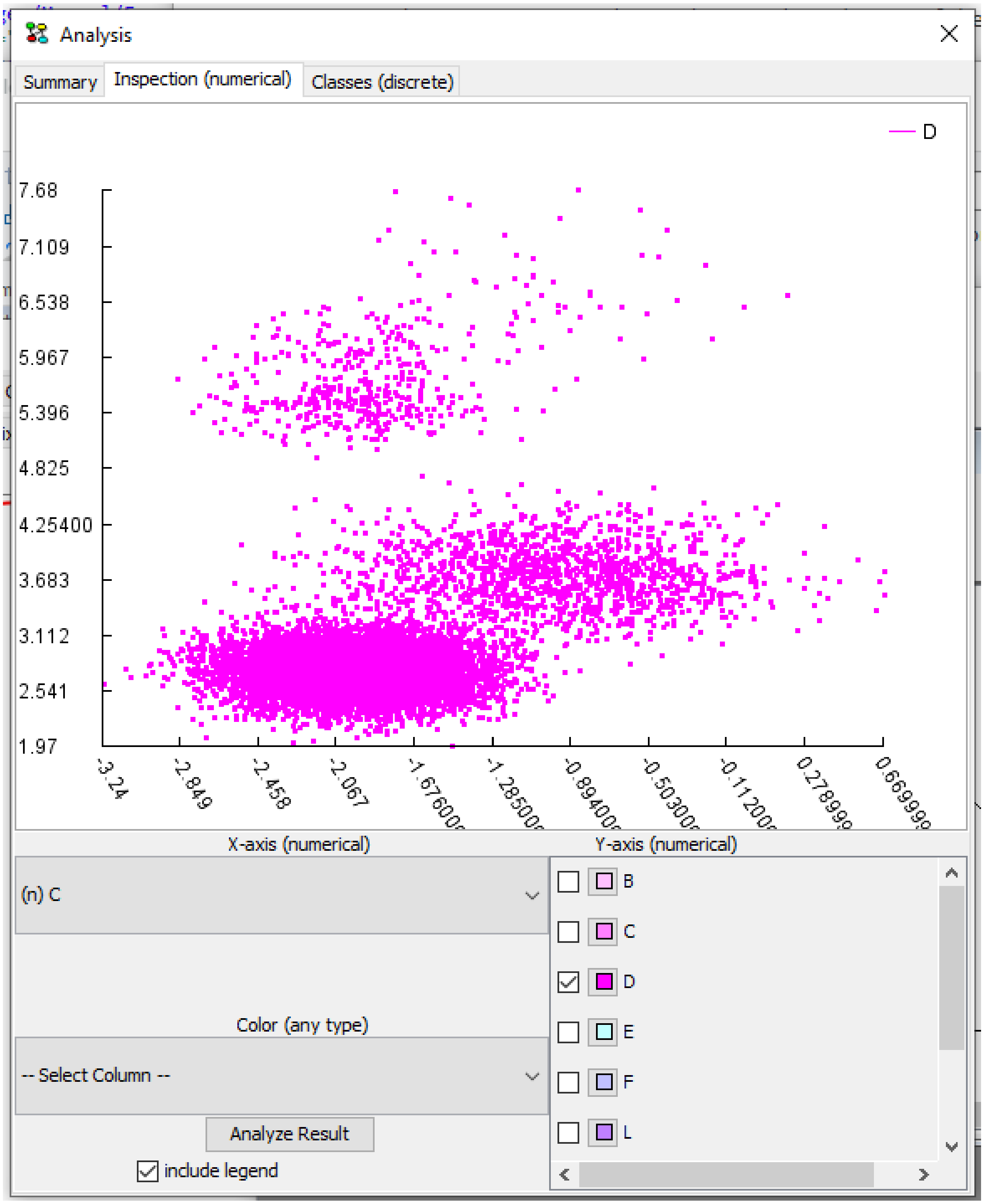
Multiple variables can be selected on the Y-axis by selecting multiple checkboxes in the bottom right panel. The colors of each variable can be customized by clicking on the colored icon next to the variable name.

To plot a set of variables as a sequence, select the special value “Row Id” as the variable on the X-axis. It appears as the first option in the drop-down menu.

The legend in the top-right corner can be removed by de-selecting the ‘include legend’ option at the bottom of the screen.

Note that you can re-size the window to adjust the size of the plot.

The inspection panel in Figure 2 has a plot generated from numerical data.



X-axis (numerical): (n) C

Y-axis (numerical):
- □ B
- □ C
- ☑ D
- □ E
- □ F
- □ L

Color (any type): -- Select Column --

☑ include legend

Figure 2: Inspecting plot of numerical data points - missing data is plottet outside chart

A total of three columns can be used to generate the chart: X-axis, Y-axis and coloring. Columns can be selected from the three drop-down lists. The drop-down lists contain column names, prefixed according to data type with either (n) for numerical data, (i) for interval, or (t) for text labels.

Missing data go outside the chart following these rules:

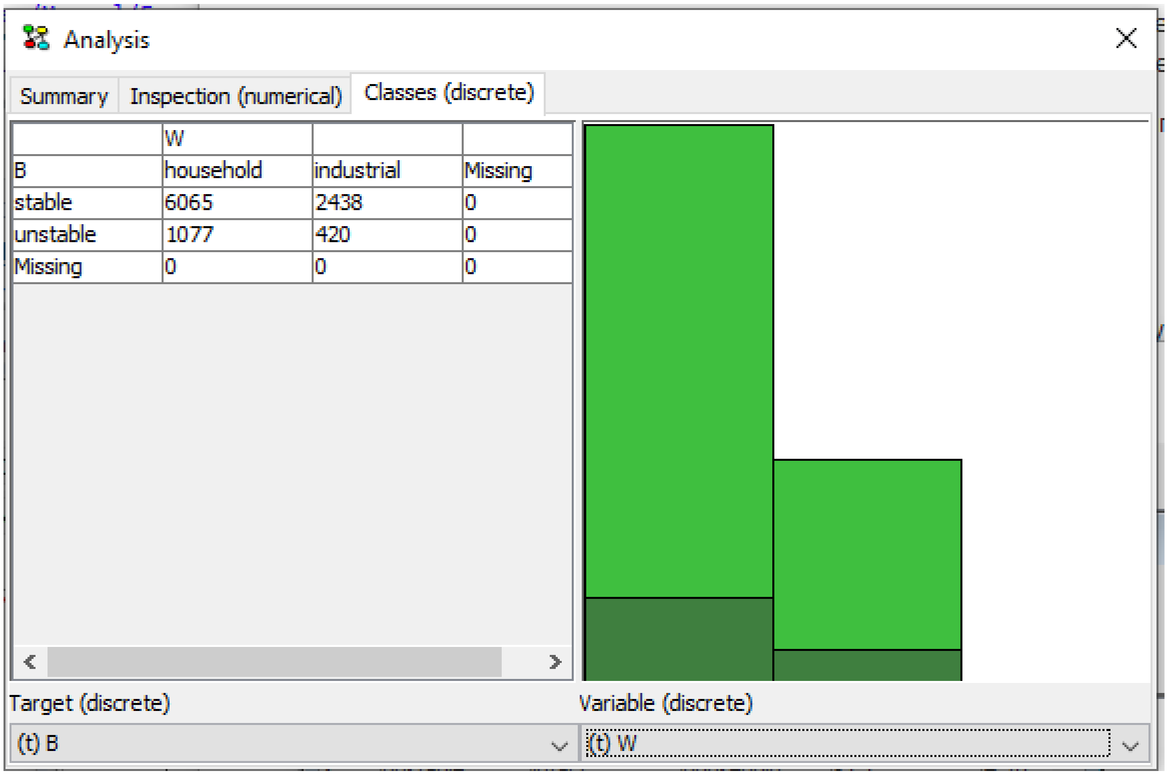
- Missing on both X and Y go in the upper right corner
- Missing on X and observed on Y go outside chart to the right
- Observed on X and missing on Y go outside chart in the top

If a column has been chosen for coloring the plot, values are mapped to colors in the range from red to green, blue indicates missing value.

Classes

The classes panel provides inspection of discrete variables by classifying a variable using a selected target class. When selecting a variable and target class, a table with counts and a column diagram is produced. When a column in the diagram is clicked, the entry in the table that corresponds to the clicked column part is highlighted. Hovering the mouse cursor over the diagram reports the corresponding configuration as well.

The classes panel in Figure 3 contains a diagram for classification using two discrete variables with three states each.



	W		
B	household	industrial	Missing
stable	6065	2438	0
unstable	1077	420	0
Missing	0	0	0

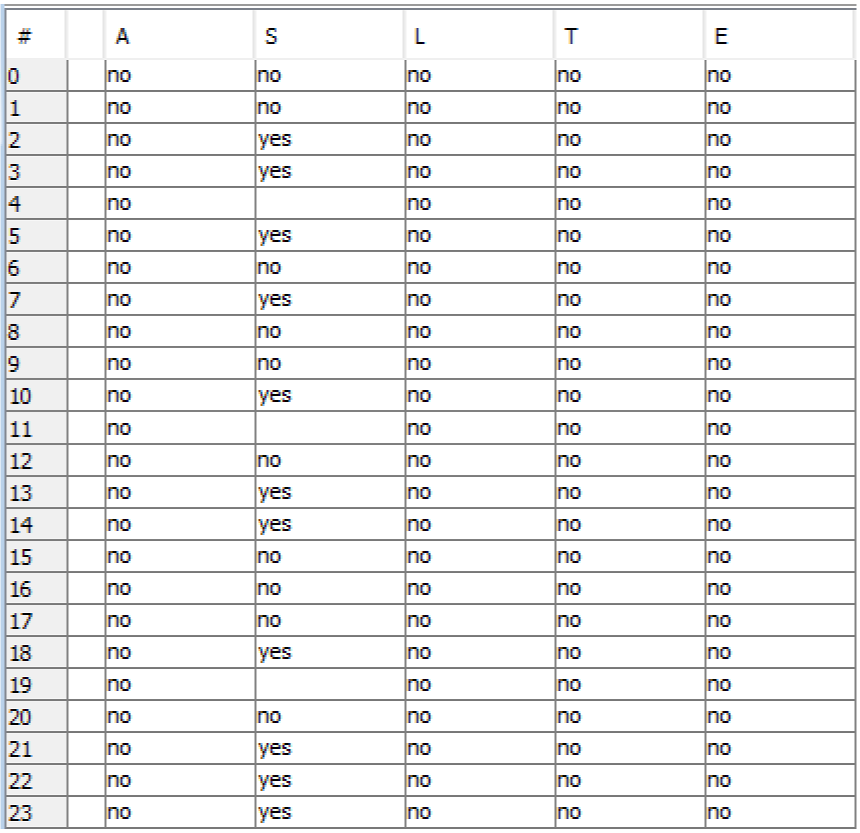
Target (discrete): (t) B

Variable (discrete): (t) W

Figure 3: Inspecting classification using discrete variables

4.8.14 Data Matrix

The data matrix is HUGINs tool for inspecting and manipulating data sets. Data is displayed using a grid like interface very similar to what most users know from Excel, see Figure 1. The data matrix is available in places dealing with data files, including the *learning-* (page 267) and *EM wizards* (page 281), the *data frame functionallity* (page 391) and other places dealing with data.



#		A	S	L	T	E
0		no	no	no	no	no
1		no	no	no	no	no
2		no	yes	no	no	no
3		no	yes	no	no	no
4		no		no	no	no
5		no	yes	no	no	no
6		no	no	no	no	no
7		no	yes	no	no	no
8		no	no	no	no	no
9		no	no	no	no	no
10		no	yes	no	no	no
11		no		no	no	no
12		no	no	no	no	no
13		no	yes	no	no	no
14		no	yes	no	no	no
15		no	no	no	no	no
16		no	no	no	no	no
17		no	no	no	no	no
18		no	yes	no	no	no
19		no		no	no	no
20		no	no	no	no	no
21		no	yes	no	no	no
22		no	yes	no	no	no
23		no	yes	no	no	no

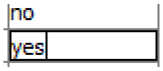
Figure 1: A typical data matrix view

The leftmost column is the row header which display row numbers, first row is 0 as rows are zero-indexed. The top header display column names, when using data with a HUGIN model the columns represent nodes.

Basic Editing

The data set can be inspected using the data matrix. Although HUGIN is not EXCEL, some basic editing of data items can be performed:

- Use Mouse pointer or cursor-keys to navigate about
- Double-click a cell to edit the data item (see Figure 2)



no
yes

Figure 2: Editing a cell

- Double-click column header to edit column name (see Figure 3)

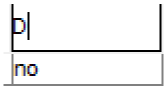
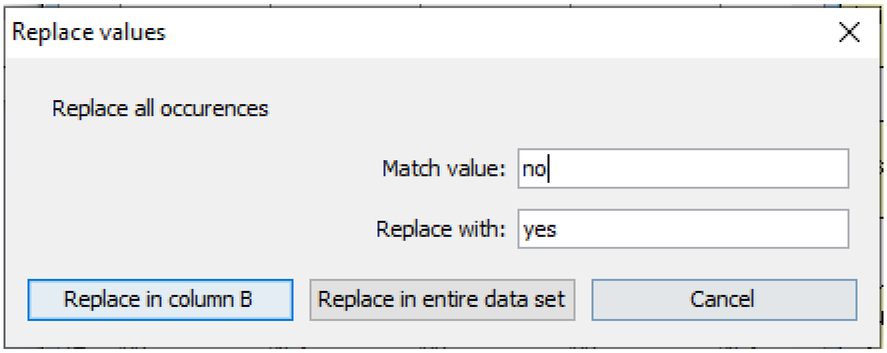


Figure 3: Renaming column

- Search and replace values in single column/all columns by right-clicking and selecting Replace values (see Figure 4)



Replace all occurences

Match value: no

Replace with: yes

Figure 4: Basic search and replace

- Re-order columns by clicking a column header and dragging the column left or right

Rows and columns can be edited as well:

- Create/delete row by right-clicking row header and selecting *insert- or delete row*
- Create/delete column by right-clicking column header and selecting *insert- or delete column*

Figures 5 and 6 display the options available by right-clicking row header and column.

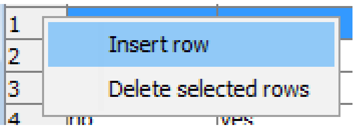


Figure 5: Right-click the row header - options

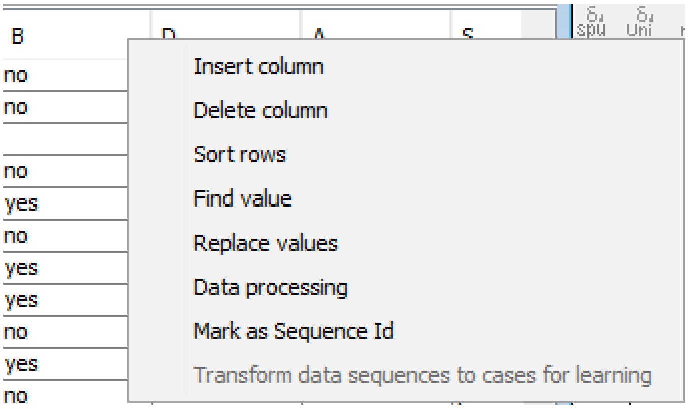


Figure 6: Right-click column - options

Advanced Functionallity

Columns containing numerical data, **exclusively**, can be discretized into a set of intervals by launching the discretizer (read more about the *discretization* (page 434) features):

- Right-click a column containing numerical data and select Discretize values

More elaborate modifications are performed using batch preprocessing (read more about the *preprocessing (Data Processing)* (page 408) features):

- Right-click and select the *Data Processing* option

To inspect properties of the data set use the data set analysis functionallity (read more about the *data analysis* (page 426) features):

- Right-click and select the *Analysis* option

To perform analysis of classifier performance (read more about the *classifier performance* (page 422) analysis):

- Right-click and select the *Classifier Performance* option

4.8.15 Data Frame With Database Connection

HUGIN can extract data from an Apache Derby database[13] into the data frame. This functionality is available from the 'Data' menu item in the HUGIN main window. Apache Derby is a database project purely written in Java allowing for flexible database integration in Java programs. More information can be retrieved on the Apache Derby project's website[14].

A screenshot of the "Database Connection" panel can be seen in Figure 1.

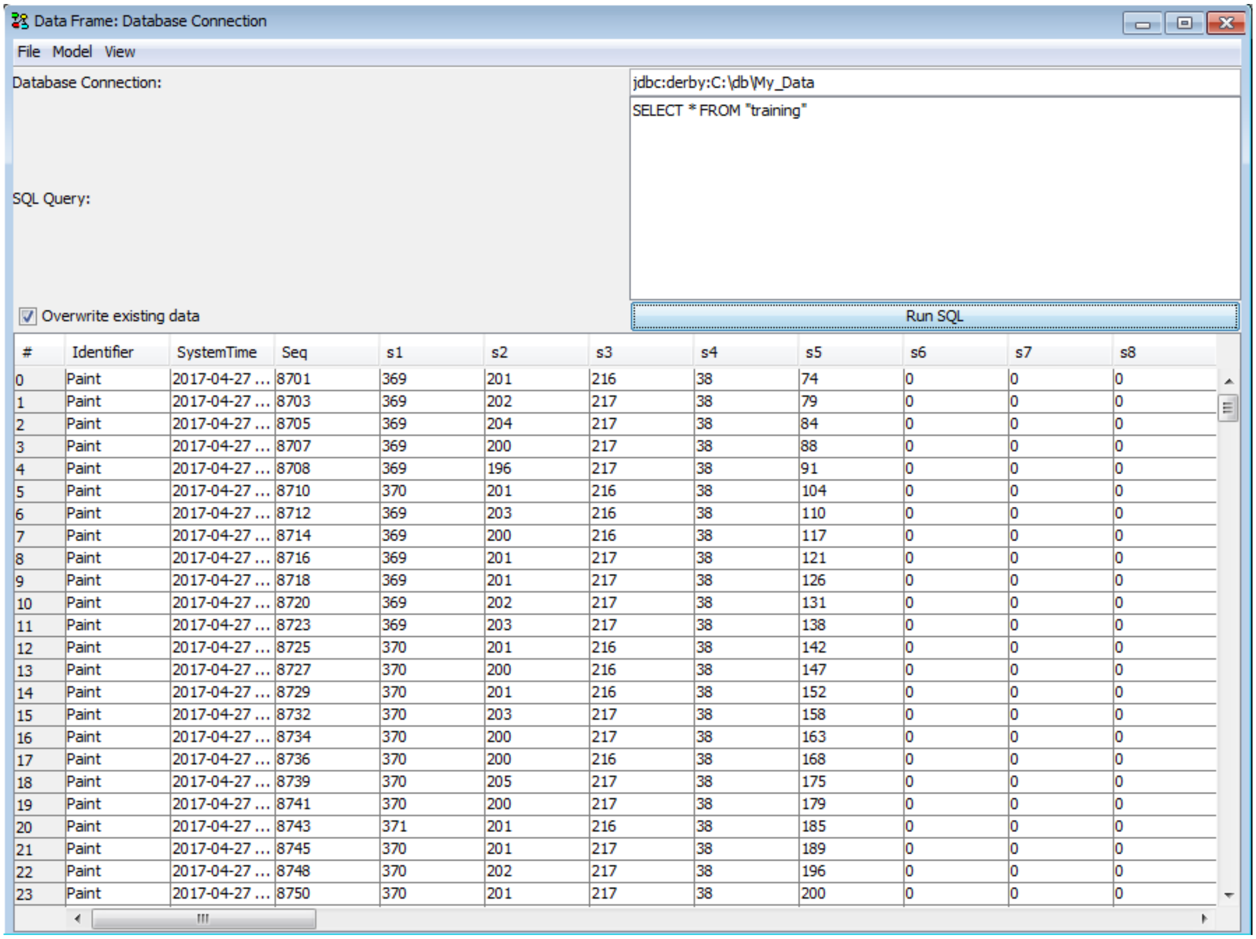
To connect to a Derby database the user needs to specify the following:

- **"Database Connection":** Enter the database connection details in the field labeled . What to type here depends on the configuration of the database. It can be located either on the local file system or it may be configured as a server on a network location. Turn to the Derby Reference Manual for more details.
- **"SQL Query":** The query to be executed. The Derby Reference manual, found on Apache Derby project's website[15] documents the Derby SQL language.
- **"Overwrite existing data":** Indicate whether the extracted data should overwrite the existing data i the below data matrix. If the box is selected, the existing data will be deleted and replaced by the new data. Otherwise the data will be appended.

The current version of the Derby Database driver included in HUGIN is Derby 10.14.1.0 Release, so you should be looking for the 10.14 Derby Reference Manuals.

Once extracted, the data can be inspected, manipulated just as if the data had been read from a file in an ordinary *Data Matrix Frame* (page 391). Changes in the data will not take effect in the database but the extracted (and perhaps edited) data can be exported to a file using the "Save As" accessible through the "File" menu item.

[13] https://db.apache.org/derby/
[14] https://db.apache.org/derby/
[15] https://db.apache.org/derby/



Database Connection: jdbc:derby:C:\db\My_Data

SQL Query: SELECT * FROM "training"

☑ Overwrite existing data

#	Identifier	SystemTime	Seq	s1	s2	s3	s4	s5	s6	s7	s8
0	Paint	2017-04-27 ...	8701	369	201	216	38	74	0	0	0
1	Paint	2017-04-27 ...	8703	369	202	217	38	79	0	0	0
2	Paint	2017-04-27 ...	8705	369	204	217	38	84	0	0	0
3	Paint	2017-04-27 ...	8707	369	200	217	38	88	0	0	0
4	Paint	2017-04-27 ...	8708	369	196	217	38	91	0	0	0
5	Paint	2017-04-27 ...	8710	370	201	216	38	104	0	0	0
6	Paint	2017-04-27 ...	8712	369	203	216	38	110	0	0	0
7	Paint	2017-04-27 ...	8714	369	200	216	38	117	0	0	0
8	Paint	2017-04-27 ...	8716	369	201	217	38	121	0	0	0
9	Paint	2017-04-27 ...	8718	369	201	217	38	126	0	0	0
10	Paint	2017-04-27 ...	8720	369	202	217	38	131	0	0	0
11	Paint	2017-04-27 ...	8723	369	203	217	38	138	0	0	0
12	Paint	2017-04-27 ...	8725	370	201	216	38	142	0	0	0
13	Paint	2017-04-27 ...	8727	370	200	216	38	147	0	0	0
14	Paint	2017-04-27 ...	8729	370	201	216	38	152	0	0	0
15	Paint	2017-04-27 ...	8732	370	203	217	38	158	0	0	0
16	Paint	2017-04-27 ...	8734	370	200	217	38	163	0	0	0
17	Paint	2017-04-27 ...	8736	370	200	216	38	168	0	0	0
18	Paint	2017-04-27 ...	8739	370	205	217	38	175	0	0	0
19	Paint	2017-04-27 ...	8741	370	200	217	38	179	0	0	0
20	Paint	2017-04-27 ...	8743	371	201	216	38	185	0	0	0
21	Paint	2017-04-27 ...	8745	370	201	217	38	189	0	0	0
22	Paint	2017-04-27 ...	8748	370	202	217	38	196	0	0	0
23	Paint	2017-04-27 ...	8750	370	201	217	38	200	0	0	0

Figure 1: Derby Database Connection Window.

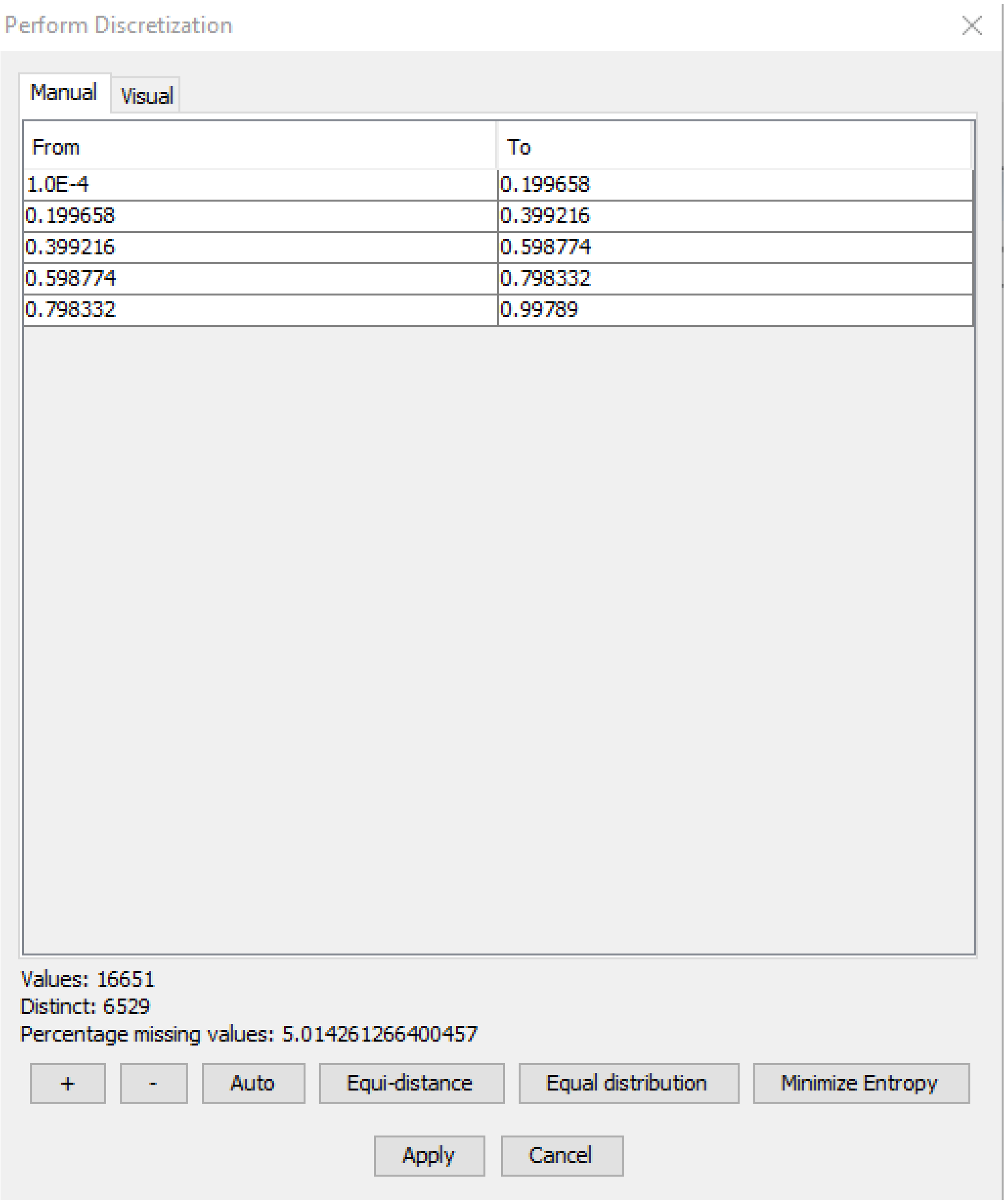
4.8.16 Discretization

Discretization is the task of dividing a set of numerical values into a number of intervals. This task can be performed using the discretization tool. The discretization tool is available for *data matrix* (page 431) columns containing numerical data.

When specifying intervals, the lower bound of each interval is inclusive, and the upper bound exclusive - except for the last interval where both lower and upper bounds are inclusive. E.g. If we specify the intervals 0-1, 1-2, 2-3, then the first interval 0-1 would contain all values >=0 and <1, the second interval would contain values >=1 and <2, the last interval would contain values >=2 and <=3.

Intervals must be contiguous, non-overlapping and bound all values in the data to be valid!

The discretization tool can be seen in Figure 1.



From	To
1.0E-4	0.199658
0.199658	0.399216
0.399216	0.598774
0.598774	0.798332
0.798332	0.99789

Values: 16651
Distinct: 6529
Percentage missing values: 5.014261266400457

Figure 1: Discretization tool - manualy enter intervals

The top part of the discretization tool has two tabs, Manual and Visual, each providing alternative ways for specifying intervals. From the manual tab, intervals can be entered by hand, and from the visual tab intervals can be defined using a slider.

The bottom part of the tool summarizes a few intersting details about the data, the number of values (not counting missing values), the number of distinct values, and the percentage of missing values.

Below the data summary is a row of buttons for adding and removing intervals, as well as for performing a number of automatic discretization functions.

Function Buttons

Button +

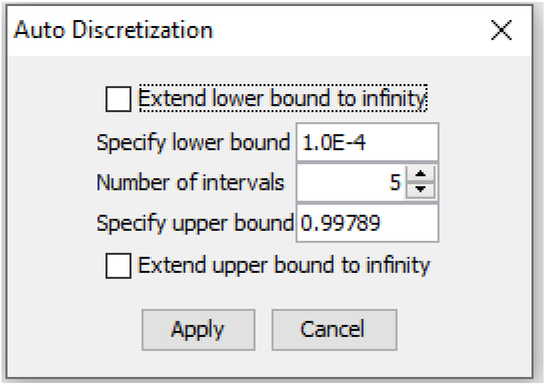
Add interval.

Button -

Remove interval.

Auto

Clicking the Auto button brings forward the auto discretization dialog, see Figure 2. Auto discretization divides the values into a specified number of equally distanced intervals. From the dialog, the number of intervals can be specified, as well as lower and upper bounds and whether to extend bounds to infinity.



☐ Extend lower bound to infinity
Specify lower bound 1.0E-4
Number of intervals 5
Specify upper bound 0.99789
☐ Extend upper bound to infinity

Figure 2: Auto discretization dialog - quickly generate a number of equally distanced intervals

Equi-distance

This performs the same kind of discretization as provided by the *Auto discretization dialog*, just using the current number of intervals.

Equal distribution

This function tries to fit the bounds of the intervals such that each interval contains approximately the same number of values. Depending on how well the data values are scattered, the number of intervals may be pruned in order to make the resulting intervals equally distributed.

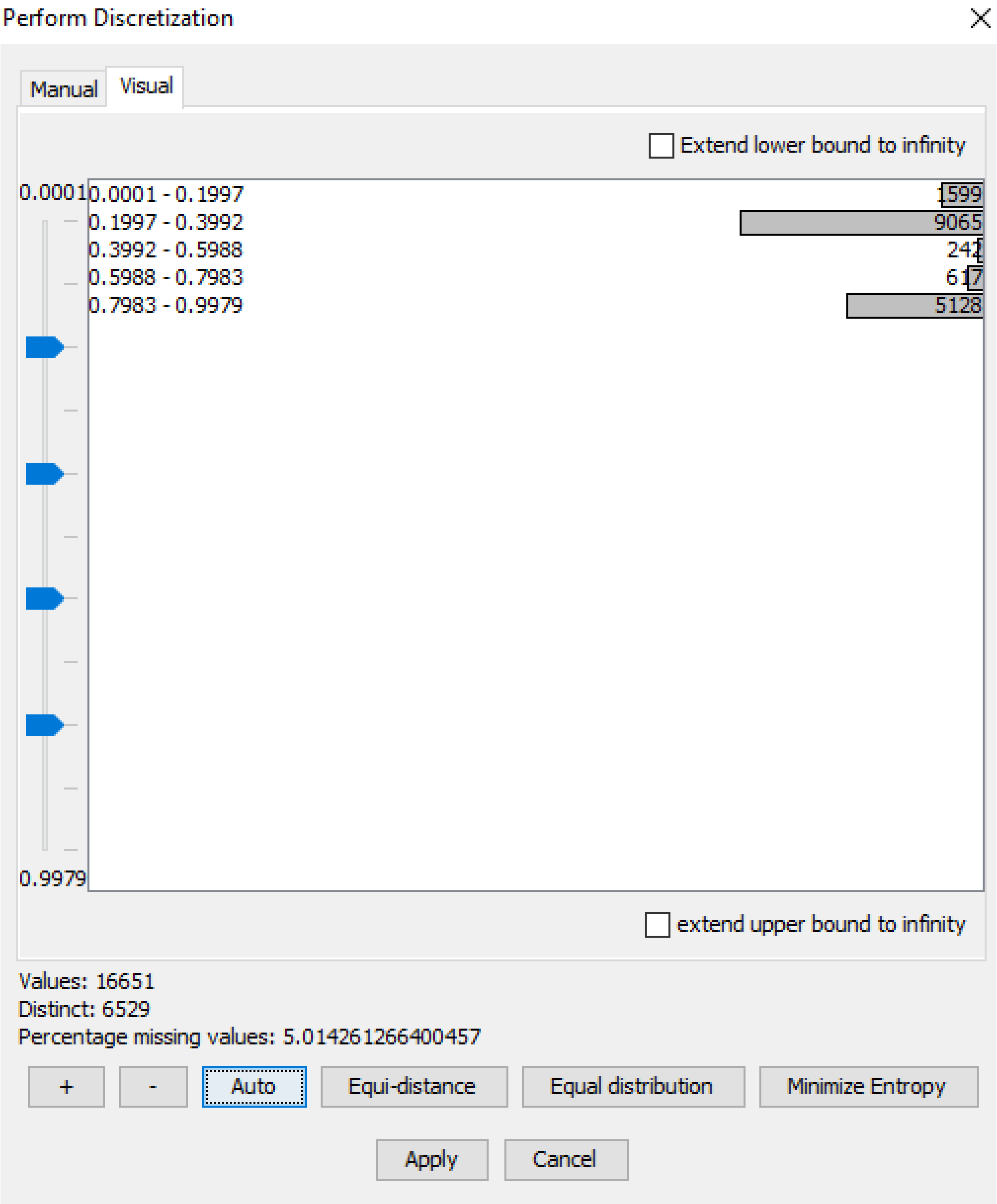
Manual

The manual tab contains the table which was seen in figure 1. Each row is an interval, and the columns are the lower and upper bounds for each interval. Intervals are added and removed using the + and - buttons. Upper and lower bounds for an interval is edited by double-clicking the appropriate cell.

Positive and negative infinity are specified using the strings *'inf'* (for positive infinity), and *'-inf'* (for negative infinity). Negative infinity is only valid for the lower bound of the first interval, positive infinity is only valid for the closing upper bound of the last interval.

Visual

The visual tab contains a list of intervals, and a set of sliders, see Figure 3.



☐ Extend lower bound to infinity

0.0001 - 0.1997	1599
0.1997 - 0.3992	9065
0.3992 - 0.5988	242
0.5988 - 0.7983	617
0.7983 - 0.9979	5128

☐ extend upper bound to infinity

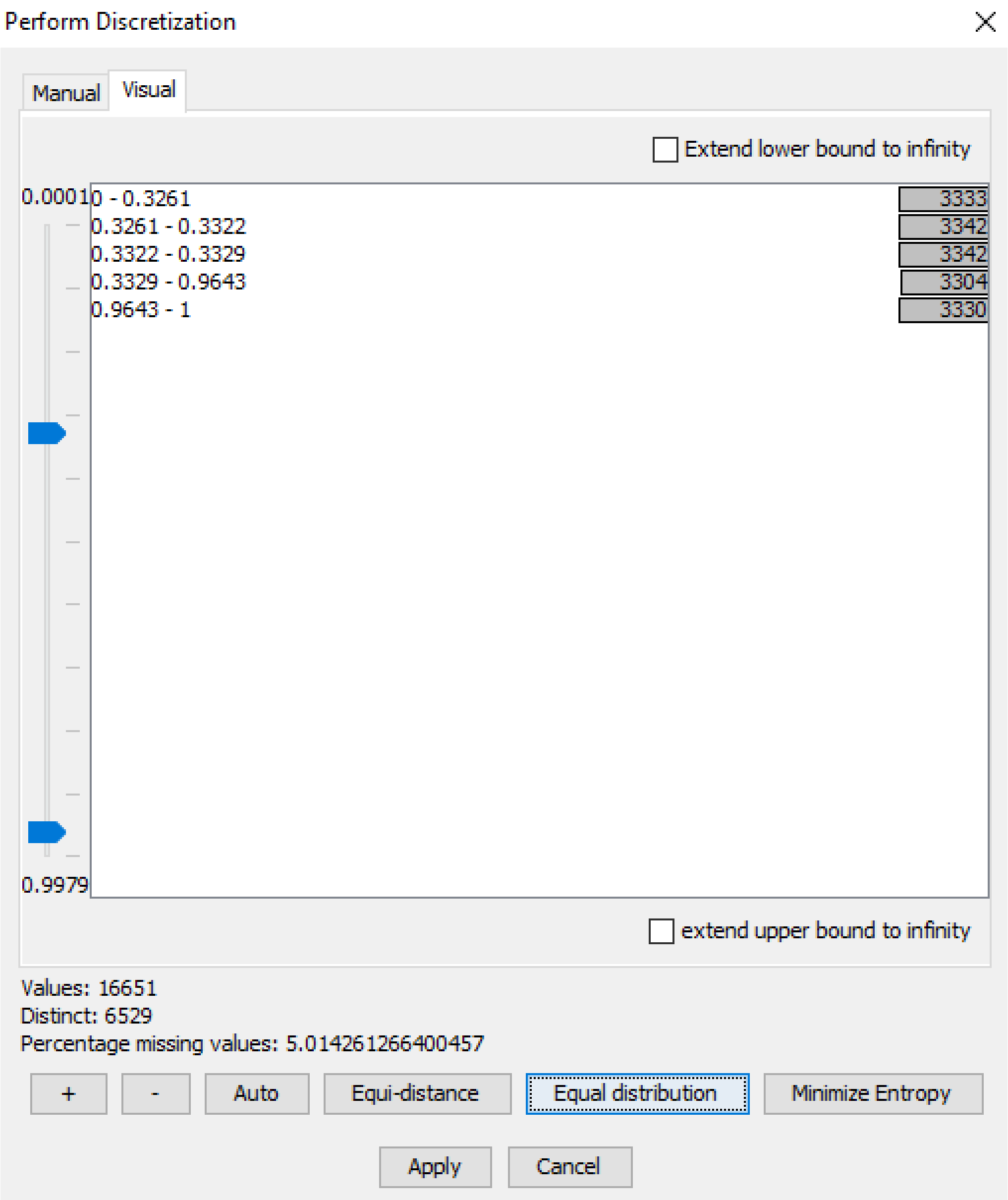
Values: 16651
Distinct: 6529
Percentage missing values: 5.014261266400457

Figure 3: Visual discretization - bars show value count for each interval

The ruler on the left has a slider for each interval. Interval bounds are modified by adjusting the sliders.

The interval list is annotated with bars and value counts. The bars show the proportional number of data values contained in a specific interval. This provided a quick visual overview of how data values are distributed between intervals.

The bars from Figure 3 show an unequal distribution of values between intervals, Figure 4 shows a much better distribution.



□ Extend lower bound to infinity

0 - 0.3261	3333
0.3261 - 0.3322	3342
0.3322 - 0.3329	3342
0.3329 - 0.9643	3304
0.9643 - 1	3330

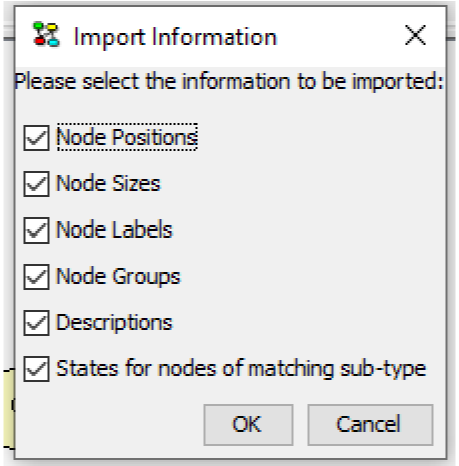
□ extend upper bound to infinity

Values: 16651
Distinct: 6529
Percentage missing values: 5.014261266400457

Figure 4: Bars show the values are equally distributed between intervals

4.8.17 Import Information

Various kinds of information can be inherited from nodes of one network to nodes of another network via the Import Information feature, which is available via the *Network Menu* (page 185) of the *Main Window* (page 186). When this feature is activated and a file has been chosen, a dialog box appears (see Figure 1), which offers the user a possibility to select the kinds of information that should be imported. By default, all kinds of information are selected.



Import Information

Please select the information to be imported:

☑ Node Positions

☑ Node Sizes

☑ Node Labels

☑ Node Groups

☑ Descriptions

☑ States for nodes of matching sub-type

Figure 1: The Import Information dialog box.

This feature is very useful when using the *Learning Wizard* (page 267), allowing one to save (and later reload) structural constraints (i.e., structural domain knowledge). Please note that the Structural Constraints check box is available only when using the Import Information feature from the Learning Wizard.

4.8.18 Import Network Description from Tables

The Import Network Description From Tables wizard can import network descriptions on a primitive table based format.

As most computer software is able to export data to a primitive table based format, this wizard may be the only option for importing network information from, or integrating HUGIN with a thirdparty program unaware of the HUGIN Net language.

A primitive table consists of rows and columns. Individual columns are separated by symbols <SPACE>, <TAB> or comma (‘,’). Rows are separated by either <NEW-LINE> or <CARRIAGE-RETURN><NEW-LINE>.

Importing a network description is performed in three steps:

1. Parse *variable definitions*
2. Parse *link definitions*
3. Parse *table descriptions*

Each kind of description is a file containing a primitive table with only a few formatting conventions.

Variable definitions

The variable definitions table consists of three columns:

```
<NAME> <TYPE> <STATE>
```

The first column <NAME> indicates node name. A node name must begin with a letter and only contain characters [a-z][A-Z][0-9].

The second column <TYPE> indicates node type. Valid values are:

L (labelled node)
B (boolean node)
N (numbered node)
I (interval node)
C (continuous node)
F (function node).

The third column <STATE> specify information about node states and is required for labelled, numbered and interval nodes. The specific number of states for a node is determined by the number of lines pertaining to specifying the particular node. States are created in the order they appear in the file. Values have different meaning depending on node type:

Labelled nodes: Value specify state label. The label must be enclosed in quotation marks (") if it contains any whitespace characters or characters outside the range [a-z][A-Z][0-9]. To specify *n* states, the node must appear in n lines in the file.

Numbered nodes: Value specify numeric state value. Value must be a number which can be specified using characters [0-9] and dot ('.'). To specify *n* states, the node must appear in *n* lines in the file.

Interval nodes: Value specify state interval endpoints. Value must be a number which can be specified using characters [0-9] and dot ('.'). Positive and negative infinity is specified by the string 'infinity' and *'-infinity'*. To specify *n* states, the node must appear in *n + 1* lines in the file. First line indicate lower bound for first interval, following lines indicate upper bound for previous interval and lower bound for current interval. The last line *n + 1* indicate upper bound for state *n*.

Link definitions

The link definitions table consists of a line for each node specifying the parents, consisting of a variable number of columns:

```
<NODE> ( <PARENT-NODE>)*
```

The value in the first column <NODE> indicates a node and values in each of the following <PARENT-NODE> columns specify the parents of this node. Thus the network structure can effectively be described as a number of lines in the link definitions table.

Table descriptions

Node table descriptions are specified in a number of separate files, one per node. Each file contains a primitive table describing the CPT for a particular node. Table descriptions distinguish between tables with model nodes and tables without model nodes.

Tables without model nodes:

first line	<NODE>	
second line	<PARENT-LIST>	<NODE-STATE-LIST>
following lines	<PARENT-CONFIGURATION>	<PROBABILITY-LIST>

The <*PARENT-LIST*> is a list of parent node names:

```
<PARENT-LIST> := (<PARENT-NODE> )*
```

The <*NODE-STATE-LIST*> is a list of states for the node specified in <*NODE*>. A state can be either a state number or a state label:

```
<NODE-STATE-LIST> := (#<STATE-NUMBER> |<STATE-LABEL> )+
```

When specifying tables for continuous nodes the <*NODE-STATE-LIST*> instead denotes alpha and variance:

```
<NODE-STATE-LIST> := alpha variance
```

The <*PARENT-CONFIGURATION*> specify a unique configuration of the parent nodes. The list identifies, in parent node order, a state for each parent node specified in <*PARENT-LIST*>. A state can be either a state number or a state label:

```
<PARENT-CONFIGURATION> := (#<PARENT-STATE-NUMBER> |<PARENT-STATE-LABEL> )*
```

The <*PROBABILITY-LIST*> specifies the probabillity for the particular configuration in <*PARENT-CONFIGURATION*> for each state in <*NODE-STATE-LIST*>:

```
<PROBABILITY-LIST> := (<NUMBER> )+
```

In the case of continuous nodes, the <*PROBABILITY-LIST*> specifies the values for *mean* and *variance*:

```
<PROBABILITY-LIST> := <NUMBER-MEAN> <NUMBER-VARIANCE>
```

Tables for function nodes without model nodes

Specifying a table for a function node is even simpler, as only an expression must be supplied in the form of a quoted string. The table thus consists of a single line with two columns:

single line	<NODE>	<EXPRESSION-AS-STRING

Tables with model variables Specifying a table for a node with model variables is almost the same as without model variables. The difference is the second line, which only contains a <*PARENT-LIST*> and the following lines which instead of a <*PROBABILITY-LIST*> contains the expression for the node based on the given parent configuration:

first line	<NODE>	
second line	<PARENT-LIST>	
following lines	<PARENT-CONFIGURATION>	<EXPRESSION-AS-STRING>

Example

This example is based on the `Chest Clinic` example and demonstrates what the Chest Clinic network would look like on primitive tables form.

Variable definitions

File: asia.nodes
A L yes
A L no
S L yes
S L no
T L yes
T L no
L L yes
L L no
B L yes
B L no
E L yes
E L no
X L yes
X L no
D L yes
D L no

Link definitions

File: asia.parents
T A
L S
B S
E L T
D B E
X E

Table descriptions

File: A.table
A
yes no

```
0.01 0.99
```

File: B.table

```
B
S yes no
yes 0.6 0.4
no 0.3 0.7
```

File: D.table

```
D
B E yes no
yes yes 0.9 0.1
yes no 0.8 0.2
no no 0.1 0.9
no yes 0.7 0.3
```

File: E.table

```
E
T L yes no
yes yes 1 0
yes no 1 0
no no 0 1
no yes 1 0
```

File: L.table

```
L
S yes no
yes 0.1 0.9
no 0.01 0.99
```

File: S.table

```
S
yes no
0.5 0.5
```

File: T.table

```
T
A yes no
yes 0.05 0.95
no 0.01 0.99
```

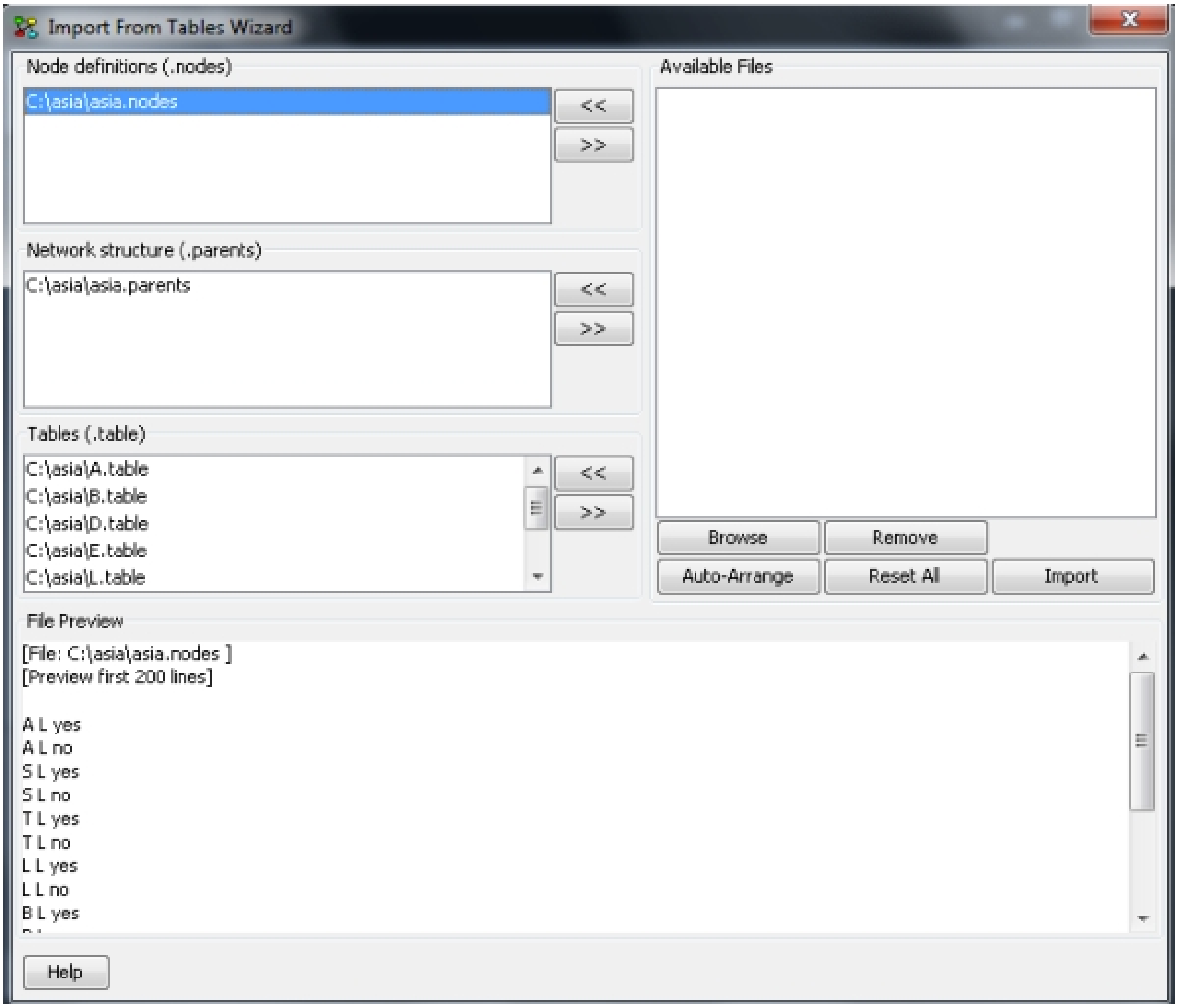
File: X.table

```
X
```

E yes no
yes 0.98 0.02
no 0.05 0.95

Importing in HUGIN

To import the network description click *File -> Import -> Import network from tables*. The Import From Tables Wizard appears, see Figure 1.



Node definitions (.nodes): C:\asia\asia.nodes

Network structure (.parents): C:\asia\asia.parents

Tables (.table):
C:\asia\A.table
C:\asia\B.table
C:\asia\D.table
C:\asia\E.table
C:\asia\L.table

File Preview:

```
[File: C:\asia\asia.nodes ]
[Preview first 200 lines]

A L yes
A L no
S L yes
S L no
T L yes
T L no
L L yes
L L no
B L yes
```

Figure 1: Import From Tables Wizard

Importing:

1. Browse for primitive table files by clicking the *Browse* button.
2. Arrange files onto appropriate lists using the << and >> buttons to move files between the *Available Files* list and *Node definitions*, *Network Structure* and *Tables* lists. Clicking the *Auto-Arrange* button arranges the lists based on file endings (*.nodes*, *.parents*, *.table*).
3. Click on a file to preview its contents.

4. Finally click *Import*.

4.8.19 Load Case

The Load Case function can be activated by selecting the "Load Case" item of the *File Menu* (page 191) when the network is in run mode. It will load the *case* (page 388) stored in the chosen file, and insert the corresponding evidence into the network.

4.8.20 Save Case

The Save Case function can be activated by selecting the "Save Case" item of the *File Menu* (page 191) when the network is in run mode, and evidence has been inserted. It will save the inserted evidence, as a *case* (page 388), in the chosen file.

4.8.21 Recently Used Files

The recently used files are found in the last part of the *File Menu* (page 191) - just above the *Exit* (page 165) item.

This list of HUGIN network files enables the user to open a recently used network faster than using the standard *Open* (page 165) function. Selecting the menu item for one of the recently used files opens this network file in a new *Network Window* (page 182).

In the *Preferences* (page 200) dialog box, you can specify how long this list should be.

4.8.22 Save in a File without GUI Attributes

The "Save in a File without GUI Attributes" menu item can be found in the *File Menu* (page 191), and will write your model file in a file cleansed from GUI Attributes ("HR_").

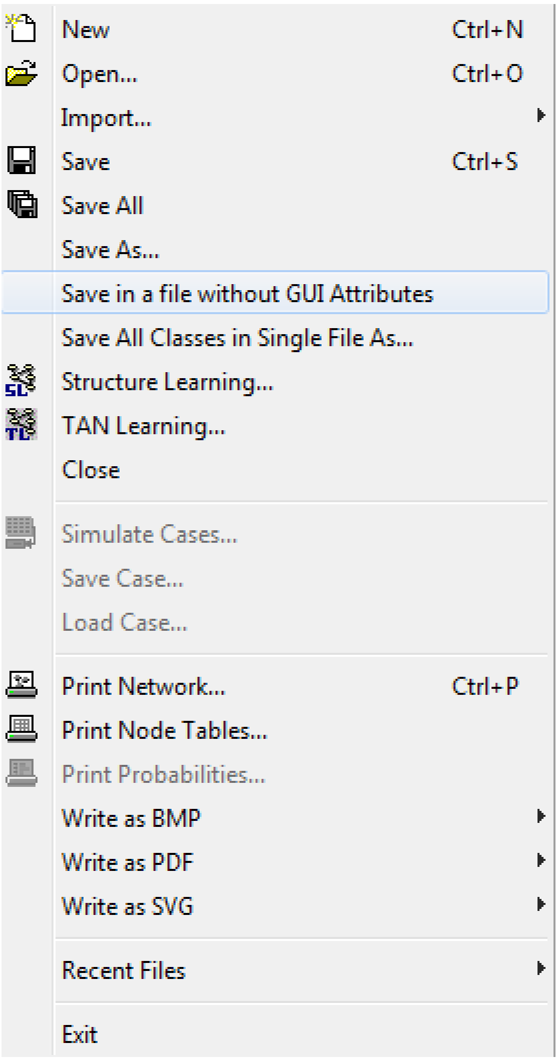


Figure 1: The Save in a File without GUI Attributes Menu Item.

Background

The HUGIN GUI will write some attributes in the network files that are mostly related to how the models are rendered (e.g. splines). These attributes are referred to as GUI Attributes, and can be recognized by the fact that they start with "HR_". The amount of GUI attributes can be quite extensive in some cases and some users have reported that they find the GUI attributes confusing when reading the network files in a text editor. In some cases these attributes may even contain information about nodes that have been deleted. Usually, this would not cause any issues. Since the GUI Attributes are only relevant to how the network is rendered in the HUGIN GUI, they can be removed without having any effect on how the model calculates.

How to remove the GUI Attributes

Once activated, the menu item will overwrite your file with a new file cleansed from GUI Attributes. If the file has not been saved yet, you will be prompted for a destination. Text Labels are implemented using GUI attributes. They are the only GUI attributes that will not be deleted during the operation. In order to complete, HUGIN will close your model window. That is done in order to prevent the HUGIN GUI from writing new GUI attributes which may happen during the normal save procedures.

Limitations

The functionality is only supported for "flat" networks. The model must not be instantiated in any other loaded models and the model must not contain any instantiations.

4.8.23 Write as BMP

The "Write as BMP" submenu can be found in the *File Menu* (page 191), and gives access to the possibility of writing networks, tables, and probabilities as BMPs (for each insertion into other programs).

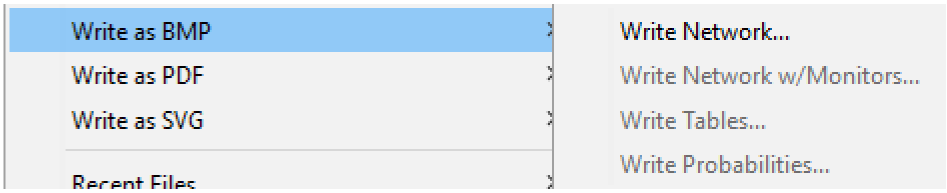


Figure 1: The Write as BMP submenu.

Write Network

The Write Network function saves the active network to a BMP file. Since BMP files can become very large, it is possible that there is not enough memory on the machine to create a large enough file. If this is the case, Hugin will ask for permission to split the BMP into a number of files, which can be created.

Write Network w/Monitors

Same as Write Network, but includes all visible *monitors* (page 194) in the BMP file as well.

Write Tables

The Write Tables function writes the *open* tables to a BMP file. That is, it is not dependent on the selected nodes, only the tables shown in the *table pane* (page 239). Again, the BMP may need to split into several files.

Write Probabilities

The Write Probabilities function writes the current probabilities (beliefs) of the selected nodes to a BMP file. Again, the BMP may need to split into several files.

4.8.24 Write as PDF

The “Write as PDF” submenu can be found in the *File Menu* (page 191), and gives access to the possibility of writing networks (for each insertion into other programs).

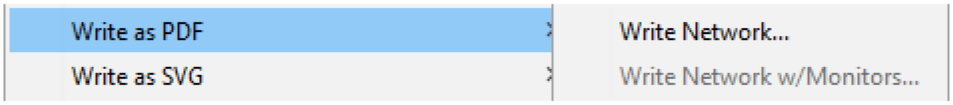


Figure 1: The Write as PDF submenu.

Write Network

The Write Network function saves the active network to a PDF file.

Write Network w/Monitors

Same as Write Network, but includes all visible *monitors* (page 194) in the PDF file as well.

4.8.25 Write as SVG

The “Write as SVG” submenu can be found in the *File Menu* (page 191), and gives access to the possibility of writing networks (for each insertion into other programs). SVG is a vector-based graphics format which renders nicely when imported in documents also when resized. Therefore, the SVG format is recommended when you wish to share your work in books or reports. The SVG format is supported by Word and LaTeX.

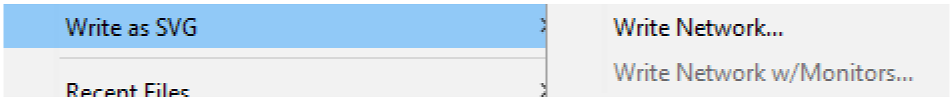


Figure 1: The Write as SVG submenu.

Write Network

The Write Network function saves the active network to an SVG file.

Write Network w/Monitors

Same as Write Network, but includes all visible *monitors* (page 194) in the SVG file as well.

Legal Notice

The SVG feature is based on the JFreeSVG[16] library, with a commercial license from Object Refinery Ltd.[17].

4.8.26 Data Dependences

The Data Dependences page of the Learning Wizard allows you to investigate the strengths of the marginal dependences between pairs of variables, using a slider.

Please note that the actions performed in the Data Dependences page have no effect on the resulting Bayesian-network model. The purpose of this Data Dependences page is only to gain insight into the strenghts of the pairwise dependences.

Interpretation of the Learned Graph

The Data Dependences page of the Learning Wizard initially shows the independence graph learned from data. This graph is directed, but with no directed cycles (i.e., it's a DAG). The DAG represents the conditional and marginal dependences and independences found in the data. The links, however, cannot necessarily be interpreted as causal links. The directions of the links only ensure that the dependences and independences found can be read from the DAG, using e.g. Pearl's d-separation criterion.

Notice that the DAG is learned only indirectly, based on measures of conditional and marginal dependences and independences found in the data.

To further investigate the dependences and independences found in the data, an undirected graph can be shown, where each link represents a marginal dependence with strength larger than one minus the current slider value. To see how to switch between the directed and the undirected graphs, see the description of the toolbar.

The Toolbar

The Data Dependences page contains a toolbar, including the following functionalities:

- *Show Directed Graph* (page 450)
- *Show Undirected Graph* (page 450)
- *Show Enforced Links* (page 450)
- *Show Enforced Non-Links* (page 450)
- *Stretch Slider Scale* (page 451)
- *Compress Slider Scale* (page 451)

[16] http://www.jfree.org/jfreesvg/

[17] http://www.object-refinery.com/

Show Directed Graph

This function displays a directed independence graph (i.e., a DAG). The DAG gets displayed by pressing the button

Please note that the Show Directed Graph and the Show Undirected Graph modes are mutually exclusive (i.e., the two associated buttons act as a couple of radio buttons).

Show Undirected Graph

This function displays an undirected independence graph. This graph is not necessarily identical to the directed graph with the directed links replaced by undirected ones. Each link in the undirected graph represents a marginal dependence with strength greater than one minus the current slider value. The undirected graph gets displayed by pressing the button

As mentioned above, please note that the Show Directed Graph and the Show Undirected Graph modes are mutually exclusive (i.e., the two associated buttons act as a couple of radio buttons).

Show Enforced Links

This function gives you the opportunity to show the must-exist links specified in the Structural Constraints page. These links are shown/hidden by pressing the toggle button

Please note that whenever this toggle button is selected the slider gets disabled to avoid overlaps between enforced links and links appearing and disappearing during sliding.

Show Enforced Non-Links

This function gives you the opportunity to show the must-not-exist links specified in the Structural Constraints page. These links are shown/hidden by pressing the toggle button

Please note that whenever this toggle button is selected the slider gets disabled to avoid overlaps between enforced non-links and links appearing and disappearing during sliding.

p-Value

Whether or not there is going to be a link between a pair of variables, say A and B, in the independence graph learned from the data depends on the degree to which A and B are (conditionally) (in)dependent - if they are marginally dependent, there will be a link; otherwise there won't be a link. This degree is quantified through so-called p-values associated with the hypothesis that the two variables are (conditionally) independent. For each (small) set, C, of conditioning variables, a p-value for {A,B} is computed. This value expresses the probability that A and B are conditionally independent given C. The marginal p-value is the p-value corresponding to C={}.

The *marginal dependence* between A and B is defined as one minus the marginal p-value associated with {A,B}. Thus, a marginal dependence of 0 means that A and B are completely independent, and 1 means that they are completely dependent.

Slider

The current slider value represents a threshold such that only links in the current (directed or undirected) graph with marginal p-values less than the threshold are shown (or, equivalently, links with marginal dependence greater than one minus the threshold value). Thus, the slider provides a means of detecting the marginal strengths of the links. This can be very useful in determining which links should be forced to be included (see the help page for the Structural Constraints page of the Learning Wizard for a more detailed discussion of this issue).

Stretch Slider Scale

The value of the lower endpoint of the slider can be decreased by pressing the button or by dragging the slider ticks downwards, using the mouse.

Please note that the minimum value of the lower endpoint of the slider equals the maximum of the smallest floating point number available and the smallest marginal p-value over all links in the graph.

Compress Slider Scale

The value of the lower endpoint of the slider can be increased by pressing the button by dragging the slider ticks upwards, using the mouse.

Please note that the maximum value of the lower endpoint of the slider equals 1E-10.

Importing Information

Pressing the “Import”-button : , allows for import of all network information, such as node positions, labels, sizes, etc., from a net-file. This can be very useful, if the data relates to a network whose structure is known. In that case, you can simply import the labels and positions of the nodes. The learned network can then easily be compared to the existing one.

4.8.27 Data Acquisition

The Learning Wizard reads data directly from a text for the learning:

Plain Text Files

The plain text files must conform to a certain standard:

- Each line in the file must consist of a list of values
- The values must be separated by a separator, with the default separator being “,”.
- All lines must contain the same number of values
- Missing values are indicated by an empty entry

A sample data file could be:

```
field1,field2,field3
true,true,false
```

```
true,true,
false,false,true
false,true,false
false,,false
```

Using this datafile will result in a model with three nodes named: field1, field2, and field3. Each node will contain the states: true, false.

4.8.28 Feature Selection

The Learning Wizard can perform feature selection in discrete node networks.

When there are many possible features available for a classification problem, we might want to select a subset of these features for the actual classification. We can learn such a subset by using mutual information.

We start with the **target variable**. Then we successively select features such that the next selcted feature has the highest sum of mutual information scores to the currently selected features.

Select target node

The target node for the features.

Number of features

The maximum number of nodes

4.8.29 Nodes Summary

A summary of the node types inferred from the data.

Columns containing numeric data can be either a discrete numbered node or a continuous gaussian node. The user must select desired node type (discrete numbered is the default option).

Numeric -> Numbered

A click on this button configures all numeric nodes as discrete numbered nodes.

Numeric -> Continuous

A click on this button configure all numeric nodes as continuous nodes.

4.8.30 Prior Knowledge

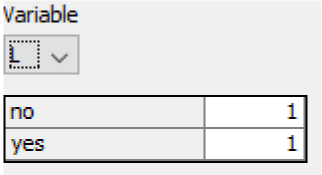
Before the Learning Wizard learns the marginal and conditional probabilites from the data, it is possible to specify prior distributions for the variables.

This is useful if prior knowledge about the (conditional) distributions for (some of) the variables exists. It is possible to include such knowledge in the model before the marginal and conditional probabilities are learned. By including the knowledge before the learning, the resulting probabilities will be based on both the prior knowledge and the data. The weight of the prior knowledge is inserted by specifying experience counts.

An example of the use of both prior probability distributions and their experience counts is given in the last part of this text.

Specifying Prior Conditional Probabilities

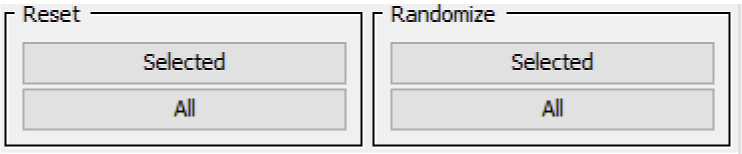
Prior distributions can be specified separately for each variable in the learned structure. When a variable is selected in the "Variable" combo box, its conditional probability table is displayed, and the prior distributions can be entered.



Variable: L

no	1
yes	1

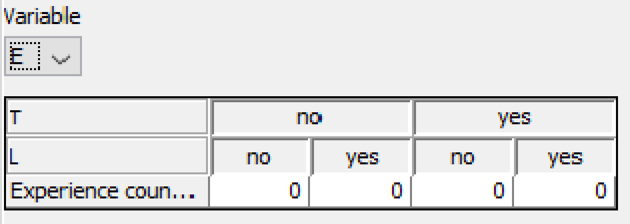
It is possible to either randomize or reset the prior probabilites. This is done by using the appropriate button at the bottom of the panel.



Randomizing the probability distributions can be used to test the "robustness" of the results found from data with missing values. The randomized probability distributions are used as the base when learning the probabilities. Since each missing value is found by looking at the current conditional probability table (after propagation of all evidence from the case), a randomized prior probability distribution may affect the outcome of the learning. If a node has a model, then the conditional probability table corresponding to the expressions specified in the model is generated.

Specifying Experience Counts

Experience counts are specified in the same manner as the prior conditional probabilities. That is, the variable for which experience counts are to be inserted is selected from the "Variable" combo box, and experience counts are entered for each parent configuration.

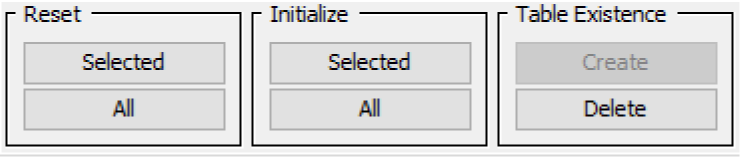


Variable: E

T	no		yes	
L	no	yes	no	yes
Experience coun...	0	0	0	0

It is also possible to reset one or all of the experience tables by pressing one of the buttons, as was the case for the prior probability tables. However, it does not make sense to randomize the experience counts. Instead, it is possible to delete and (obviously) create experience tables.

If a variable does not have an experience table, its probability table will not be altered during the learning process. That is, if the distribution of a variable is known with 100 % accuracy, the experience table can be deleted. This will speed up the learning process a bit.



Experience tables are implicitly only when the "Prior Knowledge" step is performed as a step of the Learning Wizard (and not as a step of the EM Learning Wizard).

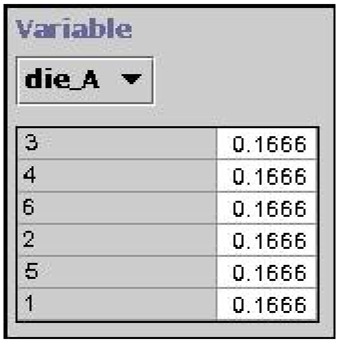
Prior Distribution Example

In this example, the data contains information on the fairness of two brands of six-faced dice. The data includes 1000 cases.

Die B, say, is a completely new brand of which nothing is known, and die A is a well known brand, which has always scored high on fairness.

To incorporate this knowledge into the learned domain, nothing is entered into the tables of die B. This will let the resulting probabilites depend solely on the data.

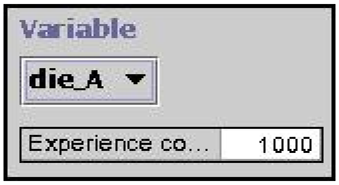
For die A, we will enter the value 0.1666 (one sixth) in all the fields, thus specifying fairness (note that we might as well have left the fields at 1, since the values are normalized so that they will sum to 1; what is important is that the values are all equal).


Variable

die_A

3	0.1666
4	0.1666
6	0.1666
2	0.1666
5	0.1666
1	0.1666

This, however, is not enough. It only describes part of our knowledge. We should also specify our confidence in the fairness of die A. This is done by selecting the "Experience"-pane, and entering experience counts in the table.


Variable

die_A

Experience co...	1000

© Copyright 2023, HUGIN EXPERT A/S

The entered experience counts will determine the weight of the inserted knowledge compared to the data. If we want to let the data have the same weight as the inserted knowledge, we should specify an experience count of 1000. This value tells the learning algorithm that the inserted knowledge has the weight of 1000 cases (and thus the same weight as the data). That is, the more confidence we have in the prior knowledge, the higher the experience counts should be set.

4.8.31 EM Learning

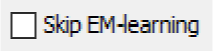
EM Learning is the process of learning the conditional probabilities for the variables from the data. It is an iterative process which improves the estimated conditional probabilities in each iteration. That is, the conditional probabilities will match the data better and better for each iteration.

It is possible to control the process to some extent by using two different parameters:

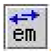
- maximum number of iterations for the process to run.
- convergence threshold.

Once these parameters have been specified, the Learning Wizard can complete the learning. When the conditional probability tables have been learned, the Learning Wizard will exit, and the learned model will be inserted into the Hugin GUI, for further use.

It is possible to skip the learning of the tables by selecting the check box at the bottom of the panel:


□ Skip EM-learning

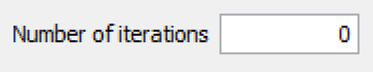
If, however, EM-learning has been skipped, it is still possible to perform the learning "manually" by pressing the EM-learning button in the tool bar panel:



.

Maximum Number of Iterations

As described, EM learning is an iterative process, which continuously improves the resulting conditional probabilities. Setting the maximum number of iterations will force the learning process to terminate when it reaches that number (note, that it may terminate earlier, if the convergence threshold is reached).


Number of iterations 0

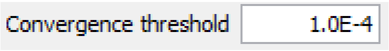
Setting the maximum number of iterations to zero will make the learning process disregard the maximum number of iterations. Convergence Threshold After each iteration, the EM learning will calculate the convergence. This is calculated as:

$$Convergence = logLikelihood(n) - logLikelihood(n-1)$$

Where n is the number of the current iteration, and where:

$$logLikelihood(n) = \sum_{i=0}^{caseCount-1} P(case_i)$$

The learning will process will terminate when the convergence reaches a value smaller than the convergence threshold (or earlier, if the maximum number of iterations is reached).



Convergence threshold 1.0E-4

Note, that the convergence threshold must have a value larger than zero, for the Learning Wizard to finish.

4.8.32 Structural Constraints

The Learning Wizard allows you to specify available knowledge about dependences or independences among pairs of variables in the data set. That is, if a pair of variables are known to be marginally dependent (i.e., a there must be a link between them) or conditionally independent (i.e., there must not be a link between them), such knowledge can be specified by imposing structural constraints upon the graphical model learned from the data. You can specify four different kinds of constraints:

- *Existence of a Causal Link* (page 456)
- *Non-Existence of a Causal Link* (page 456)
- *Existence of a Link* (page 457)
- *Non-Existence of a Link* (page 457)

Please note that only one constraint can be specified per pair of variables.

Existence of a Causal Link

If a variable, say B, is known to be causally dependent on another variable, say A, then this knowledge can be specified using the Arrow Constraint Tool, which is activated by pressing its associated button:

Once activated, the causal link from A to B is specified by pressing the left mouse button at A, dragging the mouse to B, and then releasing the button. Please note that the tool remains activated until you select another tool.

Non-Existence of a Causal Link

If a variable, say B, is known not to be causally dependent on another variable, say A, then this knowledge can be specified using the No-Arrow Constraint Tool, which is activated by pressing its associated button:

Once activated, the constraint that a causal link from A to B is forbidden is specified by pressing the left mouse button at A, dragging the mouse to B, and then releasing the button. Please note that the tool remains activated until you select another tool.

Notice that specifying that a causal link is not allowed from A to B does not disallow a causal link from B to A.

Existence of a Link

If a pair of variables, say A and B, are known to be marginally dependent (i.e., no matter what evidence is given, A and B are dependent), but it is not known which variable causes which, then this knowledge can be specified using the Link Constraint Tool, which is activated by pressing its associated button:

Once activated, the link between A and B is specified by pressing the left mouse button at one of the nodes, dragging the mouse to the other node, and then releasing the button. Please note that the tool remains activated until you select another tool.

Notice that in the model resulting from structural learning, the link between A and B will be directed.

Non-Existence of a Link

If a pair of variables, say A and B, are known to be conditionally independent (i.e., with some, possibly empty, set of evidence given, A and B are independent), then this knowledge can be specified using the No-Link Constraint Tool, which is activated by pressing its associated button:

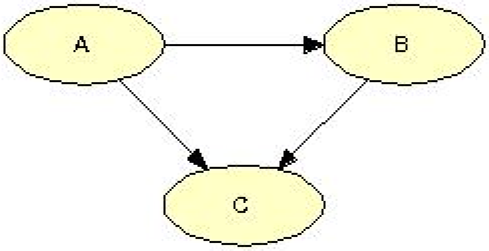
Once activated, constraint that no link is allowed between A and B is specified by pressing the left mouse button at one of the nodes, dragging the mouse to the other node, and then releasing the button. Please note that the tool remains activated until you select another tool.

Usefulness of Enforced Links

Not only can specification of domain knowledge in the form of link constraints be useful in guiding the learning algorithm towards the best possible model, it also can be indispensable in some cases.

Consider, for example, a case where for three variables, A, B, and C, the following statements hold true * A and B are independent given C * A and C are independent given B * B and C are independent given A * A and B are marginally dependent * A and C are marginally dependent * B and C are marginally dependent

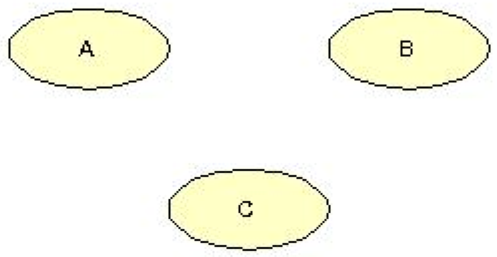
It is not possible to represent all of these statements in a single DAG. The DAG, however, must represent all *dependence* statements (i.e., if a pair of variables are marginally dependent, there must be a link between them), as the posterior probabilities will otherwise be wrong. Unrepresented *independence* statements may lead to more complex inference, but do not lead to wrong probabilities. Therefore, the following DAG is a *correct*, but not a *complete*, representation of the above statements.



However, if for a pair of variables, say X and Y, it holds true that X and Y are conditionally independent given a third variable (or a set of variables), then the PC learning algorithm makes sure that X and Y do not get connected by a link.

Thus, the DAG resulting from the PC algorithm completely represents the independences found in the data, but it does not necessarily represent all dependences.

Thus, in the above example, the following structure will be generated.



Therefore, it can be necessary to manually specify some dependences when using the PC algorithm. Alternatively, one may wish to use the NPC algorithm, which takes care of situations like this (see the help page of the next page of the Learning Wizard). In the subsequent Data Dependences page such marginal dependences can be identified.

Saving / Importing Information

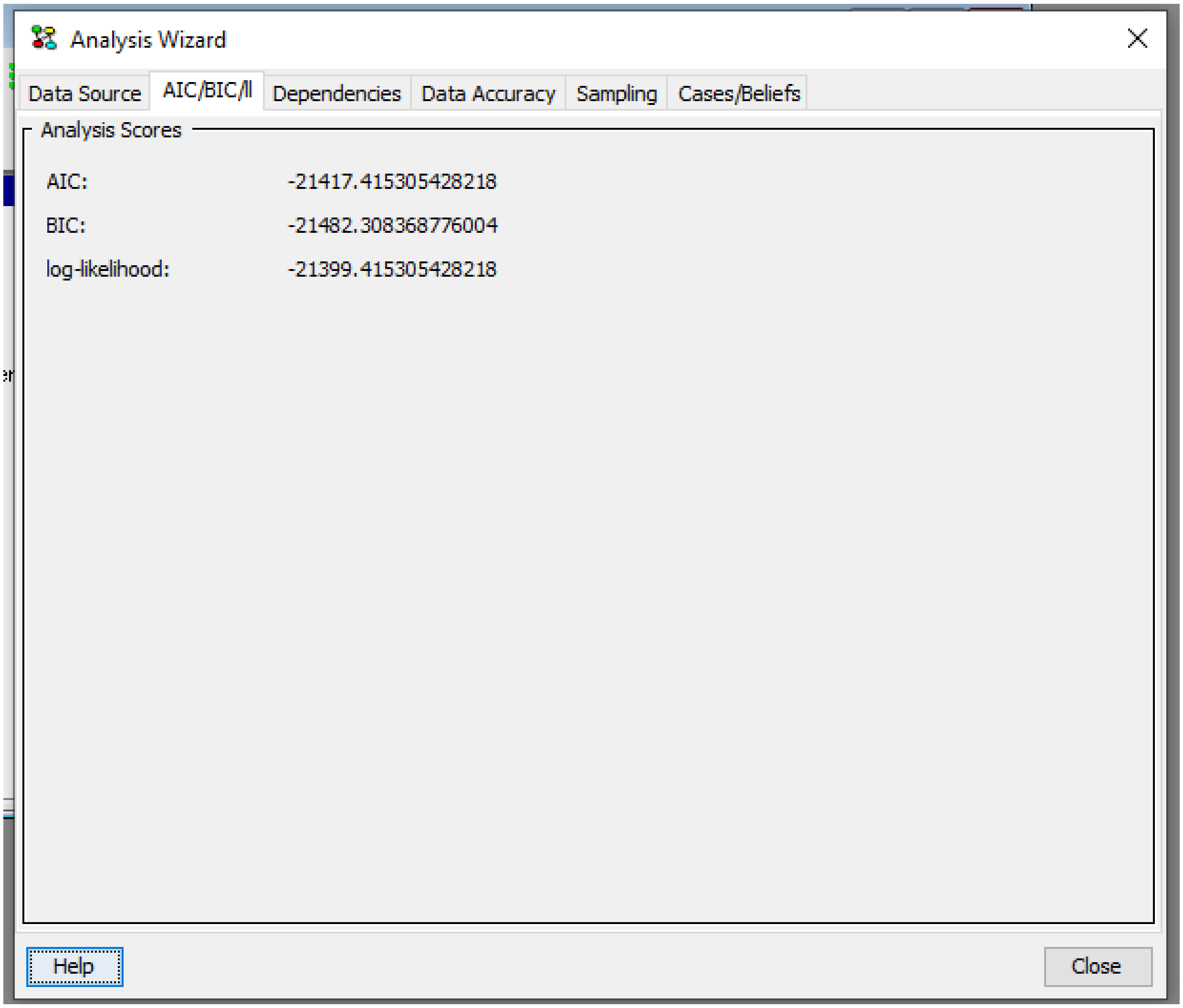
If the learning process is to be repeated a number of time, it can be rather cumbersome to specify the same constraints over and over again. To avoid this, the model information, including the constraints, can be saved to a net-file using the "Save"-button :

This allow for later import of the saved information by using the "Import"-button : .

Note, that this allows for import of all the model information, such as node positions, labels, sizes, etc. This can be very useful, if the data relates to a network whose structure is known. In that case, you can simply import the labels and positions of the nodes. The learned network can then easily be compared to the existing one.

4.8.33 AIC / BIC / LL

Clicking on the "AIC/BIC/ll" tab will bring forward the AIC/BIC/ll pane, see figure 1. The wizard calculates the AIC, BIC and log-likelihood scores of the model given the case data.



AIC: -21417.415305428218

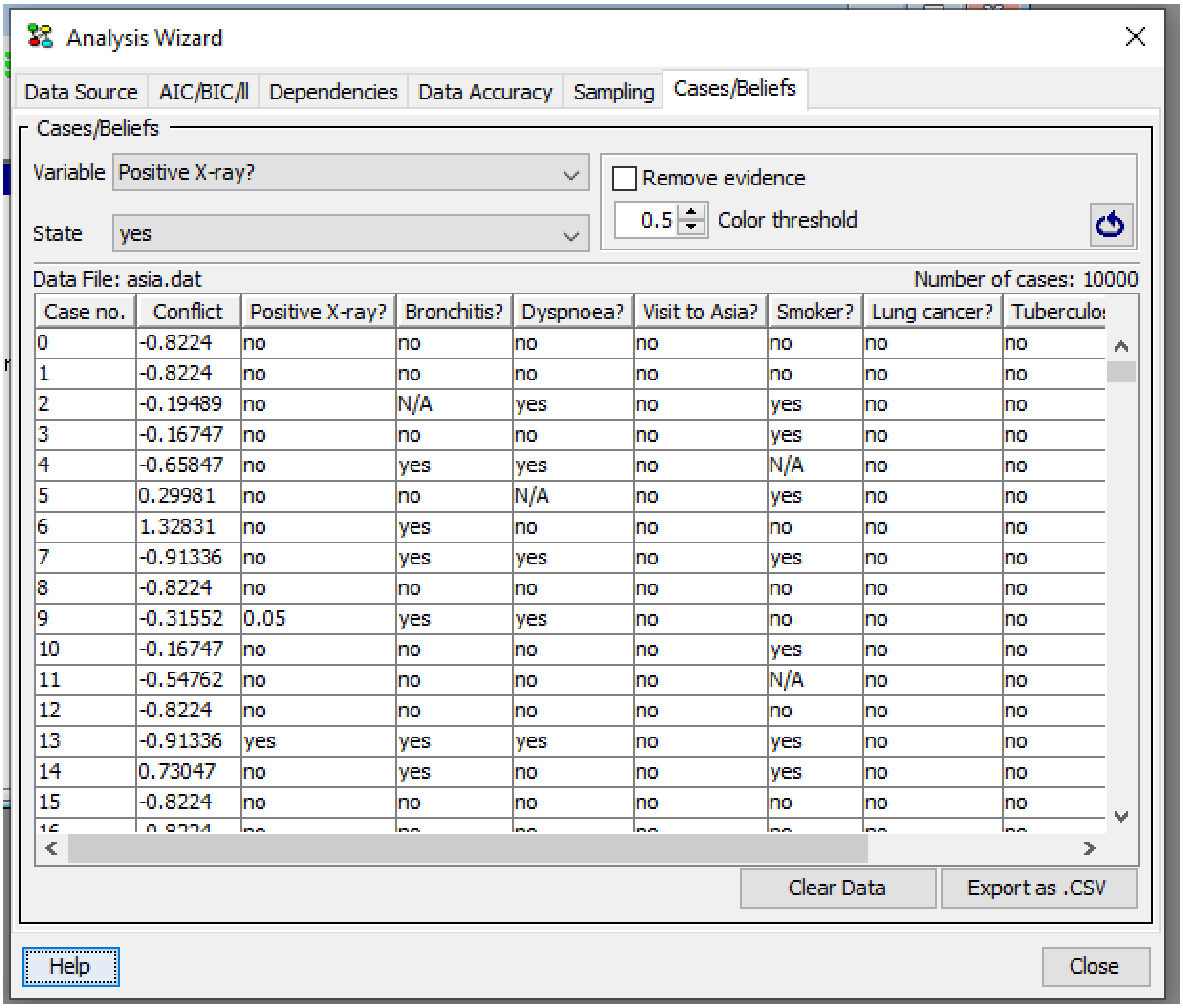
BIC: -21482.308368776004

log-likelihood: -21399.415305428218

Figure 1: AIC/BIC/ll pane.

4.8.34 Case Beliefs

Clicking on the "Cases/Beliefs" tab will bring forward the cases pane, see Figure 1. The case pane offers the option of selecting a node and one of it's states and displaying the beliefs for that state in each case, highlighted by a color.



Cases/Beliefs

Variable: Positive X-ray?

State: yes

☐ Remove evidence

0.5 Color threshold

Data File: asia.dat

Number of cases: 10000

Case no.	Conflict	Positive X-ray?	Bronchitis?	Dyspnoea?	Visit to Asia?	Smoker?	Lung cancer?	Tuberculo:
0	-0.8224	no	no	no	no	no	no	no
1	-0.8224	no	no	no	no	no	no	no
2	-0.19489	no	N/A	yes	no	yes	no	no
3	-0.16747	no	no	no	no	yes	no	no
4	-0.65847	no	yes	yes	no	N/A	no	no
5	0.29981	no	no	N/A	no	yes	no	no
6	1.32831	no	yes	no	no	no	no	no
7	-0.91336	no	yes	yes	no	yes	no	no
8	-0.8224	no	no	no	no	no	no	no
9	-0.31552	0.05	yes	yes	no	no	no	no
10	-0.16747	no	no	no	no	yes	no	no
11	-0.54762	no	no	no	no	N/A	no	no
12	-0.8224	no	no	no	no	no	no	no
13	-0.91336	yes	yes	yes	no	yes	no	no
14	0.73047	no	yes	no	no	yes	no	no
15	-0.8224	no	no	no	no	no	no	no

Figure 1: Cases - Beliefs pane

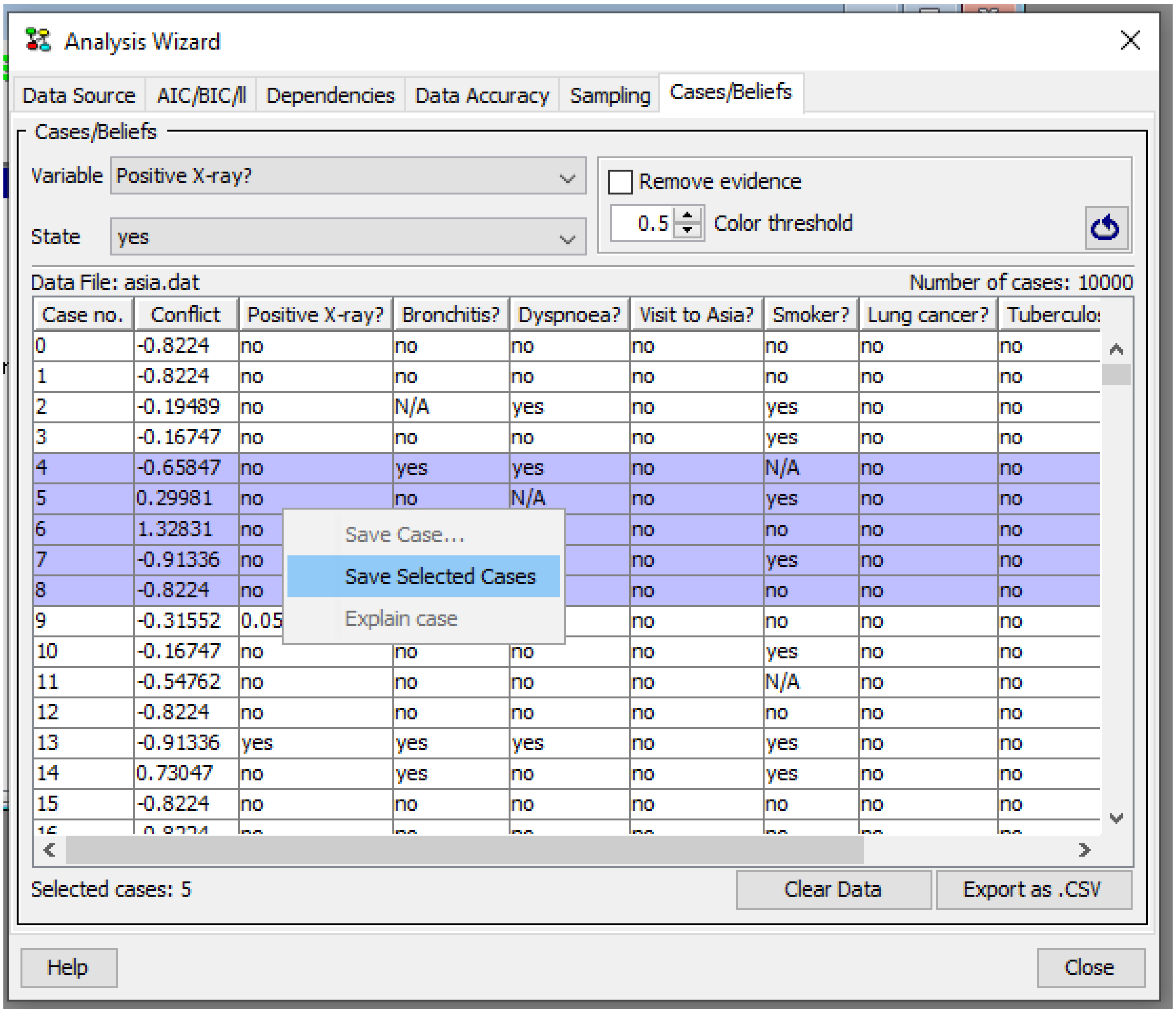
This pane consists of six elements, a drop down box at the top to select what node to analyse, a dropdown box to select one of the node's states, a check box (Remove evidence), the color threshold spinner, a refresh button and a table showing the cases.

By selecting a node and one of its states, the case table displays the beliefs for that node in each case for which no evidence is entered. The information for the selected node will always be displayed in the first column after the *"Case no."* column (second column).

By checking the *Remove evidence check box* and pressing the refresh button, beliefs are retracted and displayed for cases where evidence was entered in the node under analysis

The *color threshold spinner* offers the option of selecting the belief values to be colored (the values higher than the selected are colored). The tone of the color indicates how high the value is (the darker the color the higher the value is). The coloring ability together with the ability to sort the column makes it easier to notice the values desired.

To sort the column click, once on it's header for ascending order or twice for descending. It is possible to save one or more cases to a file so they can be used in another wizard (f.ex Parameter Sensitivity). After selecting one or more cases right click on the mouse and select "Save Case" for saving one case, or "Save Selected Cases" for saving many cases. (see Figure 2).



Cases/Beliefs

Variable: Positive X-ray?

State: yes

□ Remove evidence

0.5 Color threshold

Data File: asia.dat

Number of cases: 10000

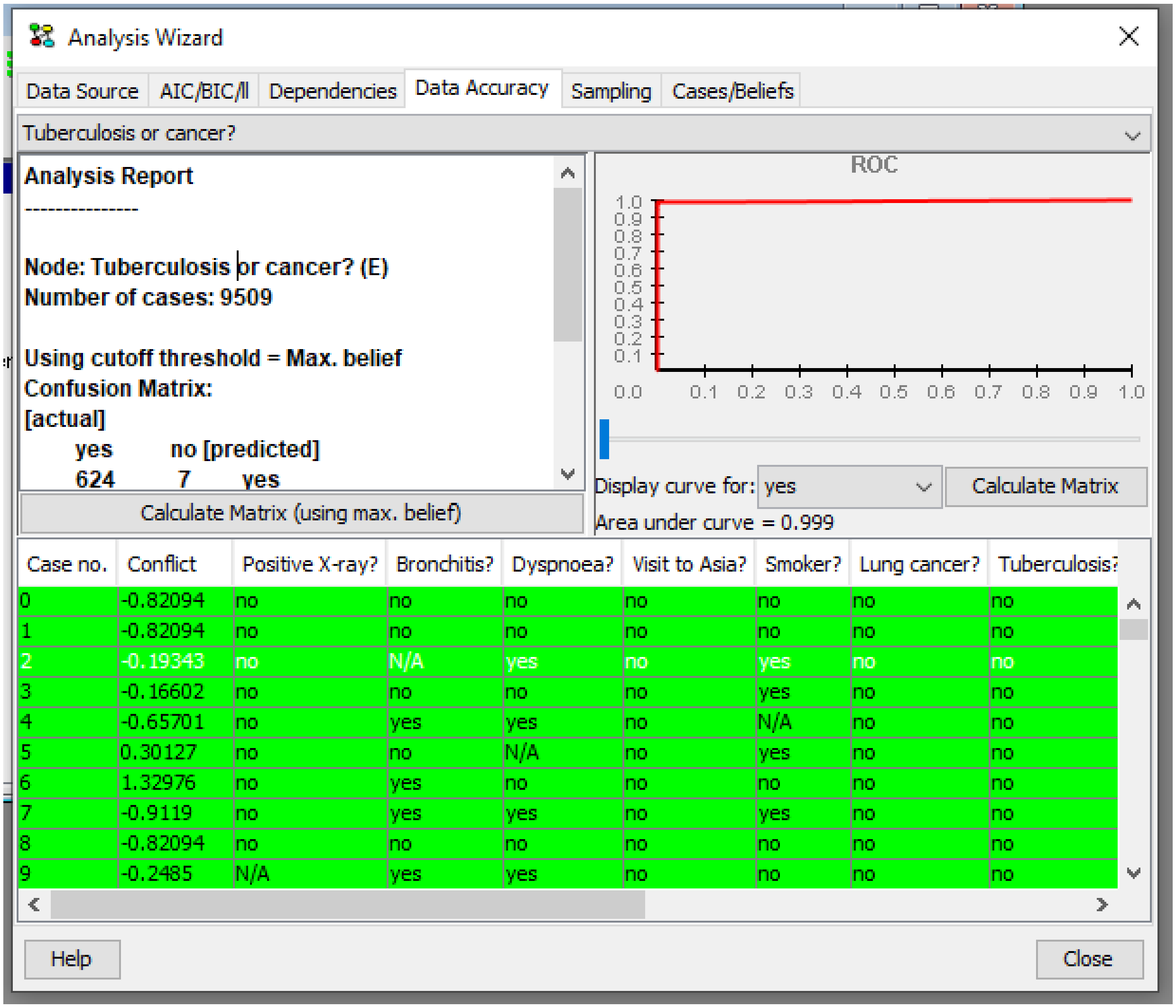
Case no.	Conflict	Positive X-ray?	Bronchitis?	Dyspnoea?	Visit to Asia?	Smoker?	Lung cancer?	Tuberculo:
0	-0.8224	no	no	no	no	no	no	no
1	-0.8224	no	no	no	no	no	no	no
2	-0.19489	no	N/A	yes	no	yes	no	no
3	-0.16747	no	no	no	no	yes	no	no
4	-0.65847	no	yes	yes	no	N/A	no	no
5	0.29981	no	no	N/A	no	yes	no	no
6	1.32831	no			no	no	no	no
7	-0.91336	no			no	yes	no	no
8	-0.8224	no			no	no	no	no
9	-0.31552	0.05			no	no	no	no
10	-0.16747	no	no	no	no	yes	no	no
11	-0.54762	no	no	no	no	N/A	no	no
12	-0.8224	no	no	no	no	no	no	no
13	-0.91336	yes	yes	yes	no	yes	no	no
14	0.73047	no	yes	no	no	yes	no	no
15	-0.8224	no	no	no	no	no	no	no

Selected cases: 5

Figure 2: Cases - Beliefs pane "save selected cases"

4.8.35 Data Accuracy

Clicking on the "Data Accuracy" tab will bring forward the Data accuracy analysis pane, see Figure 1. Given a node and a set of cases, this pane generates an analysis report with information about how well the predictions of the network match the cases.



```
Analysis Report
---------------

Node: Tuberculosis or cancer? (E)
Number of cases: 9509

Using cutoff threshold = Max. belief
Confusion Matrix:
[actual]
      yes       no [predicted]
      624       7      yes
```

Area under curve = 0.999

Case no.	Conflict	Positive X-ray?	Bronchitis?	Dyspnoea?	Visit to Asia?	Smoker?	Lung cancer?	Tuberculosis?
0	-0.82094	no	no	no	no	no	no	no
1	-0.82094	no	no	no	no	no	no	no
2	-0.19343	no	N/A	yes	no	yes	no	no
3	-0.16602	no	no	no	no	yes	no	no
4	-0.65701	no	yes	yes	no	N/A	no	no
5	0.30127	no	no	N/A	no	yes	no	no
6	1.32976	no	yes	no	no	no	no	no
7	-0.9119	no	yes	yes	no	yes	no	no
8	-0.82094	no	no	no	no	no	no	no
9	-0.2485	N/A	yes	yes	no	no	no	no

Figure 1: Data Accuracy pane.

This pane consists of four elements, a drop down box at the top to select what node to analyze, an ROC curve, an analysis report and a table showing the cases.

Analysis Report

The analysis report list the number of cases used for analysis (only cases without observations of the actual node is ignored). A confusion matrix is generated showing how well the observed states match the predicted. Prediction may be based on selecting the state with the highest belief to be the predicted state, or based on selecting a state with a belief greater than or equal to the ROC cutoff threshold (Note that if a node has two or more states that qualify to be the predicted state, one of them is randomly selected as the predicted state). An error rate is calulated telling how many cases actually match the predictions. The button Calculate Matrix (using max. belief) makes a report where prediction is based on selecting the state with the highest belief as the predicted state. Average euclidian distance and Kulbach-Leibler divergence are reported. The distance measures are from the true distribution x inferred from the case data e with respect to the selected node X, and the distribution y resulting from propagating eX.

The Euclidian distance is computed as:

$$dist_Q(x, y) = \sum_i (x_i - y_i)^2$$

The Kulbach-Leibler divergence is computed as:

$$dist_K(x, y) = \sum_i (log_2 x_i - log_2 y_i)^2$$

The reported distance measure is averaged over all cases where X has been observed.

ROC Curve

The ROC curve lets you inspect the performance of a given variable as a classifier for the data set. X-axis is the false positive rate, and the Y-axis is the true positive rate. The ROC curve may be based on the accumulated performance for predicting all states (multi-class), or a specific state. The area under the can be used as a measure for the "goodness" of the network as a classifier for the given variable. The button *Calculate Matrix* makes a report where prediction is based on selecting a state with a belief greater than or equal to the ROC cutoff threshold as the predicted. The threshold can be specified by moving the threshold slider left and right. The current threshold is also signified by a point on the curve, signifying the expected performance with regards to the ratio between true and false predictions.

Case table

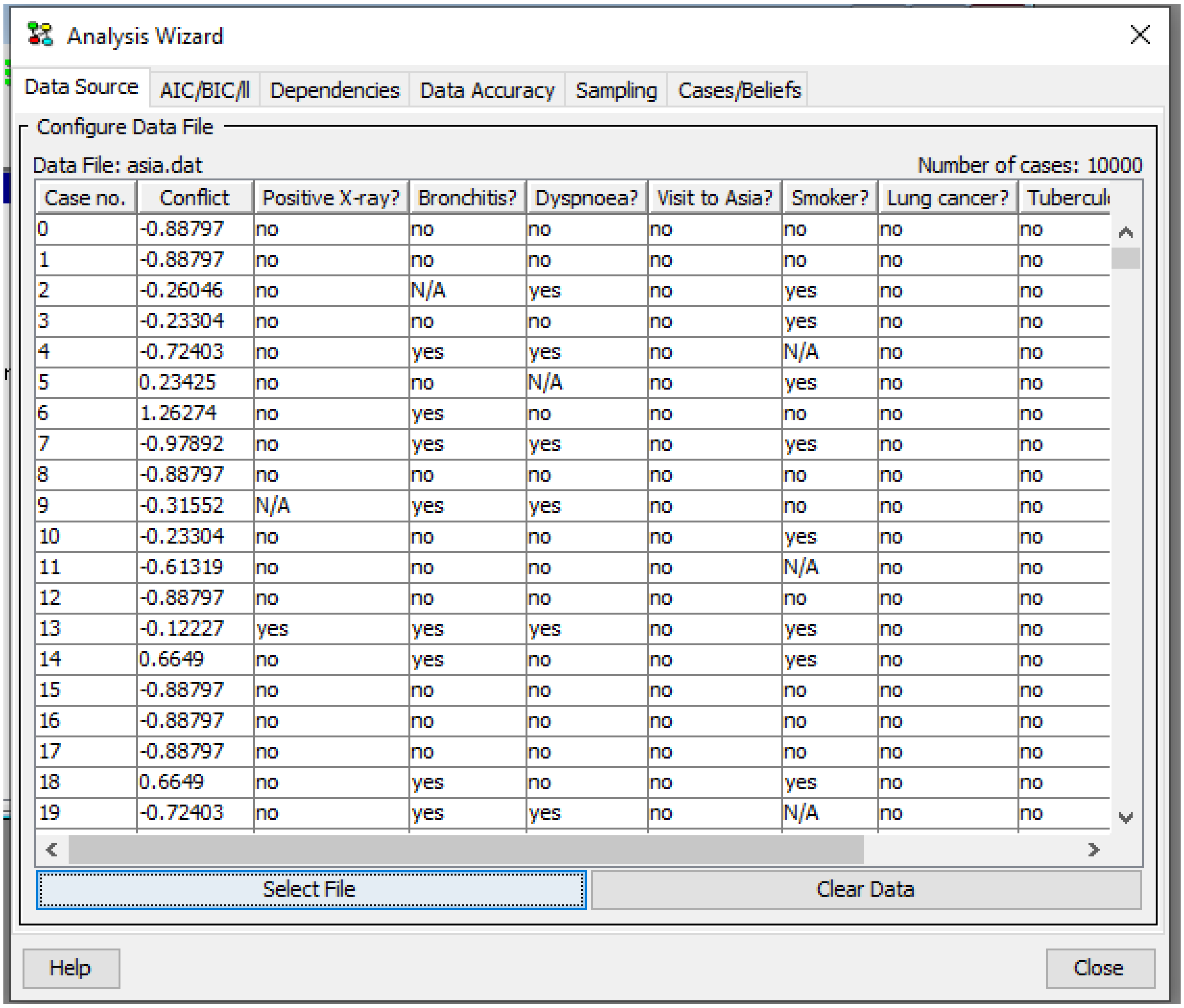
The table shows the cases used for analysis. The node selected for analysis will always be located in the second last column of the table. The last column in the table report the probability of the node being in the state observed in the case, given the case without information about the state of the node itself. Cases are colored green for a match, red for no match and blue if ignored. A match is when the predicted state for the node is the same state observed in the case.

4.8.36 Data Source

To perform any analysis the wizard will first need a set of cases. Cases can be imported by * loading a hugin data file, * Connect to a database service and import. * copying cases (if any) from the network, * or sampling a number of random cases.

Click on button "select file" to load a hugin data file, button "use cases from network" to copy cases from the network and click on the tab "Sampling" to sample a number of random cases to use as data source.

After importing a set of cases, the Data Source pane will display a table with the imported cases, see Figure 1. To discard all imported cases click on the button "Clear Data".



Data File: asia.dat

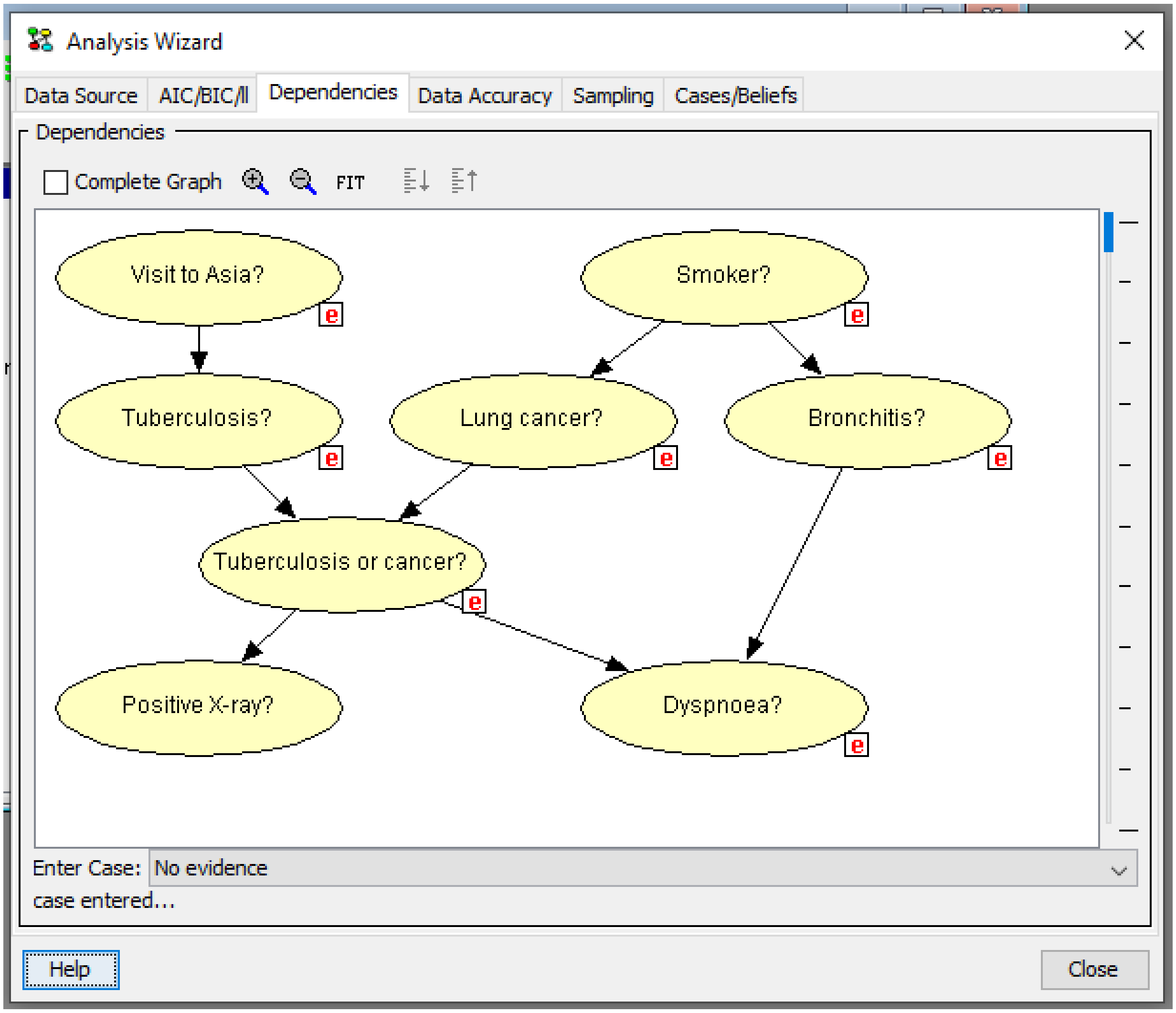
Number of cases: 10000

Case no.	Conflict	Positive X-ray?	Bronchitis?	Dyspnoea?	Visit to Asia?	Smoker?	Lung cancer?	Tubercul
0	-0.88797	no	no	no	no	no	no	no
1	-0.88797	no	no	no	no	no	no	no
2	-0.26046	no	N/A	yes	no	yes	no	no
3	-0.23304	no	no	no	no	yes	no	no
4	-0.72403	no	yes	yes	no	N/A	no	no
5	0.23425	no	no	N/A	no	yes	no	no
6	1.26274	no	yes	no	no	no	no	no
7	-0.97892	no	yes	yes	no	yes	no	no
8	-0.88797	no	no	no	no	no	no	no
9	-0.31552	N/A	yes	yes	no	no	no	no
10	-0.23304	no	no	no	no	yes	no	no
11	-0.61319	no	no	no	no	N/A	no	no
12	-0.88797	no	no	no	no	no	no	no
13	-0.12227	yes	yes	yes	no	yes	no	no
14	0.6649	no	yes	no	no	yes	no	no
15	-0.88797	no	no	no	no	no	no	no
16	-0.88797	no	no	no	no	no	no	no
17	-0.88797	no	no	no	no	no	no	no
18	0.6649	no	yes	no	no	yes	no	no
19	-0.72403	no	yes	yes	no	N/A	no	no

Figure 1: Data Source pane. A set of cases has been imported.

4.8.37 Dependencies

Clicking on the “Dependencies” tab will bring forward the data dependencies pane, see figure 1.



Enter Case: No evidence

case entered...

Figure 1: Data dependencies pane.

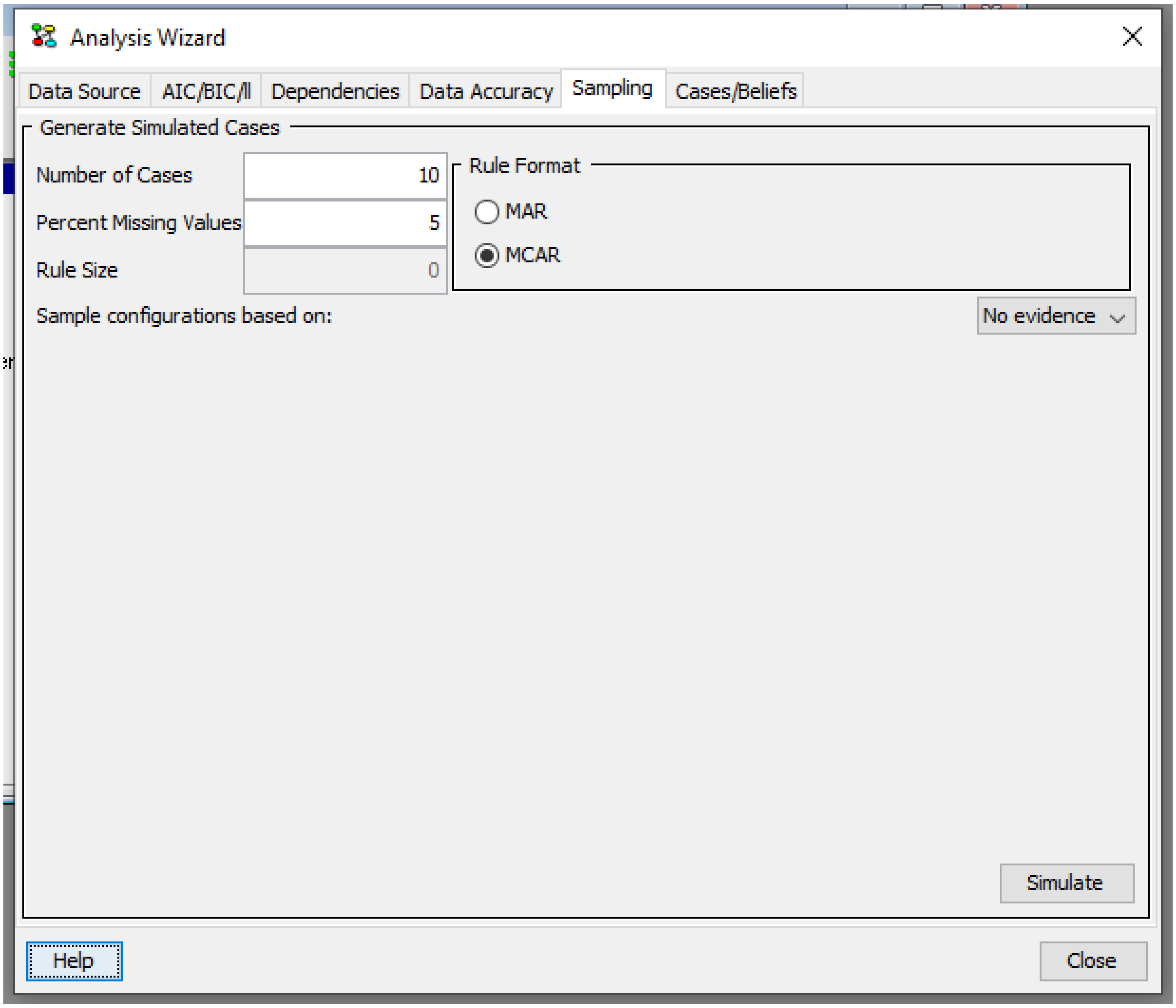
The Dependencies pane allow for inspecting the mutual information of nodes. Depending on the threshold set by the slider, edges will be hidden according to the mutual information of the nodes. Toggle between inspecting mutual information for all nodes or nodes with directed edges by clicking the checkbox “Complete graph”. Mutual information will be computed based on the case selected in the “Enter Case” drop down box.

Buttons at the top of the pane allows for zooming the view of the network and adjusting the scale of the slider if possible.

- Stronger and weaker dependencies can be identified by moving the threshold slider, as edges are only displayed if the mutual information between the connected nodes are above the threshold.
- Mutual information between all nodes can be inspected by clicking ‘Complete Graph’.
- Mutual information computation will be based on the the network instantiation chosen in the ‘Enter Case’ drop-down list

4.8.38 Sampling

Clicking on the tab “Sampling” will bring forward the sampling pane. The sampling pane is shown in figure 1.



Generate Simulated Cases

Number of Cases: 10

Percent Missing Values: 5

Rule Size: 0

Rule Format

○ MAR

● MCAR

Sample configurations based on: No evidence

Figure 1: Sampling pane.

Cases can be generated in two different ways: MCAR (Missing Completely at Random) or MAR (Missing at Random). MCAR sets some values to N/A by removing values of some nodes in the generated case set randomly (i.e., MCAR considers all the values in the generated cases and not one case at a time).

MAR randomly sets some values in a case to N/A based on auto-generated templates. These templates specify that if some nodes have a specific value, then the values of a subset of the nodes are randomly set to N/A. Note that the specification of the templates is generated randomly.

In both cases the conditional probability distribution is considered as a factor to the number of cases generated with a specific set of observations. Also in both MCAR and MAR it is possible to indicate the percentage of missing values.

The cases generated will be based on the conditional probability distribution given by the scenario chosen in the combo box “Sample configurations based on”.

Any cases generated will be added to the current set of imported cases.

4.8.39 Application Update

Each time the Hugin Graphical User Interface is launched it will check the http://www.hugin.com web-site for new updates. If a new update is available, the user is given the option to download and install the update.

The updating procedure consists of the following few steps: * Download a new jar-file to temporary directory * *And when Hugin is closed:* * Rename existing *hgui.jar* to *hgui.old* * The new jar-file is moved to the location of the "old" *hgui.jar*

If any of the above steps fail, the update will not be correctly installed. If the update is not correctly installed, the user will be prompted to install the latest build each time the application is launched.

Experience has shown that the installation of an update may fail for many reasons such as insufficient privileges of the user to save and rename files, security issues (e.g. related to firewalls, etc), etc.

When the installation fails, the update may be installed manually by downloading an updated jar-file from http://www.hugin.com and replacing the existing *hgui.jar* file in the installation directory.

The check for updates functionality may be disabled under the *Preferences* menu by deselecting *Check for updates*.

4.8.40 What is Data Conflict Analysis?

Conflict analysis is the activity of detecting, tracing, and explaining possible conflicts among observations of variable values (i.e., evidence or data). Inconsistencies among observations are easily detected (P(evidence) = 0), but also flawed findings should be detected and traced. For example, in a diagnostic situation a single flawed test result may take the investigation in a completely wrong direction. To understand what conflict analysis is and how it can be used, there are several issues of interest: * *Definition of Data Conflict* (page 467): How do we define data conflict? * *Conflict Measure* (page 468): How do we define an appropriate measure of data conflict? * *Conflict Resolution* (page 468): How do we distinguish between a conflict and a rare case? * *Tracing Conflicts* (page 469): How do we identify the pieces of evidence that contribute to a data conflict? * *Hypothesis Variables* (page 469): What set of uninstantiated variables should be considered when searching for a hypothesis or observation that may eliminate the current conflict? * *Partial Conflicts* (page 469): How * *Individual Conflicts* (page 469): How does each piece of evidence impact a given hypothesis? * *Hypothesis Driven Conflict Analysis* (page 470): How does each piece of evidence impact a given hypothesis?

Definition of Data Conflict

We define two sets of observations e_1 and e_2 to be in a possible conflict with one another if they are negatively correlated.

For positively correlated findings we expect that P($e_{\text{1`|e:sub:`2}}$) > P(e_1) and vice versa (i.e., observing e_2 makes it more likely to also observe e_1 (and vice versa)). In other words, we expect that

$$P(e_1, e_2) > P(e_1)P(e_2)$$

if e_1 and e_2 are positively correlated,

$$P(e_1, e_2) < P(e_1)P(e_2)$$

if e_1 and e_2 are negatively correlated, and

$$P(e_1, e_2) = P(e_1)P(e_2)$$

if e_1 and e_2 are independent.

Conflict Measure

Therefore, given a set of observations (evidence), e = {e_1,...,e_n}, we define the conflict measure for e as

$$conf(e) = log\frac{\prod_{i=1}^{n} P(e_i)}{P(e)}$$

If conf(e) is positive, e_1,...,e_n are negatively correlated, indicating a possible conflict among these pieces of evidence. (The choice of base for the log function is immaterial.)

Notice, that if conf(e) is negative (i.e., no apparent conflict among e_1,...,e_n), then this gives you no guarantee that all of e_1,...,e_n are positively correlated. It may well happen that there is a local conflict (i.e., that conf(e') > 0 for a proper subset e' of e) although conf(e) < 0.

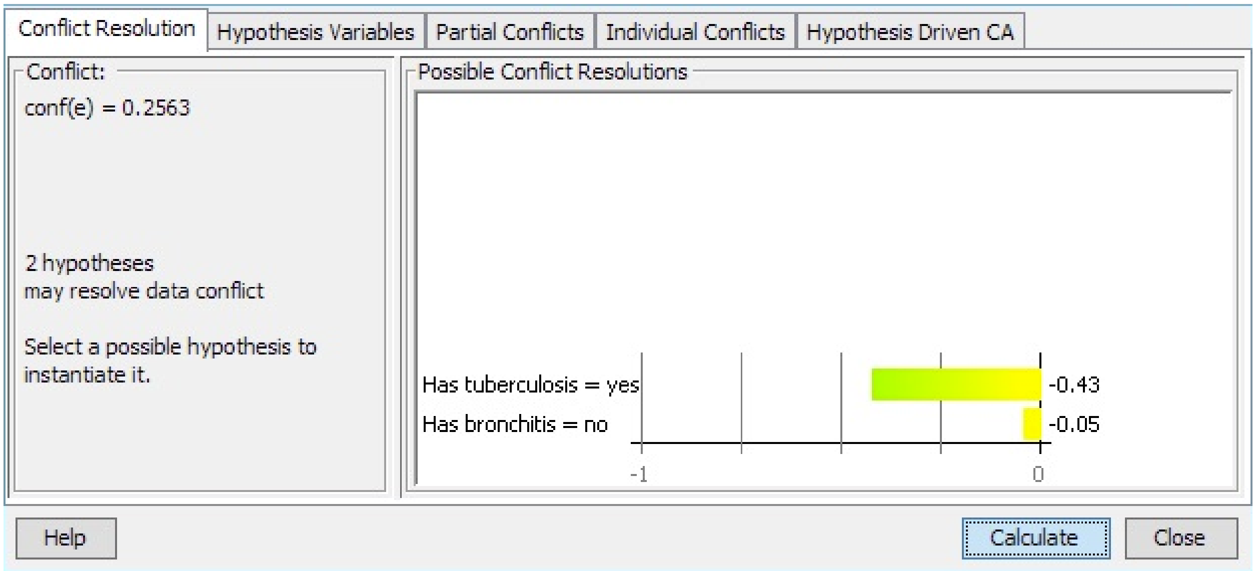
For more information about detection of local conflict, see the help page of the junction tree panel.

Conflict Resolution

There are situations in which a positive conflict measure is computed, where there is no real conflict. These include:

- *Rare case*: Typical data from a very rare case may indicate a possible conflict. If conf(e_1,...,e_n) > 0 and there is a hypothesis H=h such that conf(e_1,...,e_n,h) < 0, then h explains away the conflict. That is, if H=h is the correct hypothesis (e.g., a diagnosis) in the current situation, then there is no conflict.
- *Missing observation*: Basically the same situation, where conf(e_1,...,e_n) > 0 but conf(e_1,...,e_n,I=i) < 0, where I=i is a missing piece of information. That is, there is a local conflict among e_1,...,e_n, but the observation I=i explains the conflict.

By activating the button with the 🔑 symbol, one can obtain a list of possible instantiations of currently uninstantiated variables that can eliminate the current conflict. An example of the dialog box that appears when this button is activated is shown in Figure 5.



Conflict:

conf(e) = 0.2563

2 hypotheses
may resolve data conflict

Select a possible hypothesis to
instantiate it.

Figure 5: Conflict resolutions dialog box. The box contains a list of possible instantiations of currently uninstantiated variables that can eliminate the current conflict.

The dialog box contains a list of possible instantiations in the form

::

    <CM>:<variable>=<value>

where <CM> is the new conflict measure obtained if <variable> is instantiated to <value>. Only instantiations (if any) with a resulting conflict measure less than or equal to 0 get displayed.

The Instantiate button enters the currently selected instantiation (if any) as evidence.

Tracing Conflicts

Whenever a positive conflict has been observed that cannot be explained as a rare case, it is important to pinpoint the piece (or pieces) of evidence that is in conflict with the majority of the pieces of evidence

Basically, this involves computation of conflict measures for subsets of the evidence. The junction tree is useful for this purpose; see the help page for the junction tree panel for more information.

Hypothesis Variables

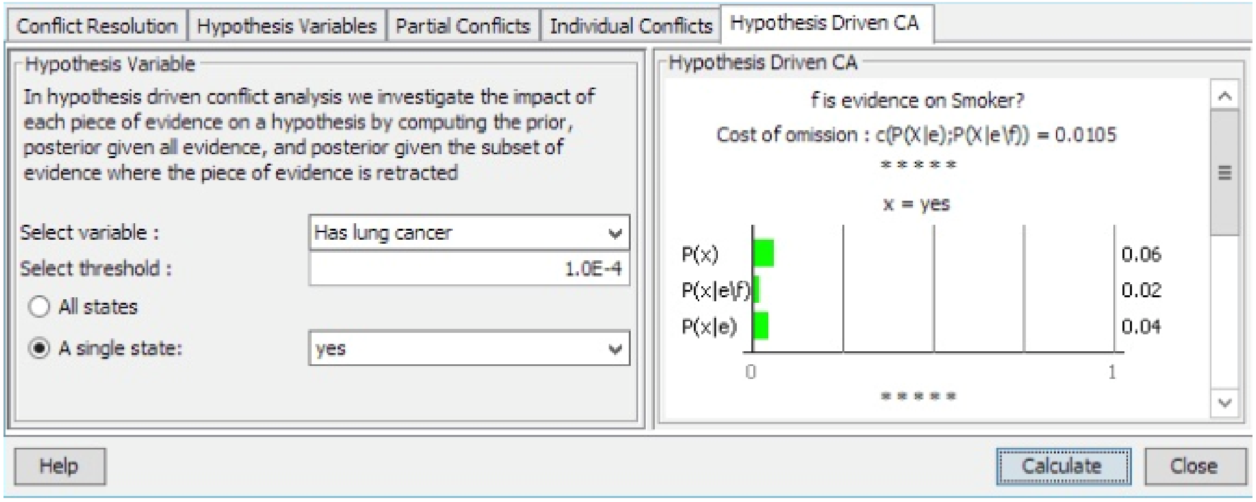
When searching for a hypothesis or observation that may eliminate the current conflict it may be desirable to restrict the set of uninstantiated variables considered in the search. This is done by selecting a set of hypothesis variables. Any subset of discrete chance nodes may be selected as hypothesis variables. Only the selected set of hypothesis variables are considered in the search for instantiations that may eliminate the current conflict.

Partial Conflict Analysis

To support tracing the source of a conflict or to identify conflict in a subset of the evidence, partial conflict analysis is possible. Here the partial conflicts for subsets of the evdience are computed. This is useful to trace the source of a conflict in the overall entered and propagated evidence. To reduce the computational burden it is possible to define an upper limit on the size of the subsets considered.

Individual Conflict Analysis

This functionality is useful to investigate the contribution of each individual finding to the overall conflict in the evidence. To assess the contribution of each finding to the conflict we compute the normalized likelihood of that finding given the other findings in the evidence.



In hypothesis driven conflict analysis we investigate the impact of each piece of evidence on a hypothesis by computing the prior, posterior given all evidence, and posterior given the subset of evidence where the piece of evidence is retracted

Select variable : Has lung cancer

Select threshold : 1.0E-4

○ All states

● A single state: yes

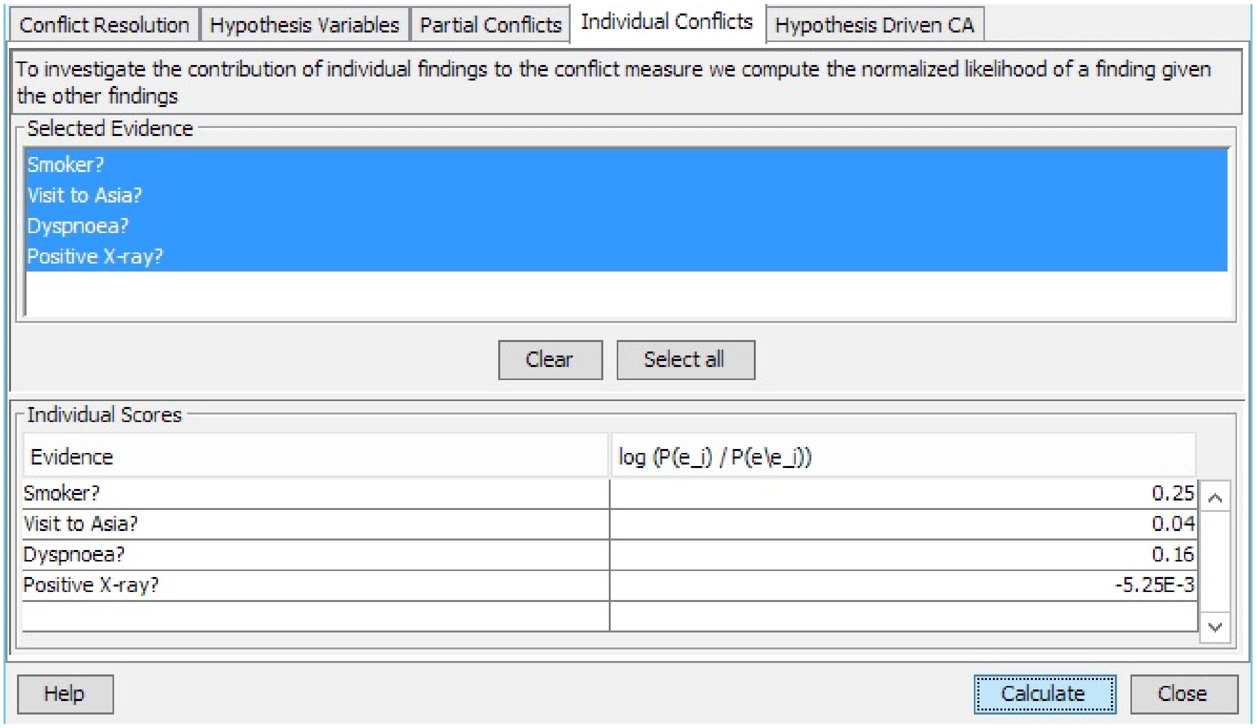
```
f is evidence on Smoker?
Cost of omission : c(P(X|e);P(X|e\f)) = 0.0105
*****
x = yes
P(x)      🟩  0.06
P(x|e\f)  🟩  0.02
P(x|e)    🟩  0.04
0          1
*****
```

The figure above shows an example where the conflict measure for the entire set of evidence is 0.2511. Using individual conflicts it is possible to identify Smoker? as the finding that has the highest contribution to the evidence and Dyspnoea? as the second highest contribution to the evidence as measured by the normalized likelihood.

Hypothesis Driven Conflict Analysis

So far we have considered data conflict analysis, i.e. how to identify, resolve, or explain a possible conflict in data. In hypothesis driven data conflict analysis we investigate how each piece of evidence impacts a given hypothesis.

In hypothesis driven conflict analysis we investigate the impact of each piece of evidence on a given hypothesis. For each piece of evidence (finding) f we compute the prior P(h) of the hypothesis, the posterior P(h|e) of the hypothesis given the entire set of evidence e and the posterior P(h|ef) of the hypothesis given the subset of evidence where the piece of evidence f under consideration is retracted.



To investigate the contribution of individual findings to the conflict measure we compute the normalized likelihood of a finding given the other findings

Selected Evidence

Smoker?
Visit to Asia?
Dyspnoea?
Positive X-ray?

Individual Scores

Evidence	log (P(e_i) / P(e\e_i))
Smoker?	0.25
Visit to Asia?	0.04
Dyspnoea?	0.16
Positive X-ray?	-5.25E-3

Hypothesis driven conflict analysis allows the user to investigate how a single piece of finding impacts the probability of the hypothesis.

A threshold value is applied to reduce the number of findings considered. Only results for findings where the cost-of-omission is above the threshold are displayed where cost-of-omission is a measure of the distance between P(H|e) and P(H|e\f).

4.8.41 Structural Uncertainties Help

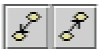
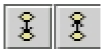
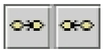
The structure derived by the NPC algorithm contains ambiguous regions (i.e., groups of inter-dependent uncertain links) and/or other undirected links. The current page of the Learning Wizard provides you with an intuitive graphical interface for resolving these structural uncertainties.

Should you not wish to provide such information, just click the Next button, and the NPC algorithm will resolve the uncertainties. Note, however, that directionality for undirected links will be decided on a random basis, and that for those ambiguous regions with no information provided, all of the uncertain links will be removed, possibly resulting in a poor model.

For more information on the NPC algorithm and the notions of undirected links and ambiguous regions, see the help page of the previous page of the Learning Wizard.

Undirected Links

Instead of randomly determining the directionality of the links of the learned structure that cannot be determined automatically from the data, the NPC algorithm gives the user the apportunity to determine the directionality of such links. The undirected links (not belonging to ambiguous regions) are drawn in black. When selected, such a link gets highlighted and the two directionality buttons get enabled:

or  or  or 

Which of the above appearances of the pair of buttons are used depends on the relative positioning of the nodes of the selected link. Thus, it should be readily apparent which of the two buttons should be selected for achieving the desired directionality for the selected link. However, when positioning the mouse cursor on top of a button, a tool tip will appear after a short while, explaining precisely which direction is associated with the button in question.

Note that assigning directionality to a link may cause other links to automatically be directed. If, for example, there are undirected links between variables x and y, between y and z, and between x and z, then enforcing the directionalities x –> y and y –> z implies x –> z; otherwise, a directed cycle would occur.

Ambiguous Regions

An ambiguous region consists of a set of inter-dependent uncertain links: Absence of a link in an ambiguous region depends on the presence of one or more of the other links of the region, and vice versa. For more information on ambiguous regions, see the help page of the previous page of the Learning Wizard. Each ambiguous region is easily identified as consisting of links in the same color. (Note that there can be so many ambiguous regions that colors have to be reused, making it difficult from the coloring alone to distinguish them.) Also, when selecting a link of an ambiguous region, all links of the region will be highlighted; the one selected will be drawn bold and the other links will be drawn with double lines.

When a link of an ambiguous region has been selected, the include/exclude link buttons get enabled:

When a decision has been made that a link should be present or absent, each of the other links of the ambiguous region will be affected in one of several ways:

- Unaffected.
- Disappear.
- Turned into an undirected link, not belonging to the region anymore.

- Turned into a directed link, not belonging to the region anymore.

Which of these consequence will be observed depends on the conditional independence and dependence statements (CIDs) found from the statistical tests performed by the NPC algorithm. For more information see the help page of the previous page of the Learning Wizard, or consult the references mentioned on that help page.

Undo

Often it is impossible (or very hard) to predict the consequences of a decision made regarding directionality of an undirected link, or regarding the presence or absence of a link of an ambiguous region. Therefore, you will probably find the undo facility of the current page of the Learning Wizard quite useful. The most recent (non-undo) action performed can be undone by clicking the undo button:

↶

Please note that an arbitrary number of operations performed can be undone by repeatedly clicking the undo button.

Importing Information

Pressing the “Import”-button : , allows for import of all network information, such as node positions, labels, sizes, etc., from a net-file. This can be very useful, if the data relates to a network whose structure is known. In that case, you can simply import the labels and positions of the nodes. The learned network can then easily be compared to the existing one.

p-Value

Whether or not there is going to be a link between a pair of variables, say A and B, in the independence graph learned from the data depends on the degree to which A and B are (conditionally) (in)dependent - if they are marginally dependent, there will be a link; otherwise there won’t be a link. This degree is quantified through so-called p-values associated with the hypothesis that the two variables are (conditionally) independent.

For each (small) set, C, of conditioning variables, a p-value for {A,B} is computed. This value expresses the probability that A and B are conditionally independent given C. The *marginal p-value* is the p-value corresponding to C={}.

4.8.42 Structure Learning Help

At this point, the Learning Wizard is ready to learn the structure of the model from the (possibly preprocessed) data and with the help of the possible structure constraints specified.

The user can select from six different algorithms for learning the structure of the model from data: *The PC algorithm* (page 246), the *NPC algorithm* (page 246), the *Greedy search-and-score algorithm* (page 258), the *Chow-Liu tree algorithm* (page 254), the *Rebane-Pearl polytree algoritm* (page 253) or the *Tree Augmented Naive Bayes algorithm* (page 253). The first to algorithm (PC and NPC) are constraint-based approaches whereas the later two are both based on the Chow-Liu algorithm for learning a tree representation of the underlying joint distribution.

Note that, in this step of the Learning Wizard, only the structure of the network is determined. The conditional probability tables (CPTs) to be associated with the learned structure, will be learned in a subsequent step.

Constraint-Based Algorithms

Two pieces of information are needed in order to start the learning process using the PC or NPC algorithm: * the *level of significance* (page 120) to use for the learning, and * the learning algorithm to be used; either the *The PC algorithm* (page 246) or the *NPC algorithm* (page 246).

Level of Significance

The PC and NPC structure learning algorithms are based on making dependence tests that calculate a test statistic which is asymptotically chi-squared distributed assuming (conditional) independence. If the test statistic is large for a given independence hypothesis, the hypothesis is rejected; otherwise, it is accepted. The probability of rejecting a true independence hypothesis is given by the *level of significance*:

Level of Significance 0.05

The level of significance can be entered into this text field. The default value is set to 0.05, which should be appropriate for most learning sessions, but it can take on any value in the open interval (0;1). In general, the higher the significance level the more links will be included in the learned structure. Reducing the level of significance will usually also reduce the running time of the structure learning algorithms.

The PC Algorithm

The PC algorithm, which is a variant of the original PC algorithm due to Peter Spirtes and Clark Glymour, belongs to the class of constraint-based learning algorithms. The basic idea of these algorithms is to derive a set of conditional independence and dependence statements (CIDs) by statistical tests.

The algorithm performs the following steps:

- Statistical tests for conditional independence are performed for all pairs of variables (except for those pairs for which a structure constraint has been specified).
- An undirected link is added between each pair of variables for which no conditional independences were found. The resulting undirected graph is referred to as the *skeleton* of the learned structure.
- Colliders are then identified, ensuring that no directed cycles occur. (A *collider* is a pair of links directed such that they meet in a node.) For example, if we find that A and B are dependent, B and C are dependent, but A and C are conditionally independent given S, not containing B, then this can be represented by the structure A –> B <– C.
- Next, directions are enforced for those links whose direction can be derived from the conditional independences found and the colliders identified.
- Finally, the remaining undirected links are directed randomly, ensuring that no directed cycles occur.

One important thing to note about the PC algorithm is that, in general, it will not be able to derive the direction of all the links from data, and thus some links will be directed randomly. This means that the learned structure should be inspected, and if any links seem counterintuitive (e.g., sweat causes fever, instead of the other way around), one might consider going back and insert a constraint specifying the direction of the link, or use the NPC algorithm, which allows the user to interactively decide on the directionality of undirected links.

Traditional constraint-based learning algorithms produce provably correct structures under the assumptions of infinite data sets, perfect tests, and DAG faithfulness (i.e., that the data can be assumed to be simulated from a DAG). In the case of limited data sets, however, these algorithms often derive too many conditional independence statements. Also, they may in some cases leave out important dependence relations.

Generally, it is recommended to use the NPC algorithm, as the resulting graph will be a better map of the (conditional) independence relations represented in the data. In particular, when the data set is small, the NPC algorithm should be the one preferred. The NPC algorithm, however, has longer running times than the PC algorithm.

The NPC Algorithm

NPC stands for *Necessary Path Condition*, and it is a criterion developed by researchers at Siemens in Munich for solving some of the problems of constraint-based learning algorithms like the PC algorithm. The basic machinery is the same in the PC and the NPC algorithms (i.e., they are both based on generating a skeleton derived trough statistical tests for conditional independence).

The NPC algorithm seeks to repair the deficiencies of the PC algorithm, which occur especially in the face of limited data sets. The solution provided by the NPC algorithm is based on inclusion of a criterion known as the *necessary path condition* (page 474). This criterion forms the basis for introducing the notion of *ambiguous regions* (page 474), which in turn provide a language for selecting among sets of inter-dependent uncertain links. The resolution of ambiguous regions is performed in interaction with the user (see the next page of the Learning Wizard).

In constraint-based learning algorithms, the skeleton of the graph is constructed by not including a link in the induced graph whenever the corresponding nodes are found to be conditional independent. There can, however, be inconsistencies among the set of conditional independence and dependence statements (CIDs) derived from limited data sets. That is, not all CIDs can be represented simultaneously. The inconsistencies are assumed to stem solely from sampling noise; i.e., we still assume that there exists a perfect independence map of the (unknown) probability distribution from which the data is generated.

The number of inconsistencies in the set of CIDs reflects structure model uncertainty. Thus, the number of uncertainties is a confidence measure for the learned structure and can as such be used as an indication of whether or not sufficient data has been used to perform the learning. The inconsistent CIDs produce multiple solutions when inducing a directed acyclic graph (DAG) from them. These solutions differ with respect to the set of links included.

To resolve the inconsistencies, the NPC algorithm relies on user interaction where the user gets the opportunity to decide on directionality of undirected links and to resolve the *ambiguous regions* (page 474).

The Necessary Path Condition

Informally, the necessary path condition says that in order for two variables X and Y to be independent conditional on a set S, with no proper subset of S for which this holds, there must exist a path between X and every Z in S (not crossing Y) and between Y and every Z in S (not crossing X). Otherwise, the inclusion of Z in S is unexplained. Thus, in order for an independence statement to be valid, a number of links are required to be present in the graph.

Ambiguous Regions

When the absence of a link, a, depends on the presence of another link, b, and vice versa, we define a and b to be inter-dependent. Both a and b are constitute what we call *uncertain links*. An *ambiguous region* is a maximal set of inter-dependent links. That is, an ambiguous region consists of a set of uncertain links.

The main goal is to obtain as few and small ambiguous regions as possible. It should be noted that deterministic relations between variables will also produce ambiguous regions.

The user is offered the possibility of providing information as to how the ambiguous regions should be resolved. The next page of the Learning Wizard will provide an intuitive graphical interface for this interaction.

References

The interested reader is referred to the literature for a more detailed description of the notions behind the NPC algorithm. See e.g. * Steck, H. (2001). *Constrained-Based Structure Learning in Bayesian Networks Using Finite Data Sets,* PhD Thesis, Institut für der Informatik der Technischen Universität München. * Steck, H. and Tresp, V. (1999). Bayesian Belief Networks for Data Mining, Proceedings of *The 2nd Workshop on Data Mining und Data Warehousing als Grundlage Moderner Entschidungsunterstuezender Systeme,* DWDW99, Sammelband, Universität Magdeburg, September 1999. * Steck, H., Hofmann, R., and Tresp, V. (1999). *Concept for the PRONEL Learning Algorithm,* Siemens AG, Munich.

The Greedy Search-And-Score Algorithm

The greedy search-and-score algorithm for learning the structure of a Bayesian network uses a score function to evaluate the *goodness* of a candidate network structure. The algorithm performs a search through the space of possible network structures and returns the structure with the highest score. The operators used to perform the search given a current candidate are add an arc, remove an arc and reverse an arc.

The candidate network structure is provided with a maximum likelihood estimate of the conditional probability tables associated with the nodes of the network in order to compute the score of the candidate structure. If the data is incomplete, the EM algorithm must be used. However, if the data is complete, this can be done by counting cases and normalizing the counts.

For simplicity, we assume that the data is complete (by ignoring the missing data).

We also assume that the score function is *decomposable*. This means that the total score can be computed as a sum of scores of the nodes in the network. This makes it easy to compute the effect of a simple modification to the network (such as adding or removing an edge).

The user can chouse between using the *Akaike Information Criterion* (AIC) or the *Bayesian Information Criterion* (BIC) as the score function.

The user can specify an upper limit on the number of parents of each node in the network structure.

Maximum Number of Parent

The user has the option to specify an upper limit on the number of parents in the network structure.

Chow-Liu Tree Based Algorithms

The Chow-Liu Tree Algorithm

The Chow-Liu algorithm is a structure learning algorithm that findsthe best tree representation of the joint distribution induced bythe data.

A Chow-Liu tree is the *best possible* tree-shaped network approximation to a probability distribution such that all edges are directed away from the root. The quality of the approximation is measured using the Kullback-Leibler distance between the true distribution and the distribution defined by the Chow-Liu tree. When we learn from data, the *true* distribution is defined by the frequencies of the observations. Chow and Liu (1968) show that the optimum tree can be found as the maximum-weight spanning tree over all variables, where the weight of each edge is given as the mutual information between the variables connected by the edge.

A Chow-Liu tree can be constructed as follows: * Calculate the mutual information MI(Xi, Xj) for each pair (Xi, Xj). * Consider the complete MI-weighted graph: the complete undirected graph over {X,. . . ,Xn}, where the links (Xi, Xj) have the weight MI(Xi, Xj). * Build a maximal-weight spanning tree for the complete MI-weighted graph. * Direct the resulting tree by choosing any variable as a root and setting the directions of the links to be outward from it.

The Rebane-Pearl Polytree Algorithm

The Rebane-Pearl polytree algorithm constructs a polytree-shaped belief network approximation to a probability distribution (Rebane and Pearl, 1987).

The Rebane-Pearl algorithm is similar to the *Chow-Liu tree algorithm* (page 254) for constructing the "best possible" tree-shaped belief network approximation to a probability distribution such that all edges are directed away from the root. Instead of constructing a tree-shaped belief network approximation, the algorithm constructs a polytree-shaped belief network.

Rebane-Pearl polytree recovery algorithm: The Chow-Liu algorithm constructs a simple tree structure (each node has one parent — except the root). The Rebane-Pearl algorithm turns this tree into a polytree (each node can have multiple parents, but the underlying network structure is still a tree).

The Tree Augmented Naive Bayes Algorithm

The Tree Augmented Naive (TAN) Bayes algorithm is based on theChow-Liu algorithm.

The TAN algorithm is useful for constructing classification networks where a specific node of the model is target of the reasoning. The target node is used to construct a *conditional* Chow-Liu tree (i.e., a Chow-Liu tree over all nodes except the selected *target*) with the selected root as *root* of the tree. Weights are defined as conditional mutual information (conditional on the target), and all nodes (except target) have *target* as an extra parent.

References * C. K. Chow, C. N. Liu (1968). Approximating DiscreteProbability Distributions with Dependence Trees. * G. Rebane, J. Pearl, (1987). The recovery of causal polytreesfrom statistical data.

CHAPTER

FIVE

RELEASES

5.1 Release 6.0

Version 6.0 of HUGIN Expert's Bayesian network environment was released early November 2001.

The new release consists of a heavily improved HUGIN Decision Engine and a completely re-engineered graphical user interface (GUI). The new GUI runs on a range of different platforms, as it is written in Java. Moreover, the GUI contains a number of new features, including support for object-oriented construction of Bayesian networks, supporting network construction with repeated changes of level of abstraction (i.e., top-down construction, bottom-up construction, or a mix of the two). The handling of tables has also been improved, as multiple tables can be displayed simultaneously and the display of tables can be turned on and off. The new GUI also provides a powerful zooming capability, as the display of networks and monitor windows for discrete nodes can be scaled arbitrarily.

5.2 Release 6.1

On April 15th, 2002 - we have released new versions of all HUGIN products. The new version of the HUGIN Graphical User Interface is 6.1 and the new version of the HUGIN Decision Engine is v5.4.

The main features of this release are: The introduction of a *Learning Wizard* (page 267) including database connectivity in the Graphical User Interface, optimal triangulation and faster inference, adaptation in influence diagrams in the Graphical User Interface, and improved stability of the Graphical User Interface.

The functionality of the Hugin Graphical User Interface has been extended with a Learning Wizard. The *Learning Wizard* (page 267) provides increased support for learning Bayesian networks from data. The Learning Wizard includes support for data acquisition (database connectivity via ODBC, interface to Oracle databases, and plain text files), data pre-processing, specification of domain expert knowledge (i.e., structural constraints and prior distributions), a data dependency slider to detect the relative strengths of links as found in the data, generation of randomised conditional probability distributions, etc.

A new triangulation method has been implemented. This method makes it possible to find a (minimal) triangulation that has the minimum sum of clique state space sizes (i.e., an optimal triangulation). For large networks, this method can improve time complexity of inference by an order of magnitude (sometimes even more), compared to the heuristic methods provided by earlier versions of the HUGIN Decision Engine. The new triangulation method is available in both the HUGIN APIs and in the HUGIN Graphical User Interface.

The computations performed by the HUGIN Decision Engine during the inference process have been reorganized to make better use of the caches in modern CPUs. The result is faster inference.

The functionality of the Graphical User Interface with respect to adaptation has been extended to include adaptation on the chance variables of an influence diagram.

Finally, a large effort has been put into increasing the stability of the HUGIN Graphical User Interface.

5.3 Release 6.2

On January 13th, 2003 - we have released new versions of the HUGIN Graphical User Interface (v6.2) and HUGIN Decision Engine (v6.0).

The main new features of this release are: support for *object-oriented Bayesian networks* (page 346) and influence diagrams in the HUGIN Decision Engine, a new algorithm for learning the structure of a Bayesian network from a database of cases in the HUGIN Graphical User Interface, and various other improvements of the HUGIN Graphical User Interface.

The *PC algorithm* (page 246) for learning the graphical structure of a Bayesian network from a database of cases has been extended with a necessary path condition. This condition is a necessary condition for the existence of a perfect map of the conditional dependence and independence statements derived by statistical tests. This has produced a new algorithm, which will be referred to as the *NPC algorithm* (page 246). The NPC algorithm has a number of advantages over the PC algorithm. For instance, the user now has the possibility to interact with the learning process in order to resolve structural uncertainties by making decisions on the presence or absence of uncertain edges. Both the PC and the NPC algorithm now supports user decisions on the directionality of edges, which cannot be directed using data alone. In the HUGIN Graphical User Interface it is now possible to learn arbitrarily complex structures, as tables for the conditional probability distributions are not created as part of the structural learning.

Various other aspects of the HUGIN Graphical User Interface have been improved: node tables, belief monitors, and operations on sets of nodes.

The node table has been greatly improved. For instance, expressions and manually specified tables can now coexist, cut-and-paste functionality has been improved (e.g., it is possible to cut and paste to/from Excel), tables can be exported to text files, it is possible to work with bars instead of numbers and to have numbers and bars simultaneously, the functionality for printing a table has been improved, and other new functionalities have been added to the node table.

It is possible to specify different display mode for the belief monitors in runmode. This facilitates viewing the changes in beliefs induced by a propagation of evidence.

It is now possible to perform an increased number of operations on sets of nodes (i.e., set type and set interface of sets of nodes simultaneously).

Finally, the HUGIN Decision Engine has been extended with support for object-oriented Bayesian networks and influence diagrams to make this functionality accessible through our Application Programming Interfaces (APIs). This implies that object-oriented Bayesian networks and influence diagrams are now supported both by the HUGIN Decision Engine and by the HUGIN Graphical User Interface.

5.4 Release 6.3

On June 23rd, 2003 - we have released new versions of the HUGIN Graphical User Interface (v6.3) and HUGIN Decision Engine (v6.1).

The main new features of this release are:

- *Data conflict analysis* (page 292)
- *Case files* (page 388)
- Thread-safe HUGIN Decision Engine
- User specified data preprocessing plugins
- Parameter learning wizard
- Various improvements of HUGIN Graphical User Interface

Data conflict analysis (page 292) is the activity of detecting, tracing, and resolving possible conflicts among observations of variable values. The Hugin Graphical User Interface and HUGIN Decision Engine now support identification of potential conflicts in observations and hypothesis driven conflict resolution.

The HUGIN Graphical User Interface and HUGIN Decision Engine now support the use of case files. Case files make it possible for the user to save evidence entered in a domain in a case file for easy reload.

The HUGIN Decision Engine has been extended to support safe usage in a multi-threaded application. It is now possible to safely use the HUGIN Decision Engine in multi-threaded applications where threads process their own local domains or applications where threads access a common domain. In the last scenario, it is necessary to explicitly ensure mutual exclusion.

Support for loading user specified preprocessing plugins have been included in the learning wizards of the HUGIN Graphical User Interface. This enables the user to use her own data preprocessing functionality directly in the HUGIN Graphical User Interface.

The functionality of the HUGIN Graphical User Interface has been extended with an additional Learning Wizard - the EM Learning Wizard. The EM Learning Wizard increases the user support for learning the parameters of Bayesian networks from data. The *EM Learning Wizard* (page 281) includes support for data acquisition (database connectivity via ODBC, interface to Oracle databases, and plain text files), data preprocessing, specification of domain expert knowledge on parameters, generation of randomized conditional probability distributions, etc.

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- Import of network information (groups, colors, node sizes, etc).

This is very useful in documentation of networks, but also for structural learning of Bayesian networks, where loading and saving structural constraints is now supported.

- Curved and piecewise linear arrows between nodes. This makes it possible to avoid having arrows and nodes overlap.
- View the junction tree structure (useful for conflict analysis).
- Individual node sizes
- Improved print functionalities (e.g multi-page networks and selection of printer).

Finally, the heuristic measure used by the HUGIN Decision Engine as part of the total-weight triangulation has been improved to give better results on large networks.

5.4.1 User feedback form

HUGIN Expert A/S has a continuous interest in the opinion of the users of HUGIN software and we wish to improve our software in line with our customers needs. For this reason the HUGIN Graphical User Interface has been extended with a questionnaire, which we kindly ask our users to complete. The user feedback will help us to improve our software and to guide the development of the software and keep it exciting to use. We sincerely hope that many of you will spend a few minutes in helping us improving ourselves.

5.5 Release 6.4

On March 2nd, 2004 - we have released new versions of the HUGIN Graphical User Interface (v6.4) and HUGIN Decision Engine (v6.2).

The main new features of this release are:

- The HUGIN Graphical User Interface now includes a tool for analysing dependence and independence relations between nodes given evidence *(d-separation)* (page 288)
- Automatic update of HUGIN Graphical User Interface (receive and install updates automatically)
- Compressed and password protected hkb-files
- Various improvements of HUGIN Graphical User Interface including a substantial improvement of the speed of the HUGIN Graphical User Interface
- The HUGIN Decision Engine now has support for *EM learning* (page 352) in *object-oriented Bayesian networks* (page 346)

5.5.1 HUGIN Graphical User Interface v6.4:

A tool for performing dependence and independence analysis between nodes of a probabilistic graphical model (i.e. Bayesian network or influence diagram) has been included in the HUGIN Graphical User Interface. This tool enables the user to perform d-separation analysis on sets of variables given selected sets of evidence variables. This tool is particularly useful in the knowledge acquisition phase for validating the dependence and independence properties of the model.

The HUGIN Graphical User Interface now supports live updating. This feature - when enabled - will make the HUGIN Graphical User Interface automatically check for available updates from our web site.

The HUGIN Graphical User Interface now uses compressed hkb-files (HUGIN Knowledge Base files). Furthermore, password protected hkb-files is now an option. Password protection of hkb-files is useful when models are distributed as part of an end-user application.

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- Considerable efforts have been put into improving the speed of the HUGIN Graphical User Interface.
- “Live” node resize. This feature enables the user to resize nodes by dragging.
- It is possible to view the fading, experience, and expression tables for a node at the same time as she is viewing the conditional probability table of the node.
- Parameter learning and adaptation for discrete chance nodes in influence diagrams and mixed Bayesian networks is now supported by the HUGIN Graphical User Interface.
- Simulation of node instantiations (i.e., generation of cases) is now supported for mixed Bayesian networks.
- The HUGIN Graphical User Interface now has improved support for locating nodes in large models. This feature enables the user to search for a node in a large network by its name.
- The user can select to show node names and/or labels in the nodes.
- The user can set the precision used to display the entries of a table.
- It is possible to print a junction forest to a file or a printer.
- Monitor resizing. This feature enables the user to resize *node monitors* (page 194) for better display of results of propagation.

- It is possible to set the number of samples per interval used by the Table Generator when generating the table for a discrete node given parent interval nodes.
- The user can select to view either node state labels or values in conditional probability tables.
- The HUGIN Graphical User Interface now has better support for navigating instance nodes and entering evidence on nodes in instance nodes of a compiled network. The user has the option to transfer evidence from a compiled subnet to a corresponding instance in a different compiled network.

5.5.2 HUGIN Decision Engine v6.2:

The HUGIN Decision Engine now has support for parameter learning in object-oriented Bayesian networks using the EM (Expectation Maximization) algorithm. This feature enables the user to exploit the composition of an object-oriented Bayesian network when estimating conditional probability tables from data.

The HUGIN Decision Engine now uses compressed hkb-files (HUGIN Knowledge Base files). Furthermore, password protected hkb-files is now an option. Password protection of hkb-files is useful when models are distributed as part of an end-user application.

The HUGIN Decision Engine supports parsing a set of nodes from a file. This is, for instance, useful for loading triangulations from a file. The HUGIN Decision Engine also supports parsing a database of cases stored in an ASCII text file. This is useful when learning Bayesian networks from data.

5.6 Release 6.5

On December 7th, 2004 - we are have released new versions of the HUGIN Graphical User Interface (v6.5) and HUGIN Decision Engine (v6.3).

The main new features of this release are:

- The data mining functionality of the HUGIN Graphical User Interface has been improved with a number of new features.
- Improved support for object-oriented models in the HUGIN Decision Engine and HUGIN Graphical User Interface.
- Various improvements to the HUGIN Graphical User Interface including new features such as network layout, network documentation, table import and export online drag & drop help functionality to name a few.

5.6.1 Hugin Graphical User Interface v6.5:

The data mining functionality of the HUGIN Graphical User Interface has been improved with a number of new features:

- The Learning Wizard of the HUGIN Graphical User Interface now includes a tool for determining the set of minimal resolutions of ambiguous regions. This tool has been added because the *NPC algorithm* (page 246) for learning the structure of a Bayesian network model may produce a set of ambiguous regions indicating structure uncertainties. This tool is very helpful when an ambiguous region includes a large number of edges.
- An algorithm for graph layout has been included in the *Learning Wizard* (page 267). This functionality is particularly useful after structure learning. If the user has installed the program called “dot”, the user interface may be parameterized to use this program instead of its internal algorithm.
- A data analysis tool has been included in the data acquisition step of the Learning Wizard. This tool enables the user to see the values recorded in the data for each variable. This tool is particularly useful when continuous variables are included in the data as it also supports discretization.

Support functionality for object-oriented models in the HUGIN Graphical User Interface has been extended to include a number of new features:

- A new file name suffix (".oobn") has been introduced in order to better distinguish network files specifying an object-oriented model from network files specifying a flat model.
- The class of an instance node can easily be replaced by another class with an equivalent interface.
- It is now possible to rotate the interface of an instance node. This functionality may be useful when building time-sliced models.

The HUGIN Graphical User Interface has been extended with a number of features supporting the model development process: * Better display of the network log information and a separate usage log window. * Automatic documentation of models. This feature enables the user to produce HTML documentation of a network, including the descriptions included in the network. * A print preview has been included in the *printing functionality* (page 168). Online help has been improved to support drag'N'drop.

The HUGIN Graphical User Interface has been improved with various new features. This includes: * An *arc* (page 149) between two continuous variables can be reversed. * It is possible to perform a node absorption operation on discrete chance nodes. This enables the user to eliminate nodes from a model without changing the underlying probabilistic relations between the remaining nodes of the model. * It is now possible to both import and export tables to flat text files. Table import is particularly useful when a table has been generated using a different tool. * The graph layout algorithm is also available outside the Learning Wizard. * Finally, efforts have been put into improving the stability of the HUGIN Graphical User Interface.

5.6.2 Hugin Decision Engine v6.3:

The HUGIN Decision Engine has improved support for object-oriented models. The HUGIN Decision Engine has been extended with functionality to replace the class of an instance node with another class with an equivalent interface. In addition, a similar function for replacing a parent of a node with another equivalent parent has been included. This allows the user to substitute classes and parents without loosing the tables of effected nodes.

5.7 Release 6.6

On September 12th, 2005 - we have released new versions of the HUGIN Graphical User Interface (v6.6) and HUGIN Decision Engine (v6.4).

The main new features of this release are:

- Improved data mining functionality of the HUGIN Graphical User Interface.
- *Value of Information analysis* (page 312) on discrete random variables in Bayesian networks and influence diagrams in both the HUGIN Decision Engine and the HUGIN Graphical User Interface.
- Improved performance of the HUGIN Decision Engine with respect to inserting and propagating multiple pieces of hard evidence in the junction forest.
- HUGIN Decision Engine libraries for Windows platforms are now provided for Visual Studio .NET 2003 (in addition to Visual Studio 6.0).

5.7.1 Hugin Graphical User Interface v6.6:

The *Learning Wizard* (page 267) of the HUGIN Graphical User Interface has been improved with a number of new features:

- The Data Analysis tool of the Learning Wizard shows a number of statistics on the data analyzed including an indication of the amount of missing data and the number of distinct values for each variable.
- The Data Dependency window of the Learning Wizard reports the p-value of an edge when the edge is selected and model size when the edge is deselected.
- The Structure Uncertainties window of the Learning Wizard reports the p-value of an edge when the edge is selected.
- The log information generated as part of the learning process is available to the user for inspection after completing the Learning Wizard.

The *EM Learning Wizard* (page 281) of the HUGIN Graphical User Interface has been improved with a number of new features:

- The EM algorithm reports the log-likelihood of the model given data to the network log after each iteration.
- The log information generated as part of the learning process is available to the user for inspection after completing the EM Learning Wizard.

A few features related to the learning functionality of the HUGIN Graphical User Interface have been included or improved:

- Better display of the network log information and a separate usage log window.
- Automatic documentation of models. This feature enables the user to produce HTML documentation of a network, including the descriptions included in the network.
- A print preview has been included in the printing functionality.
- Online help has been improved to support drag'N'drop.

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- A line specifying the data file used for structure learning has been included in the Structure Learning window.
- A line specifying the data file used for parameter learning has been included in the EM Learning window.
- It is now possible to add experience tables to all selected discrete chance nodes.
- It is now possible to add fading tables to all selected discrete chance nodes.
- An invocation of "generate table" is enforced on all nodes with a model prior to parameter learning using the EM Learning Wizard. This ensures that the conditional probability table of each node has been generated form the model and it will be shown in the Prior Distribution window of the EM Learning Wizard.
- The explicit creation of experience tables for all discrete chance nodes has been disabled prior to parameter learning using the EM Learning Wizard. This implies that the user must create experience tables prior to using the EM Learning Wizard or in the Prior Distributions step of the EM Learning Wizard.

The HUGIN Graphical User Interface have been extended with support for value of information analysis on discrete random variables in Bayesian networks and influence diagrams. Value of information analysis is based on computing entropy and mutual information on nodes and pairs of nodes.

The functionality of the HUGIN Graphical User Interface supporting object-oriented models has been extended to include a few new features:

- It is no longer a requirement that object-oriented models are saved with ".oobn" suffix (both .net and .oobn are supported).

- Improved functionality for saving and loading object-oriented models with expanded / collapsed instance nodes. If an instance node is expanded when the model is saved, then the instance node will be expanded and placed at the same location when the model is loaded subsequently.

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- Functionality for loading and saving node elimination orders in the HUGIN Graphical User Interface. This is very useful when a good or even optimal triangulation of a network is available or has been identified.
- Help facilities made available in the "Update HUGIN" window. Due to strict security policies some users have experienced problems using the "live update" functionality of the HUGIN Graphical User Interface. Help facilities have been included in order to clearly explain how to update the installation in case of problems performing the "live-update".
- Improved usage log information.
- Probabilities and utilities are shown as inconsistent when switching from edit mode to run mode on a previously compiled domain when tables have been updated.
- Using right mouse pop-up menu to change the subtype of a node to numbered creates a node with states starting at 0 … (not 1 as was previously the case).
- The functionality for graph Layout now supports continuous chance nodes and instance nodes.

Finally, efforts have been put into improving the stability of the HUGIN Graphical User Interface.

5.7.2 HUGIN Decision Engine v6.4:

The HUGIN Decision Engine has been extended with the following features:

- Functions for computing entropy and mutual information have been added. These functions are useful for value of information analysis.
- Direct access to the pseudo-random number generator implemented in HUGIN Decision Engine is now provided through API functions.
- The EM algorithm reports the log-likelihood of the model given data to the network log after each iteration.
- Improved performance of the HUGIN Decision Engine with respect to inserting and propagating multiple pieces of hard evidence in the junction forest.

Finally, HUGIN Decision Engine libraries for Windows platforms are now provided for Visual Studio .NET 2003 (in addition to Visual Studio 6.0).

5.8 Release 6.7

On June 27th, 2006 - we have released new versions of the HUGIN Graphical User Interface (v6.7) and HUGIN Decision Engine (v6.5).

The main new features of this release are:

- *Sensitivity to Evidence (SE) Analysis* (page 299) on discrete random variables in Bayesian networks and influence diagrams in the HUGIN Graphical User Interface.
- Improved *conflict resolution dialog* (page 292) of the HUGIN Graphical User Interface.
- Improved *value of information dialog* (page 312) of the HUGIN Graphical User Interface.

5.8.1 HUGIN Graphical User Interface v6.7:

The HUGIN Graphical User Interface has been extended with support for sensitivity to evidence (SE) analysis on discrete random variables in Bayesian networks and influence diagrams. SE analysis includes determining minimum and maximum posterior beliefs, impact of evidence analysis, discrimination between competing hypotheses, what-if analysis, and sensitivity to findings:

- Determining minimum and maximum beliefs is useful for analysing the sensitivity of a hypothesis variable relative to an unobserved variable.
- Evidence impact analysis investigates the impact of various subsets of the evidence on a hypothesis by computing the normalized likelihoods of the hypothesis given the evidence.
- Discrimination between competing hypotheses is based on the calculation of Bayes' factor. This analysis supports the identification of subsets of the evidence which discriminates between two hypotheses.
- What-if analysis investigates the impact of changing the value of an observed variable on the posterior distribution of a hypothesis variable.
- Sensitivity to findings analysis analyses the impact a single finding has on the posterior probability of a hypothesis.

The Value of Information analysis dialog of the HUGIN Graphical User Interface has been improved with new features:

- The dialog now gives a graphical representation of the mutual information score between each information variable and the target node.
- The mutual information score between the target and each information variable is compared to the entropy of the target node.
- The precision of the displayed mutual information score is sensitive to the selected precision.

The Conflict Resolution dialog of the HUGIN Graphical User Interface has been improved with a number of new features:

- The dialog now has the option of selecting the set of possible hypothesis variables.
- The dialog gives a graphical representation of the value of the conflict measure after resolution for each possible conflict resolution.
- The precision of the displayed conflict measure score is sensitive to the selected precision.
- It is possible to perform hypothesis driven conflict analysis. This enables the user to investigate the impact of individual findings on the posterior probability of the hypothesis.
- Support for tracing the source of a possible conflict has been improved. The user may compute partial conflicts for all subsets of a selected set of evidence.

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- Simulation of chance variables in Run-Mode. This functionality allows the user to simulate an instantiation of all variables given the inserted evidence.
- The menu items under the “Network” menu have been rearranged. An “Analysis” menu item has been introduced.
- Monitors and node lists are now updated immediately after entering a value on a continuous chance node (as opposed to after the propagation).
- Functionality for reporting the beliefs of a selected node or all nodes to the Network Log has been included.
- The entropy of a discrete chance node is shown in the Usage Log when the node is selected in run-mode.
- The mutual information score between two discrete chance nodes is shown in the Usage Log (in Run Mode only) when selecting their connecting edge.

- Improved support for long menus (e.g. long menus may appear as a result of having loaded a large number of classes).
- *d-separation* (page 288) analysis is now possible for NetworkModels in edit mode.
- Functionality for rearranging node states has been included.
- Functionality for printing monitor windows with the graph of a model has been included.
- It is now possible to include monitor windows when writing a model as BMP.
- It is possible to replace a parent node of a child node without losing the table of the child in the process.

Finally, efforts have been put into improving the stability of the HUGIN Graphical User Interface.

5.8.2 HUGIN Decision Engine v6.5:

The HUGIN Decision Engine has been extended with the following features:

- Functions for computing the AIC and BIC scores have been included. AIC and BIC are scores of comparing model quality taking model complexity into account.
- It is now possible to enter a data case as evidence using a single function. This is, for instance, useful for iterating over all data cases and computing posterior beliefs.
- Functions for getting the state index of a discrete chance or decision node corresponding to a value or a label have been included. This is particularly useful for inserting evidence on a node with interval subtype.
- Functionality for performing *d-separation analysis* (page 288) has been included. This includes two functions for obtaining the nodes that are d-connected and d-separated to a set nodes, respectively, given a set of hard and a set of soft evidence.
- Functionality for cloning nodes and domains has been included.

In addition some minor revisions have been made to existing functionality of the HUGIN Decision Engine:

- The amount of case data which can be handled by the learning algorithms has been doubled (given the same amount of physical memory).
- The HUGIN API now supports simple labels without quotes in case and data files (as opposed to requiring labels always to be quoted).
- The EM algorithm reports the AIC and BIC scores to its log file after completion of parameter estimation.
- Default labels for Boolean nodes have been changed to “false” and “true” in the HUGIN C API and HUGIN ActiveX server.
- The h_domain_get_log_likelihood function (in the HUGIN C API and equivalent functions in other HUGIN APIs) now return the log-likelihood using the actual parameter values (as opposed to using the parameter values of the penultimate iteration when using EM).

5.9 Release 6.8

On April 18th 2007 - we have released new versions of the HUGIN Graphical User Interface (v6.8) and HUGIN Decision Engine (v6.6).

The main new features of this release are:

- A new HUGIN .NET API is introduced with this release
- 64-bit versions of the HUGIN Graphical User Interface and most APIs
- Support for Unicode in the HUGIN software (Graphical User Interface, Java and .NET APIs)
- *A Data Analysis Wizard* (page 259) in the HUGIN Graphical User Interface
- *Value of information analysis* (page 312) on decisions in influence diagrams in the HUGIN Graphical User Interface
- Hypothesis driven conflict analysis in the HUGIN Graphical User Interface

5.9.1 HUGIN Graphical User Interface v6.8:

The HUGIN Graphical User Interface has been extended with a Data Analysis Wizard. The Data Analysis Wizard allows the user to analyse a model by computing AIC and BIC scores given a set of data, illustrating mutual information scores of all links in the graph and data accuracy on a data set. The Data Analysis Wizard also supports generation of data.

The Value of Information dialog of the HUGIN Graphical User Interface has been improved with a new feature:

- *The Value of Information* (page 312) (VOI) Dialog has been extended to include VOI analysis on decisions in influence diagrams. Value of information analysis on a decision variable may help the decision maker to determine if an additional observation on a discrete chance node will improve the maximum expected utility of the decision under consideration.

The Conflict Resolution dialog of the HUGIN Graphical User Interface has been improved with a new feature:

- *The Conflict Analysis Dialog* (page 292) has been extend with support for hypothesis driven conflict analysis on all states or a single state.

The Evidence Sensitivity Analysis dialog of the HUGIN Graphical User Interface has been improved with a new feature:

- *The Evidence Sensitivity Analysis* (page 299) dialog now has a progress bar for calculations that require a large number of propagations. This includes the option to terminate the calculation before all propagations are performed.
- It is possible to discriminate between two hypotheses corresponding to states of the same hypothesis variable.

The HUGIN Graphical User Interface has been extented with functionality for computing the convolution of distributions:

- *A Convolution and XOI loss model* (page 323) dialog for computing total impact based on the exposure, occurrence, and impact model has been included.

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- The HUGIN Graphical User Interface now supports the use of class collections. It is possible to load and save a class collection. A class collection contains a set of class definitions. Class collections allow the user to store multiple class definitions in a single file.
- A new tool, called “View Component Tree”, gives a graphical overview of all classes instantiated in a single network class and its subclasses, etc.

- The HUGIN Graphical User Interface reports the resulting number of states of a chance or decision node when adding states. Also, the number of states of a selected decision node is reported.
- Possibility to view probabilities in node tables with a forced number of decimals instead of scientific notation.
- A new "Case Manager Tool" makes it easy to manage and switch between multiple evidence scenarios in a network. The tool supports loading and saving of multiple cases as a HUGIN data file.
- An option to relax the type checking on bounding links (i.e., a link from a node to an input node) has been included. This allows the user to choose to have the type checking performed at compile time. Default behaviour is to perform type checking when a link is added to the model.
- In the *Learning Wizard* (page 267) a minimal solution to an ambiguous region consists of a number of uncertain edges. If there are multiple minimal solutions to an ambiguous region, the user has to select one of the possible solutions. To guide this selection the weight and p-value of an uncertain edge is displayed in the status bar when the uncertain edge is selected.
- *Monitor windows* (page 194) for continuous chance nodes now support retract evidence.
- GUI response when editing large CPT's have been speeded up.
- A simple tool - *The States Generator* (page 220) - for generating and coping states of discrete nodes has been developed. This tool is particularly useful for specifying states of numbered and interval nodes.
- The continuous-discrete node value transfer function (*CDVT function* (page 320)) implements support for linking a continuous node to a interval chance node. The table for the interval chance node will be generated by the tool from an expression equal to the posterior marginal distribution of the continuous node. This implement a simple method for discrete children of continuous nodes using discretization with support for forward reasoning along the CDVT function link.
- The *"Enter Likelihood Evidence" dialog* (page 141) shows labels or state values depending on the selection of the user in the table of the corresponding node.
- When a numbered discrete chance node is selected in Run-Mode, the status bar will show the average of the node. All state values of the node should be finite.
- The compilation process reports the approximate and compression rate to usage log.
- Unicode support is offered with respect to node and state labels as well as filenames. Node names do not include support for unicode.
- This release include 64-bit versions of the HUGIN Graphical User Interface. The 64-bit version is offered as a separate package with 64-bit versions of the HUGIN C/C++/.NET/Java APIs.

Finally, efforts have been put into improving the stability of the HUGIN Graphical User Interface.

5.9.2 HUGIN Decision Engine v6.6:

The HUGIN Decision Engine has been extended with the following features:

- A new .NET API is now available for the Windows platforms. This API targets the .NET 2.0 framework.
- API libraries for the Windows platforms are now provided for Visual Studio 2005 (in addition to Visual Studio 6.0 and Visual Studio .NET 2003).
- The APIs (except the Visual Basic API) are now available as 64-bit versions on all platforms except Mac OS X. On the Windows platforms, the 64-bit libraries are only provided for Visual Studio 2005.
- A new naming scheme has been introduced for the API libraries on the Windows platforms: The libraries are now uniquely named, making it possible to have all DLLs in the search path simultaneously.
- d-separation analysis is now used to improve the performance of inference. This is particularly useful for incremental propagation of evidence in large networks.

- The performance of the total-weight triangulation method has been greatly improved.
- The triangulation functions now construct junction trees, but do not allocate storage for the data arrays of the clique and separator tables. This permits the application to see the junction trees before attempting the final (and most expensive) part of the compilation process.
- It is now possible to query the size of a junction tree (even before storage is allocated for the junction tree tables).
- A function for testing whether a domain is triangulated is now provided.
- The HUGIN KB file format has changed (in order to handle HKB files produced by 64-bit versions of the API, among other things).

There are a few user-visible changes:

- If a compiled (but not compressed) domain is saved as an HKB file, it will only be triangulated when loaded. A compilation is required before inference can be performed. Compressed domains are still loaded as compressed (which implies compiled), but a propagation is required before beliefs can be retrieved.
- Functions for converting between table indexes and state configurations are now provided.
- A function to retrieve the CG size of a table is provided.

5.10 Release 6.9

On November 5th 2007 - we have released new versions of the HUGIN Graphical User Interface (v6.9) and HUGIN Decision Engine (v6.7).

The main new features of this release are:

- *Parameter Sensitivity Analysis* (page 335) in HUGIN Graphical User Interface and APIs. This includes a Parameter Sensitivity Wizard in the HUGIN Graphical User Interface
- Internationalization of HUGIN Graphical User Interface (distributions include support for English and Japanese languages)
- An *Adaptation Wizard* (page 267) in the HUGIN Graphical User Interface

HUGIN Graphical User Interface v6.9:

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- A *Parameter Sensitivity Analysis Wizard* (page 288). The Parameter Sensitivity Analysis Wizard aids the user in the process of identifying the most influential (conditional probability) parameters of a model and analyzing their effects on the “output” probabilities of the model.
- A *Color Chart* (page 144) is displayed in conjunction with a node table for discrete nodes. The purpose of the color chart is to show where the large values are concentrated.
- The *Learning Wizard* (page 267) now supports the specification of constraints between a node and a subset of nodes.
- The performance of the EM learning algorithm has been improved significantly by saving the initial clique potentials to memory, if possible. This improves the time efficiency of learning CPTs significantly, but increases the memory usage.
- A *“Save to memory”* (page 126) option has been added to *Network Properties* (page 178). The save-to-memory operation stores a copy of the initial clique potentials in memory. This implies a faster initialization process which may improve efficiency of performing multiple propagations (e.g., as part of the Analysis Wizard).

- An *adaptation wizard* (page 267) has been added. The adaptation wizard allows for performing batch adaptation using a number of HUGIN case and data files. Experience counts and posterior marginal distributions can be recorded as they develop during adaptation, and can be plotted as a graph.
- Japanese *language support* (page 200).
- Functionality has been added for computing the joint probability distribution over a set of discrete chance nodes.
- The HTML help pages have been extended with search facilities. In addition, the HTML help pages have been updated with a case and data files tutorial. Also, revisions have been made to the tutorial on adaptation and EM learning.
- *Node* (page 212) and *Domain descriptions* (page 178) support the use of HTML tags.
- It is possible to resize the dialog for *inserting likelihood evidence* (page 141).
- The conflict resolution button is enabled when evidence has been propagated. This allows the user to investigate subsets of the evidence when there is no conflict in the entire set of evidence.
- A JDBC interface has been added to the Wizards supporting database connectivity.
- A Cases/Beliefs panel has been added to the *Analysis Wizard* (page 259). This allows the user to compute and display the posterior probability of a hypothesis for a large number of cases.
- GUI response when working with large networks has been speeded up.
- Belief bars can be forced to render in scientific notation.
- The status bar reports the mean and variance for discrete numerical nodes when selected in run-mode.
- The status bar reports the mean and variance for continuous nodes when selected in run-mode.
- The selection of a continuous chance node reports the size of the node table to the status bar.
- The title of a network window now specifies whether a network is a “net” (i.e., non class model) or a “class” model.

Finally, efforts have been put into improving the stability of the HUGIN Graphical User Interface.

5.10.1 HUGIN Decision Engine v6.7:

The HUGIN Decision Engine has been extended with the following features:

- Sensitivity analysis: Functions to aid in the process of identifying the most influential (conditional probability) parameters of a model and analyzing their effects on the “output” probabilities of the model are now provided.
- New statistical distributions: LogNormal, Triangular, and PERT.
- An optional location parameter has been added to the Gamma, Exponential, and Weibull distributions.
- Truncated continuous distributions can be expressed using a new truncation operator: It is possible to specify double as well as single truncation (single truncation is either left or right truncation).
- It is now possible to combine expressions of different types in conditional expressions. This permits specification of relationships that are sometimes probabilistic and sometimes deterministic.
- The performance of the operation for computing marginals on sets of nodes has been improved.
- The performance of inference has been improved for cases in which a memory backup is not available.

5.11 Release 7.0

On September 8, 2008 - we have released new versions of the HUGIN Graphical User Interface (v7.0) and HUGIN Decision Engine (v7.0).

The main new features of this release are:

- *Limited memory influence diagrams (LIMIDs)* (page 345) replace influence diagrams as the tool for modeling decision problems. The LIMID representation removes two fundamental assumptions of the traditional influence diagram: A non-forgetting decision maker and a total order of the decisions. By removing these two assumptions, the LIMID representation implies a significant improvement in the support for modeling decision problems. In addition to supporting the LIMID representation, the tool has been updated to supply the user with additional information related to a decision model (e.g., show policies and both decision and chance nodes have a probability distribution and an expected utility function determined by the current strategy).
- The new *Expression Builder Wizard* (page 274) makes it easier to build an expression by dividing the process into steps that allow the user to concentrate on building one simple expression at a time while keeping an overview of the complete expression.

HUGIN Graphical User Interface v7.0:

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- The new *constraints panel* (page 134) of the *Parameter Sensitivity Analysis (PSA) Wizard* (page 288) allows the user to specify constraints on the posterior probability of a hypothesis variable and apply the smallest change necessary to satisfy it. The changes can be confirmed in the PSA panel.
- The color map in the PSA panel displays Min/Max parameter sensitivity values instead of parameter values.
- Improved performance of the parameter sensitivity analysis in the PSA Wizard.
- The hypothesis variable drop-down box in the PSA panel marks nodes with evidence as the analysis cannot be performed on instantiated nodes.
- The clear button in the PSA panel now also initializes the network.
- The graph info table in the PSA panel supports the selection of a line description (row) that results in displaying the selected line in the graph. (Useful when performing analysis on all states of the hypothesis.)
- The graph (PSA panel) displays upper and lower bounds for the sensitivity function.
- Display of *Markov Blanket* (page 122).
- The Data Analyzer Tool in the Data Preprocessing panel of the *Learning Wizard* (page 267) now allows the discretization of multiple numerical variables.
- The Learning Wizard and the *EM Learning Wizard* (page 281) have been improved with functionality for reporting different pieces of information relevant for the learning process.
- ROC analysis is now available as a part of the *Analysis Wizard* (page 259)
- The results displayed in the “Cases/Belief”-tab of the Analysis Wizard can be exported as a CSV file.
- A *Correlation Analysis Dialog* (page 329) allows the user to investigate the correlation between any pair of discrete chance nodes in a Bayesian network.
- The Min-Max panel of the *Evidence Sensitivity Dialog* (page 299) shows the results as graphs and numbers instead of as tables of numbers.
- Better support for running the HUGIN Graphical User Interface on Windows Vista.
- Other minor improvements.

Finally, efforts have been put into improving the stability of the HUGIN Graphical User Interface.

5.11.1 HUGIN Decision Engine v7.0:

The HUGIN Decision Engine has been extended with the following features:

- HUGIN API libraries for the Windows platforms are now provided for Visual Studio 2008 (in addition to Visual Studio 6.0, Visual Studio .NET 2003, and Visual Studio 2005).
- The “h_domain_compile” function now initializes the junction trees using the available evidence (if any).
- The “h_class_create_domain” function now assigns unique “meaningful” names to the nodes of the created domains. This is accomplished by allowing “dotted names” for nodes in domains (but not for nodes in classes). This change affects the lexical syntax of node names in files (NET, data, case, and node list files) and in strings containing expressions.
- Sensitivity analysis: It is now possible to compute sensitivity data for multiple output probabilities simultaneously. This allows for easy solution of constraints on the output probabilities.
- **Limited memory influence diagrams (LIMIDs) replace influence diagrams as the tool for modeling decision problems. This has several implications:**
    - There is no “no-forgetting” assumption in a LIMID: All information links must be explicitly represented in the network.
    - A total order of the decisions is no longer required.
    - Evidence specified for a LIMID must satisfy the “free will” condition: A chance node can’t be observed before all decisions in the ancestral set of the chance node have been made. Moreover, only simple instantiations are now permitted as evidence for decision nodes.
    - Computing optimal decisions is now a separate operation (that is, they are no longer computed as part of inference).
    - Decision nodes now have tables. These tables represent decision policies and can be specified in the same way (including the use of models) as conditional probability and utility tables. However, they are usually computed by the “h_domain_update_policies” operation.
    - In LIMIDs, there are no constraints on elimination orders used for triangulation.
    - The overall expected utility of the decision problem is provided by the new “h_domain_get_expected_utility” function.
    - “h_node_get_belief” and “h_node_get_expected_utility” now accept (discrete) chance as well as decision nodes as arguments, and “h_node_get_expected_utility” also accepts utility nodes.
    - The usage conditions and the treatment of information links has changed for d-separation analyses in LIMIDs.
    - The “nodes” argument of “h_domain_get_marginal” may now contain decision as well as chance nodes. Also, decision nodes are no longer required to be instantiated and propagated.
    - The “h_domain_simulate” operation no longer requires decision nodes to be instantiated and (for compiled domains) propagated.
    - The EM algorithm and the “h_domain_get_log_likelihood”, “h_domain_get_AIC”, and “h_domain_get_BIC” functions no longer require decision nodes to be instantiated in all cases of the data set. Instead, the evidence in a given case is only required to be “valid”.
    - Sensitivity analysis: Sensitivity functions where the input parameter is associated with a decision node can now be computed. Also, decision nodes are no longer required to be instantiated and propagated.
    - The arguments of “h_node_get_entropy” and “h_node_get_mutual_information” may now be decision as well as chance nodes. Also, decision nodes are no longer required to be instantiated and propagated.
    - The “h_domain_save_to_memory” operation is disabled for LIMIDs.

  - Calculation of conflict of evidence is not supported for LIMIDs.
  - The NET format has been extended in order to represent policies.
  - The HKB format has changed.

Back

5.12 Release 7.1

Today - March 25th, 2009 - we have released a new version of the HUGIN software (v7.1).

The main new features of this release are:

- Parameter estimation from data for continuous nodes using the *Expectation Maximization (EM) algorithm* (page 352). The HUGIN tool has been extended with functionality to perform parameter estimation for both discrete and continuous nodes in a Bayesian network. The parameter estimation is performed using the EM algorithm supporting missing values in data.
- *Two-way parameter sensitivity analysis* (page 340) has been included in the Parameter Sensitivity Wizard. The Parameter Sensitivity Wizard has been extended with support to perform two-way parameter sensitivity analysis on an output probability given a set of evidence. This allows the user to efficiently investigate the impact of co-varying two parameters on a posterior probability.

5.12.1 HUGIN Graphical User Interface v7.1:

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- Learning Wizard: Functionality for saving and loading data preprocessing definitions, e.g., the specification of a discretization operation or value replacements.
- Learning Wizard: Discretization using Information Entropy Minimization (U. M. Fayyad and K. B. Irani. ‘Multi-Interval discretization of continuous valued attributes’).
- Learning Wizard: Discretization of multiple numeric variables have been improved to become more user friendly.
- *Analysis Wizard* (page 259): Functionality for exporting the results of a series of belief update operations across a set of cases to a CSV file.
- Analysis Wizard: Functionality for generating a PDF report over the results of a series of belief update operations across a set of cases.
- *Parameter Sensitivity Wizard* (page 288): Computing Sensitivity on multiple cases from a case file.
- In addition to parallelizing table operations when compiling and propagating in run-mode. Table operations when learning and in various wizards may also be parallelized.
- Added option for specifying a background image for the *network panel* (page 177).
- Allowing the creation and parsing of invalid Expressions in the *Expression Builder Wizard* (page 274).
- Other minor improvements.

Finally, efforts have been put into improving the performance of the HUGIN Graphical User Interface.

5.12.2 HUGIN Decision Engine v7.1:

The HUGIN Decision Engine has been extended with the following features:

- The EM algorithm has been extended to learn CG distributions for continuous nodes.
- The EM algorithm now permits learning to be enabled/disabled for specific parent configurations: A nonnegative experience count enables learning, while a negative experience count disables learning.
- The EM algorithm now checks that the equilibrium is 'sum' with no evidence incorporated. As a "sanity" check, it also verifies that learning is enabled (that is, there must exist at least one node with a nonnegative experience count), and that case data has been specified.
- Because the EM algorithm is controlled by experience tables, continuous nodes may now also be given experience tables.
- The HKB format has been updated (in order to support parameter learning for continuous nodes).
- The model scoring functions now check that the junction tree potentials are up-to-date with respect to the node tables and their models (if any).

5.13 Release 7.2

November 25, 2009

Today we are releasing a new version of the HUGIN software (v7.2). The main new features of this release are:

- A new Application Programming Interface (API) for PDAs. The HUGIN Decision Engine now has an API that makes it possible to run applications using HUGIN functionality on a PDA.
- HUGIN software has been extended with a new Monte Carlo algorithm to find the most probable configurations of a set of nodes in a network.
- Parameter estimation from data for object-oriented networks using the Expectation Maximization (EM) algorithm is now supported by the HUGIN Graphical User Interface (GUI). The parameter estimation is performed using the EM algorithm supporting missing values in data.

5.13.1 HUGIN Graphical User Interface v7.2

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- Functionality to identify *non-requisite parents of a decision node* (page 291) in a limited-memory inclucence diagrams has been included.
- Value of Information Dialog: now supports *value of information analysis* (page 312) on continuous nodes through a simple approximation.
- *Evidence Sensitivity Analysis Dialog* (page 299): the what-if tab now supports evidence on continuous nodes.
- *Analysis Wizard* (page 259): the Accuracy pane is enabled for networks with both discrete and continuous nodes.
- A new triangulation method has been introduced. This triangulation method is now the default method used by the tool.
- Improved support for automatic update when new builds are released.
- The status bar now displays the size of the policy for a selected decision node.
- The status bar now displays the size of the utility function associated with a selected utility node.
- Functionality to flip the network upside down has been included.

- Other minor improvements.

Finally, efforts have been put into improving the performance of the HUGIN Graphical User Interface.

5.13.2 HUGIN Decision Engine v7.2

The HUGIN Decision Engine has been extended with the following features:

- An algorithm for finding the "requisite" parents of a decision node in a LIMID has been implemented.
- A Monte Carlo algorithm for finding the most probable configurations of a set of nodes has been implemented.
- A new triangulation method has been implemented: This method triangulates each prime component using all of the elimination based triangulation heuristics and uses the best result.
- This triangulation method is now used by the compilation operation when compiling untriangulated domains.
- As the elimination based triangulation heuristics may produce non-minimal triangulations, an extra pass that removes "redundant fill-in edges" has been added to these heuristics.

5.14 Release 7.3

Today, March 29, 2010, we are releasing a new version of the HUGIN software (v7.3). The main new feature of this release is the introduction of a new node type called the "function" node type.

A function node represents a numerical value computed from the results of a propagation in a network. A function node allows the user to define mathematical expressions representing a value computed after a successful propagation (or simulation) in a network. The function node can have continuous nodes, discrete chance nodes, and utility nodes as parents. The expression is defined using the Table Generator.

The development of the functionality concerning the function node type (introduced in HUGIN 7.3) has been sponsored by Danish mortgage credit institution Nykredit Realkredit (www.nykredit.dk).

5.14.1 HUGIN Graphical User Interface v7.3

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- A dialog for determining the number of states of a hidden node has been introduced. The number of states is determined by iterating over all state space sizes from a minimum to a maximum value set by the user. The user can select from a number of different scoring methods. For each iteration a parameter estimation is performed and the user has the option to set the number of states of the hidden node to the best scoring option.
- A dialog to perform feature selection has been introduced. The feature selection is performed relative to a target node. For each node represented in data, a score of the correlation between the node and the selected target node is computed. The result is a list of the nodes sorted according to the score.
- The *Chow-Liu algorithm* (page 254) has been included in the Learning Wizard. The Chow-Liu algorithm allows the user to learn a tree structure over the discrete chance nodes represented in data. The resulting tree is a maximum-likelihood estimate of the unknown distribution generating the data (under the tree structure constraint).
- A *tree-augmented naive Bayes model (TAN)* (page 253) learning algorithm has been included in the Learning Wizard. The TAN learning algorithm is based on the Chow-Liu algorithm. It learns a tree structure over the attribute nodes of a TAN model given a target node.

- The support for Object-Oriented Networks (OONs) has been improved with new functionality for navigating OONs in run-mode. This includes support for opening monitor and policy windows of nodes inside an instance node and traversal of instance nodes.
- Vector Quality PDF export for network structure.
- The adaptation wizard now supports the inspection of probability distributions for nodes not subject to adaptation.
- When loading a case from file the probability of the evidence in the case is reported to the status bar
- Other minor improvements.

Finally, efforts have been put into improving the performance of the HUGIN Graphical User Interface.

5.14.2 HUGIN Decision Engine v7.3

The HUGIN Decision Engine has been extended with the following features:

- A new node type has been introduced: The “function” node type. This node type is used to express calculations to be performed using the results of inference or simulation as input. However, function nodes are not involved in the inference process itself, so evidence cannot be specified for function nodes.
- **Here is a list of changes related to the introduction of function nodes:**
    - The category of function nodes is “h_category_function”, and the kind is “h_kind_other” (this kind is also used for utility and instance nodes).
    - A real-valued function is associated with each function node. This function is specified using a model.
    - The function associated with a function node must only depend on the parents of the node. The parents can be of any type of node (except instance nodes).
    - The value of a function node based on the results of inference is requested by the “h_node_get_value” function.
    - The value of a function node based on the results of simulation is requested by the “h_node_get_sampled_value” function.
    - The “h_domain_new_node”, “h_node_clone”, and “h_node_delete” functions do not uncompile if a function node is involved.
    - The “h_node_add_parent”, “h_node_remove_parent”, and “h_node_switch_parent” functions do not uncompile if the child node is a function node.
    - The HKB format has been updated to support networks with function nodes (but the old format is still used for networks without function nodes).
    - The NET language has been extended: The new “function” keyword denotes the definition of a function node, and “potential” specifications are now also used to specify links and models for function nodes.
    - The results of simulation are now invalidated by (implicit as well as explicit) uncompile operations.
    - The new “h_node_get_sampled_utility” function requests the “sampled” utility of a utility node.

5.15 Release 7.4

October 2010

Today we are releasing a new version of the HUGIN software (v7.4). The main new feature of this release is functionality in the HUGIN Graphical User Interface (GUI) to import the specification of a Bayesian network from a set of primitive tables. In addition, the support for solving influence diagrams has been improved with new features.

The new functionality to import a Bayesian network from a set of primitive tables gives the user an alternative to using the traditional HUGIN Network Specification language for specifying a Bayesian network model.

The development of the functionality concerning importing a Bayesian network from a set of primitive tables (introduced in HUGIN 7.4) has been sponsored by Danish mortgage credit institution NykreditRealkredit (www.nykredit.dk).

5.15.1 HUGIN Graphical User Interface v7.4

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- The JointDialog is enabled for LIMIDs and supports decisions.
- The CorrelationDialog is enabled for LIMIDs and supports decisions.
- It is possible to initialize policies for decision nodes in Run-mode. This enables the user to initialize policies without returning to Edit-mode, for instance, when performing Single Policy Updating under different evidence scenarios.
- It is possible to display the requisite ancestors for decision nodes. This is, for instance, useful for solving a network with decisions and utlity functions as a traditional influence diagram.
- Switching node class (e.g., switching the node class from decision to chance and vice versa).
- The Adaptation Wizard has been expanded with extra recording and logging features.
- Case counts enabled for data files in the Adaptation Wizard.
- The Sensitivity to Evidence Dialog can now be open from the cases/beliefs tab in the Analysis Wizard. This is useful for explaining the results of belief update in the Analysis Wizard.
- A new option to specify additional optimal triangulation parameters has been introduced.
- Support for native printing using lpstat/lpr and PostScript on Unix-like systems (e.g., Linux, Solaris, Mac).
- The speed of resetting, normalizing and randomizing probability, experience and fading tables has been improved.
- Internationalization of HUGIN Graphical User Interface (buttons and menus now include support for English, French and Japanese languages)
- The HUGIN Graphical User Interface has been extended with a Code Wizard to generate C, C# and Java code for runtime models. This enables the user to generate code constructing a model using the corresponding HUGIN API.
- Other minor improvements.

Finally, efforts have been put into improving the performance of the HUGIN Graphical User Interface.

5.15.2 HUGIN Decision Engine v7.3

The HUGIN Decision Engine has been extended with the following features:

- HUGIN API libraries for the Windows platforms are now provided for Visual Studio 2010 (in addition to Visual Studio 6.0, Visual Studio .NET 2003, Visual Studio 2005, and Visual Studio 2008).
- A function for identifying the "requisite ancestors" of a decision node is now provided. This is useful for identifying the relevant past observations and decisions in traditional influence diagrams (i.e., influence diagrams obeying the "no-forgetting" rule).
- It is now possible to convert a decision node to a chance node (or vice versa) – while preserving all other attributes of the node unchanged.
- The compression feature is now supported for LIMIDs.
- The data type used for table indexes within the data structures of compressed domains is now a 32-bit integer type (this used to be a 16-bit type). This change allows the construction of much larger compressed tables (at the cost of a higher memory consumption).
- Because of this, HKB files containing compressed domains use a new revision of the file format (but other HKB files use the same revision as HUGIN API version 7.3).

5.16 Release 7.5

June 2011

Today we are releasing a new version of the HUGIN software (v7.5). The main new features of this release are new algorithms for learning the structure of a Bayesian network from a database of cases in the HUGIN Graphical User Interface (GUI) and a new HUGIN Web Service Application Programming Interface (API) to the HUGIN Decision Engine.

5.16.1 HUGIN Graphical User Interface v7.5

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- The Learning Wizard now includes two new algorithms for learning the structure of a Bayesian network from a database of cases: a greedy search-and-score method and the Pearl & Rebane algorithm. The greedy search-and-score method constructs a Bayesian network by a greedy search through the space of possible network structures. It is possible to use either the BIC or AIC score to score candidate structures and to specify an upper limit on the number of parents of each node in the graph. The Pearl & Rebane algorithm constructs a poly-tree structure from the database of cases.
- The Learning Wizard has a new option to add no-links constraints to a selected set of nodes.
- The Analysis Wizard now supports the use of function nodes. This allows the user to process a large number of cases and evaluate the value of function nodes in each case.
- In the Sensitivity Set Graph of the Parameter Sensitivity Wizard it is now possible to open a table to see the sensitivity values associated with each node. In the Parameter Sensitivity Wizard there is also a new option for saving the sensitivity network panel as PDF.
- The Evidence Sensitivity Dialog has improved support for performing "what-if" analysis.
- The user now has the option to define her own language file.
- The Import Information function now enables the user to import state labels and values from a network specification file.

- When a node of a Class is selected in Run-mode, the status bar shows the entropy for discrete chance nodes, the entropy, mean and variance of discrete numeric chance nodes and the mean and variance of continuous chance nodes.
- The simulation of function nodes has been added to the Generate Simulated Cases dialog.

Finally, efforts have been put into improving the performance of the HUGIN Graphical User Interface.

5.16.2 HUGIN Decision Engine v7.5

The HUGIN Decision Engine has been extended with the following features:

- The performance of table operations involving discrete nodes only has been improved. This has led to improved performance of inference on many networks.
- Function and utility nodes can now be used as inputs and outputs of classes in OOBNs.
- A new HUGIN Web Service API is introduced including a set of pre-defined widgets to support easy deployment of HUGIN models using JavaScript. Examples can be found at: http://www.hugin.com/solutions/fraud-detection-management/online-demonstration and http://camvac.hugin.dk

5.17 Release 7.6

February 2012

Today we are releasing a new version of the HUGIN software (v7.6). The main new features of this release are support for learning the structure of a Bayesian network with both continuous and discrete variables from a database of cases, new functionality in the HUGIN Graphical User Interface (GUI) for reading data and preprocessing data, and a new input panel of the HUGIN GUI.

5.17.1 HUGIN Graphical User Interface v7.6

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- The constraint-based structure learning algorithms, i.e., the PC and NPC algorithms for learning structure, have been extended to support learning the structure of Bayesian networks with both continuous (CG) and discrete variables under the assumption that continuous nodes are conditional linear Gaussian and discrete nodes cannot have continuous parents.
- The functionality of the HUGIN GUI for reading and representing data has been improved significantly. The major changes include the use of a new flexible internal data structure for representing data as well as a completely reengineered data preprocessor for manipulating raw data.
- A new Input Panel has been added to HUGIN GUI Run-Time mode. The Input Panel includes a list of predefined nodes that the user can enter evidence on and the possibility to change the value of function nodes. This, for instance, makes it easy to enter evidence on a set of nodes as well as to change constants of expressions for tables defined as function nodes.
- The new input panel supports the computation of the p-percentile for interval nodes.
- The Value of Information Dialog now supports the calculation of mutual information between all subsets of nodes.
- The Learning Wizard now reports the table size of a selected node. This is helpful for determining if the data set is sufficient to justify the complexity of a network.
- Function and utility nodes can be specified as interface nodes.

- Function nodes can now be specified as parents of non-function nodes. This can be used to provide input to the table generation facility.

Finally, efforts have been put into improving the performance of the HUGIN Graphical User Interface.

5.17.2 HUGIN Decision Engine v7.6

The HUGIN Decision Engine has been extended with the following features:

- The structure learning algorithm now supports learning of networks with continuous (CG) nodes.
- Function nodes can now be specified as parents of non-function nodes. This can be used to provide input to the table generation facility.

5.18 Release 7.7

December 2012

Today we are releasing a new version of the HUGIN software (v7.7). The main new features of this release are the introduction of a new node type called the discrete “function” node type and the new associated operation “aggregate”.

A discrete function node is similar to a function node. Whereas a function node represents a numerical value computed from the results of a propagation in a network, a discrete function node represents a set of numerical values. That is, a discrete function node has a subtype, e.g., interval or labelled, and a set of states. A discrete function node allows the user to define mathematical expressions representing a distribution computed after a successful propagation in a network. The new operator “aggregate” can be used to aggregate two distributions (representing frequency and severity) into a single distribution. The expression is defined using the Table Generator.

It is possible to use function nodes and, for instance, the “probability” operator to connect (otherwise disconnected) sub-graphs of chance nodes. This means that a probability computed in one network can be transferred to another network.

The development of the functionality concerning the discrete function node type as well as the aggregate and probability-transfer operators (introduced in HUGIN 7.7) has been sponsored by the research project “Operational risk in banking and finance”. The project is dedicated to strengthening management of operational risk in the banking and finance sector, including develop Basel II compliant operational risk measurement and management tools in accordance with the Advanced Measurement Approach (AMA). The project is financed by the University of Stavanger and a consortium of Norwegian banks consisting of Sparebank 1 SR-Bank, Sparebank 1 SNN, Sparebank 1 SMN, Sparebanken Hedmark and Sparebank 1 Oslo and Akershus.

5.18.1 HUGIN Graphical User Interface v7.7

The HUGIN Graphical User Interface has been improved with various new features. This includes:

- The discrete function node type as well as the “aggregate” and “probability” operators are supported by the HUGIN GUI.
- The data preprocessor for manipulating raw data in the Learning and EM Learning Wizards has been improved with new features. This includes the possibility to delete columns and entropy-based discretization for supervised discretization.
- The HUGIN GUI now supports simulation in a LIMID under the current strategy.
- It is now possible to define attributes on Domain objects in the HUGIN GUI.

- A Utility Sensitivity Analysis Dialog has been added to the HUGIN GUI. This allows the user to perform sensitivity analysis on the utility parameters of a LIMID. The sensitivity analysis is solving the LIMID for a set of user defined parameter values.
- The Code Wizard has been extended and improved with new functionality. This has improved the performance of the Code Wizard.
- The performance of the "propagate ALL rows" in the DataSet Dialog has been improved significantly.
- The d-separation code in the HUGIN GUI has been replaced to improve performance.
- The mean and variance calculations for numeric nodes has been improved.
- It is possible to inspect the Table Generator expression for a node in Run-Mode using CTRL+right-click.
- The Node States Generator has been extended to support copying states between nodes of different category and identical sub-types (e.g., to copy states between a labelled discrete chance node and a labelled discrete decision node).
- Other minor improvements

Finally, efforts have been put into improving the performance of the HUGIN Graphical User Interface.

5.18.2 HUGIN Decision Engine v7.7

The HUGIN Decision Engine has been extended with the following features:

- A new node type has been introduced: The "discrete function" node type. This node type is a combination of the "discrete" and "function" node types. The function nodes implemented in previous versions of the HUGIN API are now referred to as "real-valued function" nodes.
- A new operator to express "aggregate distributions" has been introduced. An aggregate distribution is the distribution of the sum of a random number of independent identically distributed random variables. This operator can only be used in models for discrete function nodes.
- A new operator to express a probability computed from the beliefs of a discrete node has been introduced. This operator can only be used in models for function nodes.
- More general network topologies can now be specified. It is now possible to have a directed path containing "functional links" between two nodes that are not real-valued function nodes. This feature is essential in order to use the new expression operators.
- **Two new parameters have been introduced for the total-weight triangulation method. These parameters can be used to produce better triangulations as well as speed up the triangulation process.**
  - An "initial triangulation" can be specified: If a good triangulation is already known, then this can be used to speed up the search for a better triangulation.
  - The search for minimal separators can be speeded up by discarding separators larger than a prespecified size. This also makes it possible to handle larger prime components (without having to split them into smaller graphs).
- The NET language has been extended in order to support discrete function nodes and the new expression operators.
- HUGIN API libraries for the Windows platforms are now provided for Visual Studio 2012 (in addition to Visual Studio 6.0, Visual Studio .NET 2003, Visual Studio 2005, Visual Studio 2008, and Visual Studio 2010).
- The HUGIN Web Service API has been extended with new widget types for creating belief bars, doing vector graphic plotting and table editing.

5.19 Release 7.8

May 2013

Today we are releasing a new version of the HUGIN software (v7.8). The main new feature of this release is the introduction of "explanation" facilities in the HUGIN Decision Engine.

The new "explanation" facilities of the HUGIN Decision Engine makes it more efficient and straightforward for the application developer to include explanation features in her own implementations. Explanation is defined as the task of identifying the impact of subsets of the evidence on the probability of an hypothesis.

5.19.1 HUGIN Graphical User Interface v7.8

The HUGIN Graphical User Interface has been improved with various new features. These include:

- The explanation functionality of the Evidence Sensitivity Dialog is now based on API functionality of the HUGIN Decision Engine.
- A Show Decision Utilities Dialog has been added to the HUGIN GUI. This allows the user to compute and see a table of the utilities for a specific decision from which the policy is identified using Single Policy Updating. This functionality can be used to inspect the table of utilities producing a policy for a decision.
- The Table View has been extended with a CG Table Wizard for Continuous Gaussian (CG) nodes. The CG Table Wizard allows the user to define mathematical expressions for the calculation of the mean and variance of a CG node. This functionality is only available in the HUGIN GUI.
- The policy table of a decision node can be copied and pasted to another decision node.
- The Value of Information Dialog now supports the calculation of a normalized mutual information measure between all subsets of nodes.
- The Input Panel of the HUGIN GUI Run-Time mode has been extended with functionality to collapse parts of the panel. This is useful when the panel includes a lot of information, for instance, a high number of levels in an object-oriented network.
- The Data Frame window has been extended with functionality to compute the ROC and confusion matrix for classification models. The ROC and confusion matrix are useful measures for determining the performance of a classifier.
- Two new parameters have been introduced for the total-weight triangulation method. These parameters can be used to produce better triangulations as well as speed up the triangulation process.
- An "initial triangulation" can be specified: If a good triangulation is already known, then this can be used to speed up the search for a better triangulation.
- The search for minimal separators can be sped up by discarding separators larger than a prespecified size. This also makes it possible to handle larger prime components (without having to split them into smaller graphs).
- Functionality to reorder the states of a labelled discrete node has been included. This function also updates tables according to the new order of states.
- Other minor improvements.

Finally, efforts have been made into improving the performance of the HUGIN Graphical User Interface.

5.19.2 HUGIN Decision Engine v7.8

The HUGIN Decision Engine has been extended with the following features:

- A set of functions implementing explanation features. This enables the user to identify the impact of subsets of the evidence on the probability of a hypothesis using a single HUGIN function as well as functions to access the scores and subsets.
- A function to reorder the states of a decision node has been included. This function also updates tables according to the new order of states

5.20 Release 8.0

March 2014

The main new feature of this release is the introduction of Dynamic time-sliced Bayesian Networks.

Functionality to support reasoning about dynamic time-sliced systems using Bayesian networks is introduced as part of version 8.0. This introduces the notions of Dynamic time-sliced Bayesian Networks (DBNs) with a sliding time-window as well as operations on this time-window including belief update in the time window, forecasting beyond the time window and moving the time window forward. DBNs make it possible for the user to represent a dynamic time-sliced system as a model in HUGIN software. A DBN is specified using a new type of node referred to as a temporal clone. A temporal clone of a node is a representation of the corresponding node at the previous time step. Using temporal clones it is possible to specify a transition probability distribution from one time step to the next time step reflecting the dynamics of the (time-sliced) system represented in the model.

5.20.1 HUGIN Graphical User Interface v8.0

The HUGIN Graphical User Interface has been improved with various new features. These include:

- Dynamic Bayesian Networks (DBNs) including a number of supporting operations
- Link groups have been introduced. Link groups are similar to node groups. The user can define a set of link groups where each group has a specific color. This makes it possible for the user to color code links in a network
- A Network Statistics Dialog has been introduction. It shows the number of nodes, edges, parameters et cetera in a network
- It is now possible to define expressions to generate the mean and variance values of CG nodes.
- Other minor improvements.

Finally, efforts have been made into improving the performance of the HUGIN Graphical User Interface.

5.20.2 HUGIN Decision Engine v8.0

The HUGIN Decision Engine has been extended with the following features:

- Dynamic Bayesian Networks (DBNs) including a number of supporting operations. DBNs are available through all Application Programming Interfaces (APIs) except the Web Service API.
- Nodes of type Interval can now have intervals where the lower and upper values are the same (zero width intervals or point intervals). This is useful in cases where it is important to define a point value of an Interval node.
- Support for Visual Studio 2013
- Web Service API refactoring. Internal parts of the Web Service API have been refactored to improve performance of this API.

- **Optimization features introduced in JavaScript for HUGIN web service API include**
  - A script engine is now available inside the HUGIN web service host process,
  - Remote procedure calls to the host process embedded script engine, and
  - Support for batching function calls with side effects into a single HTTP request.

5.21 Release 8.1

October 2014

The main new feature of this release is the introduction of the new HUGIN COM API and extensions to the support for Dynamic time-sliced Bayesian Networks.

The functionality to support reasoning about dynamic time-sliced systems using Bayesian networks introduced as part of version 8.0 has been extended. This includes the support for dynamic Bayesian networks in the HUGIN Webservice API and the option to include continuous chance nodes and function nodes in the networks (these nodes cannot have temporal clones though). Based on multiple customer requests, we are introducing the HUGIN COM API that, for instance, makes it possible to use HUGIN in a 64-bit version of Microsoft Excel.

5.21.1 HUGIN Graphical User Interface v8.1

The HUGIN Graphical User Interface has been improved with various new features. These include:

- The DBN functionality has been extended. Now continuous chance nodes as well as discrete and continuous function nodes are now supported in DBN models with the restriction that they cannot have temporal clones.
- The DBN functionality has been extended. Now continuous chance nodes as well as discrete and continuous function nodes are now supported in DBN models with the restriction that they cannot have temporal clones.
- Functionality for Prediction and Bayesian filtering with DBNs has been included in the Data Frame.
- The Preprocessing tool used in, for instance, the Learning Wizard, has been improved and changed name to Data Processing. Among the many new features is functionality that enables evaluation of prediction and Bayesian filters.
- The Data Frame window has been extended with functionality to include the normalization constant computed during belief update, the logarithm of the normalization constant, and the probability of the evidence when propagating cases.
- It is possible to report beliefs to the Data Frame window.
- Default link groups have been introduced to make it easy to set the color of, for instance, information links in influence diagrams.
- The Code Wizard has been extended to support both DBNs and object-oriented Bayesian networks and influence diagrams.
- Function nodes and utility nodes can now be defined as interface nodes in object-oriented Bayesian networks and influence diagrams.
- Under certain circumstances, the shortcuts could be lost. This has been fixed.
- Other minor improvements.

Finally, work has been done to improve the performance of the HUGIN Graphical User Interface.

5.21.2 HUGIN Decision Engine v8.1

The HUGIN Decision Engine has been extended with the following features:

- HUGIN Webservice API extended with support for object-oriented Bayesian networks and influence diagrams as well as DBNs
- A new COM interface version of the HUGIN API - HUGIN COM is introduced. It replaces the now deprecated ActiveX server and enables both 32- and 64-bit applications to consume the HUGIN COM interface (i.e. HUGIN is now available to 64-bit Excel).
- Responsiveness of web applications built using JavaScript for HUGIN web service API is speeded up by automatic re-fetching the set of HTTP cache items invalidated whenever a HUGIN function with side-effects is invoked.
- The HUGIN Decision Engine now support saving object-oriented Bayesian networks and influence diagrams as well as DBNs in a HKB file.

5.22 Release 8.2

May 2015

The main new feature of this release is functionality for learning the structure of a Bayesian network from data in parallel using the PC algorithm. The parallel implementation is based on the use of threads and requires all data in main memory.

This release also includes a number of improvements made to the Data Frame window for data processing and model performance evaluation.

5.22.1 HUGIN Graphical User Interface v8.2

The HUGIN Graphical User Interface has been improved with various new features. These include:

- The Data Frame window has been improved with a number of new features to support model evaluation. In particular, the Data Frame window has been extended with functionality to calculate the confusion matrix for classification models where the class variable has more than two states, leave-one-out cross validation, the option to specify the value of a real-valued function node in the data set as well as functionality to show multiple curves in the same plot and to plot values against the case index.
- The PC algorithm has been extended with functionality to learn the structure of a Bayesian network from data in parallel using threads. This means the time to learn the structure of a Bayesian network from data can be reduced by taking advantage of multiple cores on the computer.
- The DBN functionality has been extended with a rewind button making it easy to reset a DBN without performing a new compilation of the model.
- The Node Properties dialog has been improved.
- Selected edges are now highlighted in Edit-mode.
- New functionality to illustrate the strengths of dependence relations in a Bayesian network has been included. This is based on computing the pairwise mutual information between connected nodes in the graph.
- Other minor improvements.

Finally, work has been done to improve the performance of the HUGIN Graphical User Interface.

5.22.2 HUGIN Decision Engine v8.2

The HUGIN Decision Engine has been extended with the following features:

- The Table Generator has a new operator to specify the state index of a parent node in an expression. This is particularly useful when a node has parent nodes of subtype Interval.
- The HUGIN Web Service API now supports the use of JSON objects for more efficient communication between a client and the server. This significantly reduces the communication overhead when the server has to perform a large number of HUGIN function calls.
- The HUGIN Decision Engine supports parallel PC structure learning using threads.

5.23 Release 8.3

February 2016

The main new feature of this release is improved support for Dynamic Bayesian networks. This includes an algorithm for approximate belief update in a Dynamic Bayesian network through approximation of the joint probability distribution over the temporal clones and a number of extensions to the Data Frame window for data processing and model evaluation involving data sequences.

5.23.1 HUGIN Graphical User Interface v8.3

The HUGIN Graphical User Interface has been improved with various new features. These include:

- Inference in a Dynamic Bayesian network can now be performed using an approximate algorithm. The approximation is supported for prediction and when moving the time-window. Belief update inside the time window is exact. This means that the approximation is mainly relevant for 1.5 time-step Dynamic Bayesian networks with a large joint probability distribution over the temporal clones. It is possible to select between exact and approximate inference.
- The Data Frame window has been improved with a number of new features to support evaluation of (mainly) Dynamic Bayesian networks on data sequences. This includes functionality to mark a column as sequence identifier when a data file contains more than one sequence. Functionality for propagating cases, manipulating data, and evaluating dynamic models are sensitive to the specification of a sequence identifier. There is, for instance, functionality to transform data sequences into single cases and to inspect the results of sequence classification in more detail.
- It is now possible to change the value of a real-valued function node without parents in Run-mode.
- It is now possible to run the OOBN EM parameter estimation on a Dynamic Bayesian Network. This is useful in combination with the improved support for handling data sequences in the Data Frame window.
- A Structure Learning Wizard has been added to the list of Wizards. This makes it more easy to construct the structure of a Bayesian network without going through the parameter estimation step of the Learning Wizard.
- The Learning Wizard and the Structure Learning Wizard have an optional step for feature selection. This step supports feature selection on discrete variables relative to a specific target variable and is useful when building classification models from data.
- The DBN functionality has been extended with a fast forward button making it easy to advance a DBN a specified number of steps.
- The DBN functionality has been extended with a menu item to open or close all monitors for a node across time-slices.
- It is now possible to expand / collapse recursively a single instance node in Run-Mode.

- The functionality for exporting beliefs to a Data Frame dialog now shows the mean and variance for CG nodes.
- The same Java code is now used to generate both the 32-bit and the 64-bit versions of the HUGIN Graphical User Interface.
- Toggle Data Frame Window from a frame inside the HUGIN GUI Frame to a freely floating window.
- Other minor improvements.

Finally, work has been done to improve the performance of the HUGIN Graphical User Interface.

5.23.2 HUGIN Decision Engine v8.3

The HUGIN Decision Engine has been extended with the following features:

- The Application Programming Interfaces (APIs) for the HUGIN Decision Engine has been extended with a new data type for representing a data set. This includes supporting functionality to manipulate and use the data set. This is useful for reading, storing and manipulating data that does not necessarily match the requirements of existing API functions for, for instance, parsing data from a file.
- The HUGIN Decision Engine supports approximate inference in Dynamic Bayesian Networks. The approximation is supported for prediction and when moving the time-window. Belief update inside the time window is exact.
- The HUGIN Web Service API has new widgets for the deployment of Dynamic Bayesian Networks.

5.24 Release 8.4

October 2016

The main new feature of this release is improved support for approximate inference in Dynamic Bayesian networks. Approximate inference using the Boyen-Koller approach can now be done between time slices within the time window of the Dynamic Bayesian network. Previously, approximate inference was only supported when advancing the time window and when making predictions beyond the time window. Furthermore, additional functionality to support the evaluation of Dynamic Bayesian networks using data sequence has been added to the Date Frame window.

5.24.1 HUGIN Graphical User Interface v8.4

The HUGIN Graphical User Interface has been improved with various new features. These include:

- Support for Boyen-Koller approximation within the time window of a Dynamic Bayesian Network
- The Data Frame window has been improved with a number of new features to support evaluation of (mainly) Dynamic Bayesian networks on data sequences. It is, for instance, now possible to manually specify “State to Column mappings” and it has been made easier to identify columns containing class probabilities.
- The Data Frame window function ‘propagate-all’ shows a summary window after completion making it easier to identify errors during propagation.
- The list of windows now contains information on Data Frames making it easier to select Data Frame windows.
- It is possible to collapse/expand individual instance nodes in the node list of an OOBN model.
- The “Networks Statistics” dialog has been extended with new information on parents and states.
- Descriptive text labels can now be added to a network using the new text label tool.
- Other minor improvements.

Finally, work has been done to improve the performance of the HUGIN Graphical User Interface.

5.24.2 HUGIN Decision Engine v8.4

The HUGIN Decision Engine has been extended with the following features:

- The HUGIN Decision Engine now supports Boyen-Koller approximation within the time window of a Dynamic Bayesian Network.
- Support for Visual Studio 2015.
- Using the HUGIN Web Service API, the programmer can now specify a filename suggestion to the browsers 'save as' dialog when serving a HKB file.

5.25 Release 8.5

May 17, 2017

The main new features of this release are support for adaptation of class parameters in Object-Oriented Bayesian Networks including Dynamic Bayesian Networks as a special case. HUGIN now supports parameter adaptation in network classes using either fractional update or Online Expectation-Maximization (EM). Previously, the parameter adaptation in case of Object-Oriented Bayesian Networks was performed in the tables of the run-time instance. Now the adaptation is performed in the network class tables. Furthermore, a new HUGIN Decision Engine Application Programming Interface for the Python programming language is introduced in this release.

5.25.1 HUGIN Graphical User Interface v8.5

The HUGIN Graphical User Interface has been improved with various new features. These include:

- Parameter adaptation in network classes using either fractional update or Online Expectation-Maximization (EM) including supporting functionality in the Data Frame window.
- In the Learning Wizard and the Structure Learning Wizard, it is now possible to specify an initial structure of the graph prior to running the greedy search-and-score structure learning algorithm.
- A new Distance Analysis dialog is introduced. This is a tool to compute the distance between the (conditional) probability tables defined in two different models over the same set of nodes. The tool supports Hellinger distance and weighted Hellinger distance as measures of the distance between the probability distributions of two models.
- The feature selection functionality of the Learning Wizard and the Structure Learning Wizard has been improved.
- It is now possible to copy state labels from a numeric node to all numeric nodes without a state label.
- Other minor improvements.

Finally, work has been done to improve the performance of the HUGIN Graphical User Interface.

5.25.2 HUGIN Decision Engine v8.5

The HUGIN Decision Engine has been extended with the following features:

- The HUGIN Decision Engine now supports two new methods for parameter adaptation in Object-Oriented Bayesian Networks including Dynamic Bayesian Networks as a special case.
- A new HUGIN Decision Engine Application Programming Interface for the Python programming language.
- Support for Visual Studio 2017

5.26 Release 8.6

March 21, 2018

The main new features of this release are HUGIN Decision Engine Application Programming Interfaces for Android and iOS platforms.

5.26.1 HUGIN Graphical User Interface v8.6

The HUGIN Graphical User Interface has been improved with various new features. These include:

- The HUGIN Graphical User Interface now has a Derby SQL database interface for loading data cases into a Data Frame.
- Other minor improvements.
- Finally, work has been done to improve the performance of the HUGIN Graphical User Interface.

5.26.2 HUGIN Decision Engine v8.6

The HUGIN Decision Engine has been extended with the following features:

- A new HUGIN Decision Engine Application Programming Interface for the Swift programming language including support for the Apple iOS platform.
- A new HUGIN Decision Engine Application Programming Interface for the Android platform.
- Support for creation of temporal clones of Continuous Gaussian (CG) nodes in Dynamic Bayesian Networks (DBNs). This can, for instance, be used to create a Kalman Filter model as well as other models with a mixture of discrete and continuous nodes. Both exact inference and partial Boyen-Koller approximation are available for models with temporal clones of CG nodes.

5.26.3 Tools introduced with HUGIN v8.6

This release introduces a new tool to integrate Bayesian Belief Networks (BBNs) and geographic information system (GIS).

5.26.4 HUGIN QGIS BBN Plugin

The QGIS Processing script makes it possible to apply a Bayesian network to each point in one or more raster layers and output a new raster layer with the results. For each point potentially covering multiple layers, the Bayesian network inserts and propagates the values of the nodes linked to raster layers. The user can select a number of output values: index of state with maximum probability, probability of state with maximum probability, and maximum expected utility.

5.27 Release 8.7

December 6, 2018

The main new features of this release are support for temporal clones of continuous variables in dynamic Bayesian networks and a Decision Sensitivity to Evidence dialog. Furthermore, a number of new features of the HUGIN Decision Engine are introduced in this release.

5.27.1 HUGIN Graphical User Interface v8.7

The HUGIN Graphical User Interface has been improved with various new features. These include:

- It is now possible to create a temporal clone of a continuous chance node in dynamic Bayesian networks (DBNs). This means that a DBN may include both discrete and continuous chance nodes as well as have temporal clones of each category of node. In the case of mixed discrete and continuous temporal clones, there are restrictions on the compilation of the model.
- The HUGIN Graphical User Interface has been extended with a Decision Sensitivity to Evidence dialog. This dialog enables the user to investigate the impact of changing the value of an observed chance node on the maximum expected utility of a decision node. This is useful for identifying the most and least important observations made prior to making a decision.
- It is now possible in the Preference Pane of the HUGIN Graphical User Interface to deselect the use of fixed font size. This is useful when running HUGIN Graphical User Interface on a high resolution small screen.
- The support for using the mouse wheel to zoom has been improved.
- In the Conflict Analysis dialog it is now possible to specify a maximum subset size on the evidence when performing partial conflict analysis to reduce the computational cost.
- In the Conflict Analysis dialog it is now possible to investigate the contribution of each individual finding to a conflict.

Finally, work has been done to improve the performance of the HUGIN Graphical User Interface.

5.27.2 HUGIN Decision Engine v8.7

The HUGIN Decision Engine has been extended with the following features:

- The HUGIN Decision Engine now supports three new methods for learning the structure of restricted Bayesian networks from data. The algorithms make it possible to learn the structure of a Rebane-Pearl polytree, a Tree-Augmented Naive Bayes Model and a Chow-Liu tree from data.
- The HUGIN Decision Engine Application Programming Interface for the Python programming language has been updated to use double precision for floating-point computations.
- The run-time performance of the algorithm for optimal triangulation has been improved.

5.28 Release 8.8

September 16, 2019

The main new features of this release are a new Application Programming Interface for the .NET core programming language and a number of interesting features in the HUGIN Graphical User Interface.

5.28.1 HUGIN Graphical User Interface v8.8

The HUGIN Graphical User Interface has been improved with various new features. These include:

- The implementation of the batch propagation functionality of the Data Frame has been refactored. This includes the addition of a set of new measures that can be computed after the propagation of a case, e.g., mean, median and quantile of a numeric node as well as state index of a state with maximum probability. This is useful for processing, for instance, GIS data in HUGIN Graphical User Interface.
- Support for generating vector based graphics of models.
- New and improved HTML help facilities.
- Other minor improvements.

Finally, work has been done to improve the performance of the HUGIN Graphical User Interface.

5.28.2 HUGIN Decision Engine v8.8

The HUGIN Decision Engine has been extended with the following features:

- A new HUGIN Decision Engine Application Programming Interface for the .Net core programming language.
- The HUGIN Decision Engine now supports computing the variance of utility nodes.
- Support for Visual Studio 2019.

5.29 Release 8.9

June 2, 2020

The main improvement of the HUGIN software in version 8.9 is the migration of the HUGIN Graphical User Interface to OpenJDK and associated updates.

5.29.1 HUGIN Graphical User Interface v8.9

The HUGIN Graphical User Interface has been improved with various new features. These include:

- Migration to OpenJDK including new installers
- Improvements to the Data Frame functionality
- Support for data conflict analysis – monitor individual contributions to the conflict for discrete nodes through the log-likelihood ratio
- The sorting functionality is improved to sort numerically
- It is possible to include “evidence” on function nodes when processing cases
- Function to reset the contents of the table for a CG node

- Monitor window of a utility node includes variance (optional)
- Monitor window of a numeric decision node includes mean and variance (optional)
- Use of new library to generate vector based graphics of models
- Other minor improvements.

Finally, work has been done to improve the performance of the HUGIN Graphical User Interface.

5.29.2 HUGIN Decision Engine v8.9

The HUGIN Decision Engine has been extended with the following features:

- NET specifications in the form of text strings can now be generated for domains, classes, and class collections. These strings are suitable for, e.g., transmission over a network, and they can be parsed by the usual functions for parsing NET specifications.
- The documentation for the HUGIN Swift API is now available as a set of HTML pages.

5.30 Release 9.0

April 27, 2021

The main improvement of the HUGIN software in version 9.0 is a new Look & Feel experience of the HUGIN Graphical User Interface and associated updates. In addition, users may experience a smoother installation experience on the Windows and macOS operating systems.

5.30.1 HUGIN Graphical User Interface v9.0

The HUGIN Graphical User Interface has been improved with various new features. These include:

- **New Look & Feel implemented across all supported platforms (Linux, macOS, and Windows)**
    - Drag & Drop to open a file
    - Improved PDF export
    - The layout of the menus and the toolbar has been refactored
    - Dark and white themes are now available for the GUI. Themes can be swapped in the preferences menu.
    - Node edges are significantly more smooth
    - New modern icons that are bigger and transparent to support the new themes and larger resolutions
- **Improvements to the Data Frame functionality**
    - Menu item to create a new empty DataFrame
    - Double-click on a row also takes values of function nodes in the corresponding case into consideration
    - New Moment Statistics Dialog
    - New Statistical Measures of Error Dialog
    - The implementation of the batch propagation functionality of the Data Frame has been improved. This includes the addition of a set of new measures that can be computed after the propagation of a case, e.g., variance and standard deviation of a numeric node.

- Store size of instance nodes and relative positions of interface nodes
- Feature Selection Analyzer now supports continuous nodes as target
- Tables are not created for chance nodes when leaving the Structure Learning Wizard
- macOS: The Hugin GUI application is now provided in two separate packages — with native support for M1 and Intel Macs, respectively.
- Other minor improvements

Finally, work has been done to improve the performance of the HUGIN Graphical User Interface.

5.30.2 HUGIN Decision Engine v9.0

The HUGIN Decision Engine has been extended with the following features:

- CSV representations of data sets can now be generated as text strings. These strings can be parsed by the usual functions for parsing DataSet specifications.
- A new “variance” operator is available for use in expressions in models for function nodes
- The Hugin Web Service API now supports OpenJDK
- The HDE for macOS is now universal — with native support for the ARM64 and x86_64 architectures.

5.31 Release 9.1

On December 14, 2021: New Release – HUGIN 9.1

The main improvement of HUGIN software in version 9.1 is the option to toggle between nodes and monitors in Run Mode of the HUGIN Graphical User Interface. In addition, a new structure learning algorithm has been added to the HUGIN Graphical User Interface and the HUGIN Decision Engine.

The Windows installation package has improved support for newer versions of Windows Operating Systems.

5.31.1 HUGIN Graphical User Interface v. 9.1

The HUGIN Graphical User Interface has been improved with various new features. These include:

- It is now possible to toggle between nodes and monitors in Run Mode
- The new Look & Feel has been extended to with additional themes
- The Learning Wizard has been extended with a Hierarchical Naïve Bayes algorithm for structure learning
- Improved printing support for large CPTs and expressions
- The contents of node selection boxes are now sorted for easy selection
- Other minor improvements

Finally, work has been done to improve the performance of the HUGIN Graphical User Interface.

5.31.2 HUGIN Decision Engine v. 9.1

The HUGIN Decision Engine has been extended with the following features:

- A new algorithm for learning structure restricted models, the hierarchical Naïve Bayes algorithm
- Support for Visual Studio 2022

5.32 Release 9.2

On June 30, 2022: New Release – HUGIN 9.2

Today we are releasing a new version of the HUGIN software (v9.2).

The main improvement of HUGIN software in version 9.2 is a new algorithm to fine-tune a Naïve Bayes classifier that has been added to the HUGIN Graphical User Interface and the HUGIN Decision Engine. In addition, a number of improvements have been added to the HUGIN Graphical User Interface such as, for instance, the option to change the font used in conditional probability tables.

Information on the expiration date of the HUGIN Support Pack is now included in the license information.

5.32.1 HUGIN Graphical User Interface v. 9.2

The HUGIN Graphical User Interface has been improved with various new features. These include:

- The Look & Feel themes have been updated
- There is a new dialog to fine-tune a Naïve Bayes classifier
- Option to change the font used to display the content of tables
- More options have been added to change the layout of the monitor windows in influence diagrams
- It is now possible to create temporal close of a selected set of nodes
- Other minor improvements

Finally, work has been done to improve the user experience of the HUGIN Graphical User Interface.

5.32.2 HUGIN Decision Engine v. 9.2

The HUGIN Decision Engine has been extended with the following features:

- A new algorithm to fine-tune a Naïve Bayes classifier
- HUGIN Web Service API has been upgraded to Open JDK v11

5.33 Release 9.3

November 28, 2022: New Release – HUGIN 9.3

Today we are releasing a new version of the HUGIN software (v9.3).

The main new features of HUGIN software in version 9.3 are the options to zoom monitor windows and detaching a monitor window in the HUGIN Graphical User Interface. Furthermore, the algorithm to fine-tune the performance of a classifier has been extended to support the tree-augmented Naïve Bayes structure. This extension is available for both the HUGIN Graphical User Interface and the HUGIN Decision Engine.

5.33.1 HUGIN Graphical User Interface v. 9.3

The HUGIN Graphical User Interface has been improved with various new features. These include:

- The algorithm to fine-tune the performance of a classifier now supports the tree-augmented Naive Bayes structure
- It is now possible to detach a monitor window in Run-Mode
- The monitor windows now support scaling
- Improved support for changing the font size in Table View
- The drawing of edges into monitor windows has been improved
- There is a new function to force the generation of all tables from expressions
- Other minor improvements

Finally, work has been done to improve the user experience of the HUGIN Graphical User Interface.

5.33.2 HUGIN Decision Engine v. 9.3

The HUGIN Decision Engine has been extended with the following features:

- The classifier fine-tuning algorithm has been extended to support tree-augmented Naïve Bayes structure
- The HUGIN Decision Engine now supports computing the probability of an interval of conditional Gaussian (CG) nodes

5.34 Release 9.4

May 15, 2023: New Release – HUGIN 9.4

Today we are releasing a new version of the HUGIN software (v9.4).

The main new features of HUGIN software in version 9.4 are improved text labels in the HUGIN Graphical User Interface and the ability of the Expectation Maximization (EM) algorithm is use threads to process cases in parallel.

5.34.1 HUGIN Graphical User Interface v. 9.4

The HUGIN Graphical User Interface has been improved with various new features. These include:

- The Expectation Maximization (EM) algorithm supports the use of threads to process cases in parallel
- The support for using text labels has been significantly improved
- The function to generate HTML documentation of a model now includes expressions and tables
- A new build update and notification dialog is shown when the HUGIN Graphical User Interface is launched
- It is possible to have a warning when deleting a node
- Other minor improvements

Finally, work has been done to improve the user experience of the HUGIN Graphical User Interface.

5.34.2 HUGIN Decision Engine v. 9.4

The HUGIN Decision Engine has been extended with the following features:

- The Expectation Maximization (EM) algorithm supports the use of threads to process cases in parallel
- If the user changes the table of a CG node before running EM, then the updated table is used as the initial table for the node

CHAPTER

SIX

EXAMPLES

In this section we present some examples of Bayesian networks and (limited memory) influence diagrams.

In the *HUGIN Examples* (page 517) section we present three examples of networks built in HUGIN. These examples are meant to give you an idea of how to construct your Bayesian networks and influence diagrams using the HUGIN Graphical User Interface.

In the *Modeling Examples* (page 525) section we present some problems that can be modeled using Bayesian networks or influence diagrams. These are meant to show the connection between a textual description and a network model.

Implementations of these and many other examples come with the installation of any version of HUGIN. If you have lost them or have an older installation of HUGIN, you have the opportunity to download them[18]

6.1 HUGIN Examples

In this section, we present a series of examples meant to give you ideas on how to construct your Bayesian networks and influence diagrams.

In the *stud farm example* (page 519), a Bayesian network is used to compute the probabilities of the horses in a stud farm being carriers of a recessive gene causing a life threatening desease.

In the *oil wildcatter example* (page 517), an influence diagram is constructed as an aid for an oil wildcatter to decide whether to drill for oil at a location where he has the opportunity to get information from seismic soundings. This example covers the problem of having more than one decision node in an influence diagram.

The *Monty Hall example* (page 523) demonstrates the use of a small Bayesian network to solve the Monty Hall Puzzle.

Implementations of these and other examples come with the installation of any version of HUGIN. If you have lost them or have an older installation of HUGIN, you have the opportunity to download them[19].

6.1.1 The Oil Wildcatter

In this example, an oil wildcatter must decide either to drill or not to drill. He is uncertain whether the hole is dry, wet or soaking in oil. The wildcatter can make seismic soundings that will help determine the geological structure of the site. The soundings will give a closed reflection pattern (indication for much oil), an open pattern (indication for some oil) or a diffuse pattern (almost no hope for oil).

The wildcatter has two decisions to make, namely whether to test with seismic soundings costing K$10 and whether to drill costing K$70. The utility gained from drilling is determined by the state of the hole (dry, wet or soaking).

The domain of this problem is modeled by the (limited-memory) influence diagram (LIMID) in Figure 1. Note the link from Seismic to Drill. It specifies that when the decision of Drill has to be made, the state of Seismic is known.

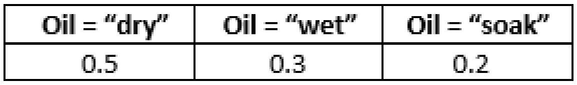
[18] http://download.hugin.com/samples

[19] http://download.hugin.com/samples

If this link is not specified, the computations will be made under the assumption that this observation is not known by the decision maker when she has to make the drill decision.

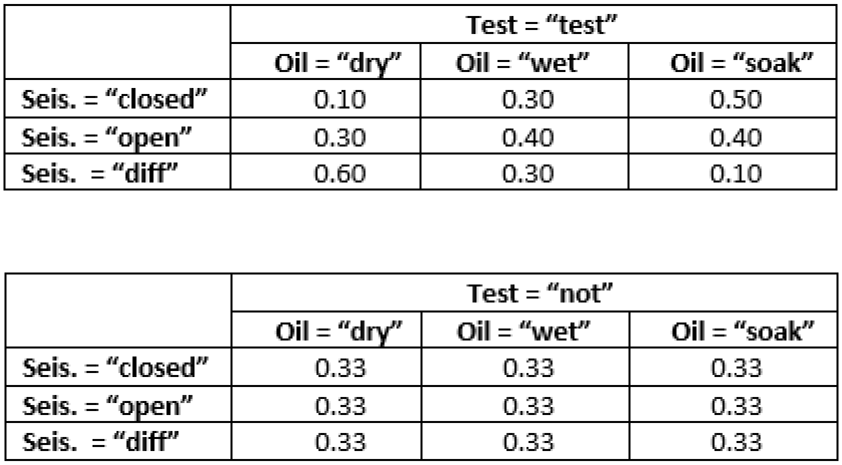
Figure 1: A LIMID modeling the oil wildcatters decision problem.

In Figure 1, the chance node Oil has states “dry”, “wet”, and “soak”, the chance node Seismic has states “closed”, “open”, and “diff”, the decision node Test has actions “test” and “not”, and the decision node Drill has actions “drill” and “not”. In Tables 1 and 2, the CPTs of the nodes Oil and Seismic are shown. In Table 3 and 4, the utility tables of the utility nodes Pay and Cost are shown.



Oil = “dry”	Oil = “wet”	Oil = “soak”
0.5	0.3	0.2

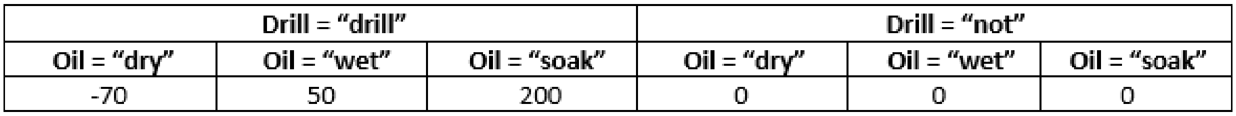
Table 1: P(Oil).



	Test = “test”		
	Oil = “dry”	Oil = “wet”	Oil = “soak”
Seis. = “closed”	0.10	0.30	0.50
Seis. = “open”	0.30	0.40	0.40
Seis. = “diff”	0.60	0.30	0.10

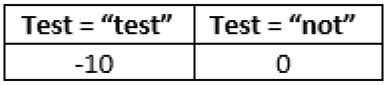
	Test = “not”		
	Oil = “dry”	Oil = “wet”	Oil = “soak”
Seis. = “closed”	0.33	0.33	0.33
Seis. = “open”	0.33	0.33	0.33
Seis. = “diff”	0.33	0.33	0.33

Table 2: P(Seismic | Oil, Test).



Drill = “drill”			Drill = “not”		
Oil = “dry”	Oil = “wet”	Oil = “soak”	Oil = “dry”	Oil = “wet”	Oil = “soak”
-70	50	200	0	0	0

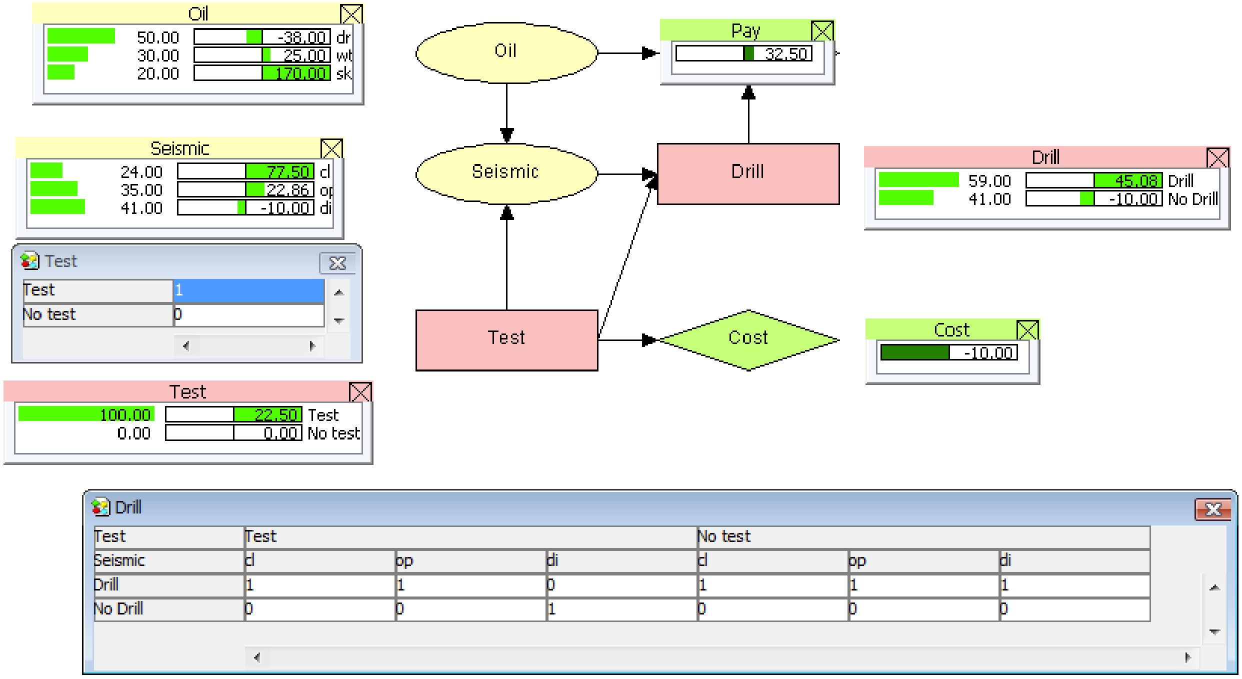
Table 3: Utility function Pay.



Test = “test”	Test = “not”
-10	0

Table 4: Utility function Test.

The influence diagram is now constructed using the HUGIN GUI and after compilation and running Single Policy Update, the results can be seen in the initial state of Figure 2 where the policies for the two decisions are shown along with the monitor windows for all nodes.



Test

Test	1
No test	0

Drill

Test	Test			No test		
Seismic	cl	op	di	cl	op	di
Drill	1	1	0	1	1	1
No Drill	0	0	1	0	0	0

Figure 2: The results of solving the LIMID using Single Policy Update.

Reading the network, we see that the expected utility of testing is k$22.5 while the no testing option as zero probability as it is not the optimal choice and therefore zero expected utility. This implies that it will be best for the wildcatter to test with seismic soundings which will give him the states of Seismic. In the monitor windows you can read the expected utility and probability of different states under the optimal strategy identified by Single Policy Updating. This network has been installed on your computer with the HUGIN software. You can find the network in the Samples subdirectory of your HUGIN installation.

You can also find the samples at the HUGIN download area[20].

6.1.2 The Stud Farm

(Requires minimal knowledge in the area of genealogy)

A Constructed Example from a Stud Farm

The stallion Alan has with the mare Ann sired Betsy and with the mare Alice sired Benny. Betsy has with Bill born Carl, and Benny has with Bonnie sired Cecily. Both Bill and Bonnie are born by Ann, but their fathers (A1 and (A2) are in no way related. Carl and Cecily have just born a colt, Dennis.

[20] http://download.hugin.com/samples

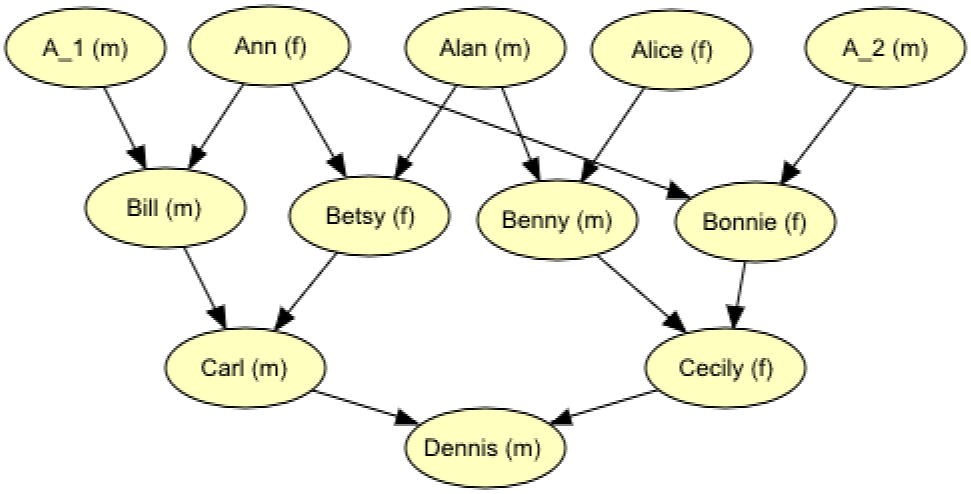


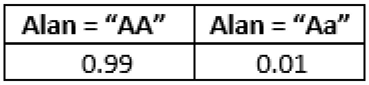
Figure 1: Dennis's genealogy.

It turns out that Dennis suffers from a life threatening hereditary disease carried by a recessive gene *a*. The corresponding dominant gene is *A*. The disease is so serious that Dennis is put down instantly, and as the stud farm wants the gene out of the production, Carl and Cecily are taken out of breeding because they both must be carriers of the gene having genotype *Aa*.

Now the problem is: Which other horses are to be taken out of breeding? Bonnie is a very fine mare, whereas Alan more easily can be replaced in the production. What will the stud farm be best off doing? It would be nice to know the probabilities of each of the horses being carrier of the sick gene. Normally the probability of being carrier is known to be 0.01.

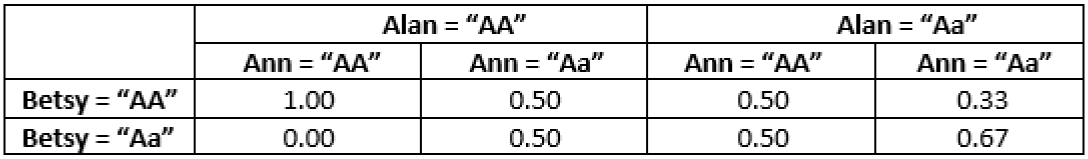
Bayesian Networks

The domain of the inheritance of genes in the stud farm can easily be modeled by a Bayesian network (BN). Actually, the genealogy in Figure 1 only needs a conditional probability table (CPT) on each node to be a BN. First we specify the states of the nodes: All horses except Dennis are either carriers (*Aa*) or not (*AA*) since none of them are sick. We give them states "AA" and "Aa". Each of the nodes in the upper layer in Figure 1 have the CPT shown in Table 1. The others except for Dennis have the CPT shown in Table 2. Dennis has the CPT shown in Table 3.



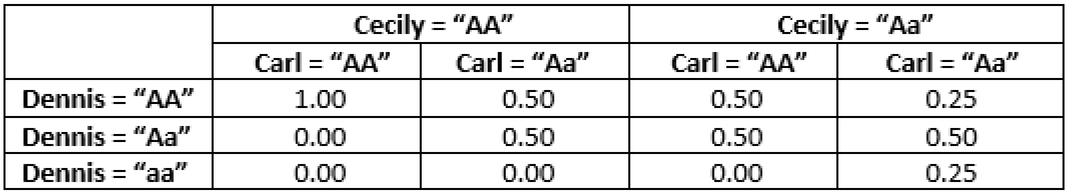
Alan = "AA"	Alan = "Aa"
0.99	0.01

Table 1: CPT of the nodes in the upper layer (Alan used as an example).



	Alan = "AA"		Alan = "Aa"	
	Ann = "AA"	Ann = "Aa"	Ann = "AA"	Ann = "Aa"
Betsy = "AA"	1.00	0.50	0.50	0.33
Betsy = "Aa"	0.00	0.50	0.50	0.67

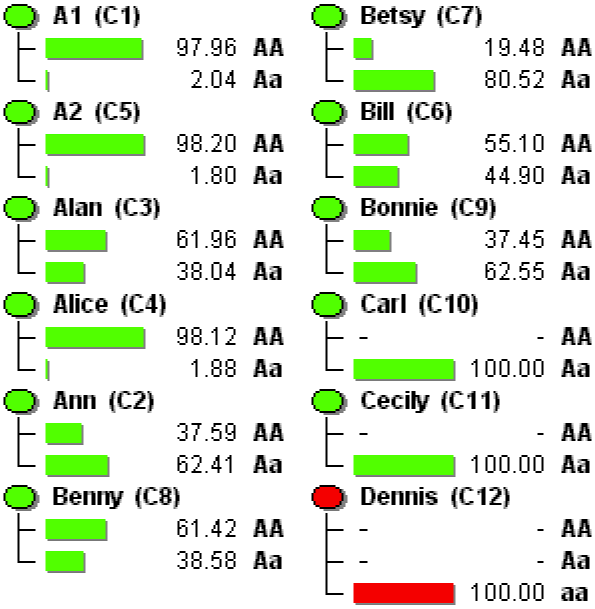
Table 2: CPT of the nodes in the middle layers (Betsy used as an example).



	Cecily = "AA"		Cecily = "Aa"	
	Carl = "AA"	Carl = "Aa"	Carl = "AA"	Carl = "Aa"
Dennis = "AA"	1.00	0.50	0.50	0.25
Dennis = "Aa"	0.00	0.50	0.50	0.50
Dennis = "aa"	0.00	0.00	0.00	0.25

Table 3: CPT of the node Dennis: P(Dennis | Carl, Cecily).

This BN has been implemented using the HUGIN Graphical User Interface in less than half an hour. Then, the evidence that Dennis is aa is entered and sum propagation is performed. The result is shown in Figure 2.

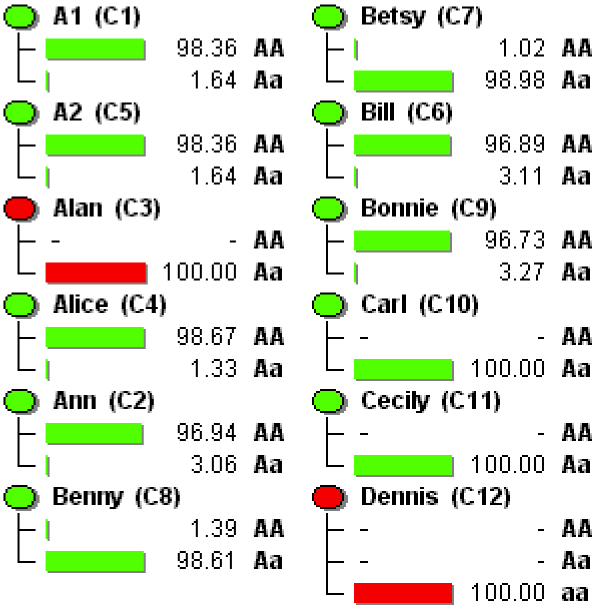


```
A1 (C1)                      Betsy (C7)
        97.96 AA                      19.48 AA
         2.04 Aa                      80.52 Aa
A2 (C5)                      Bill (C6)
        98.20 AA                      55.10 AA
         1.80 Aa                      44.90 Aa
Alan (C3)                    Bonnie (C9)
        61.96 AA                      37.45 AA
        38.04 Aa                      62.55 Aa
Alice (C4)                   Carl (C10)
        98.12 AA               -          - AA
         1.88 Aa                     100.00 Aa
Ann (C2)                     Cecily (C11)
        37.59 AA               -          - AA
        62.41 Aa                     100.00 Aa
Benny (C8)                   Dennis (C12)
        61.42 AA               -          - AA
        38.58 Aa               -          - Aa
                                     100.00 aa
```

Figure 2: The probabilities of the horses being carriers (Aa) of the sick gene.

In Figure 2, we can see that it is very likely that Betsy is carrier of the sick gene. Both her parents (Ann and Alan) also has great probability of being carriers. However, a more thorough investigation shows that it is very unlikely that both of them are carriers at the same time. In Figure 3 we see that if Alan is known to be carrier, it becomes most unlikely that Ann is also carrier. This is because a sick gene is only inherited from one parent. The Figure shows that the gene is inherited from Alan to Betsy and Benny to Carl and Cecily.

The conclusion to the results would be very dependent on how much the farmer wants to be sure of getting the sick gene out of production. He can never be absolutly sure that he gets rid of the right horses, but he should at least get rid of Betsy, Ann and Bonnie. If also wants to get rid of Alan because he is easily replaced this would have no effect if he does not also get rid of Benny since Benny probably has inherited the sick gene if Alan has it.



```
A1 (C1)                       Betsy (C7)
   98.36 AA                      1.02 AA
    1.64 Aa                     98.98 Aa
A2 (C5)                       Bill (C6)
   98.36 AA                     96.89 AA
    1.64 Aa                      3.11 Aa
Alan (C3)                     Bonnie (C9)
       - AA                     96.73 AA
  100.00 Aa                      3.27 Aa
Alice (C4)                    Carl (C10)
   98.67 AA                         - AA
    1.33 Aa                    100.00 Aa
Ann (C2)                      Cecily (C11)
   96.94 AA                         - AA
    3.06 Aa                    100.00 Aa
Benny (C8)                    Dennis (C12)
    1.39 AA                         - AA
   98.61 Aa                         - Aa
                               100.00 aa
```

Figure 3: If we assume that Alan carries the sick gene, this Figure shows that Ann is probably not carrier.

This network has been installed on your computer with the HUGIN software. Open the network in the HUGIN Graphical User Interface (Note: not all browsers can open HUGIN directly). You can find the network in the directory where you installed HUGIN (e.g. C:Program FilesHuginHugin LiteSamples).

You can also find the samples at the Hugin download area[21].

Comments

A long list of areas have essential characteristics in common with the above example, e.g. medical diagnosis and treatment, credit valuation of costumers, search for minerals, monitoring of biological production plants, image understanding, information retrieval, and fault analysis.

The areas are characterized by a cause-effect structure, where effects are not completely determined. Sometimes an event has one effect and sometimes it has another. This phenomenon is called *causal uncertainty*. A domain characterized by causal uncertainty can be modeled by a BN.

Another characteristic of the areas is that the number of essential properties can not be observed directly. This is the *diagnosis problem*: You know only the symptoms, and from them you must conclude the causes. You must so to speak reason in the opposite direction of the links in the network.

[21] http://download.hugin.com/samples

6.1.3 Monty Hall Puzzle

Describing the Monty Hall Puzzle

The Monty Hall puzzle gets its name from an American TV game show, "Let's make a deal", hosted by Monty Hall. In this show, you have the chance to win some prize if you are lucky enough to find the prize behind one of three doors. The game goes like this:

- You are asked to select one of the three doors.
- Monty Hall (who knows where the prize is) opens another door behind which there is no prize.
- You are asked if you want to alter your selection (select the other closed door).
- You get the prize if it is behind the door you selected.

The problem of the puzzle is: What should you do at your second selection? Some would say that it does not matter because it is equally likely that the prize is behind the two remaining doors. This, however, is not quite true. The right answer is that you have the best chance of winning if you alter your selection - odds 2/3 in stead of 1/3. In the following, the solution is computed through the use of a simple *Bayesian network* (page 19).

Finding the Solution Using a Bayesian Network

The Monty Hall puzzle can be modeled with three random variables: Prize, First Selection, and Monty Opens.

- Prize represents the information about which door hides the prize. This means that it has three states: "Door 1", "Door 2", and "Door 3".
- First Selection represents your first selection. This variable also has the three states: "Door 1", "Door 2", and "Door 3".
- Monty Opens represents Monty Halls choice of door when you have made your first selection. Again, we have the three states: "Door 1", "Door 2", and "Door 3".

The door hiding the prize is known to Monty and thus Prize has an influence on Monty Opens. Monty will never choose to open the door of your first selection, so also First Selection has an influence on Monty Opens. This give us the network shown in Figure 1.

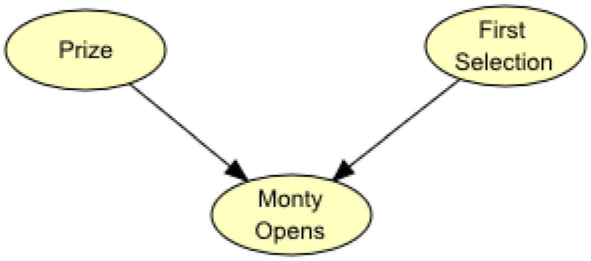
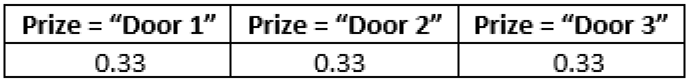


Figure 1: Bayesian network of the Monty Hall puzzle. The causal links describe that both Prize and First Selection have an influence on Monty Opens.

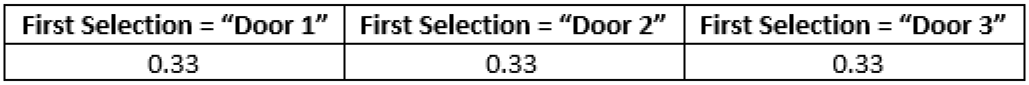
The conditional probability table (CPT) of Prize is shown in Table 1 (indicating ignorance about where the prize is hidden).



Prize = “Door 1”	Prize = “Door 2”	Prize = “Door 3”
0.33	0.33	0.33

Table 1: P(Prize).

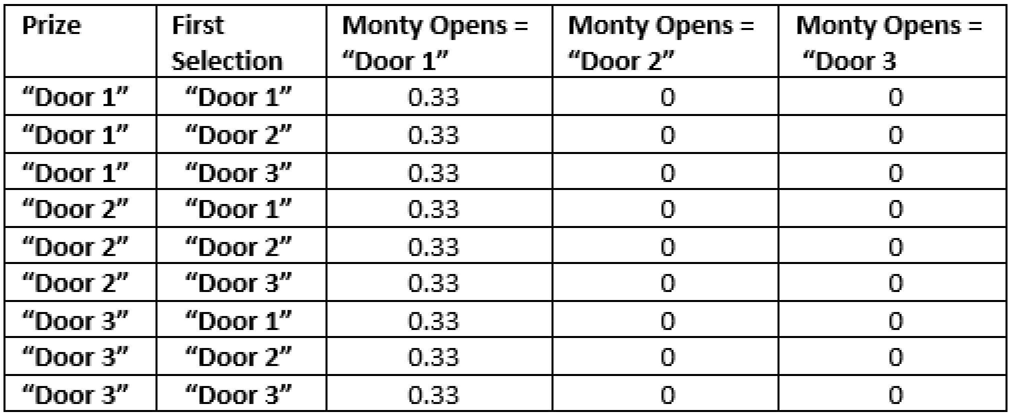
The CPT of First Selection is shown in Table 2 (the probabilities of this table are unimportant, as you will always select a specific state of this variable when you use the network).



First Selection = “Door 1”	First Selection = “Door 2”	First Selection = “Door 3”
0.33	0.33	0.33

Table 2: P(First Selection).

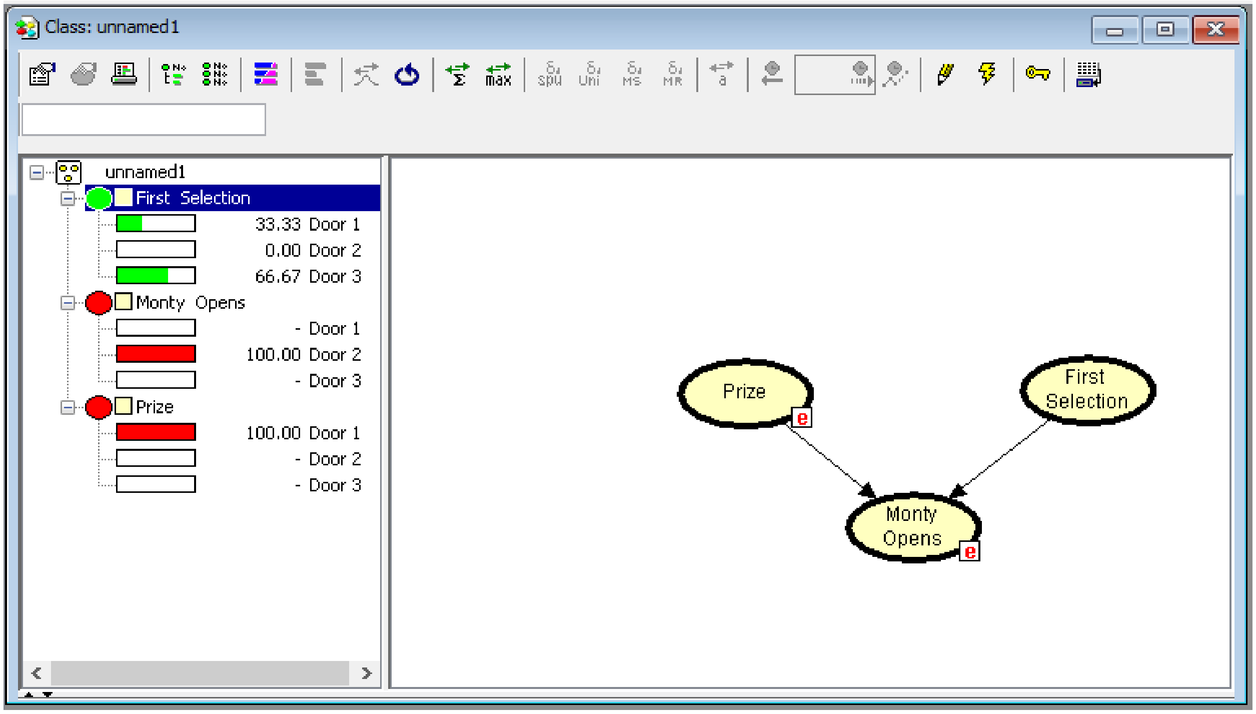
Table 3 shows the CPT of Monty Opens. This table states that if the prize is behind Door 1 and you have chosen Door 3, then Monty will open Door 2 for sure, since this door is the only possible door he can open. If you have selected the right door in your first selection, he will randomly choose one of the two other doors.



Prize	First Selection	Monty Opens = “Door 1”	Monty Opens = “Door 2”	Monty Opens = “Door 3
“Door 1”	“Door 1”	0.33	0	0
“Door 1”	“Door 2”	0.33	0	0
“Door 1”	“Door 3”	0.33	0	0
“Door 2”	“Door 1”	0.33	0	0
“Door 2”	“Door 2”	0.33	0	0
“Door 2”	“Door 3”	0.33	0	0
“Door 3”	“Door 1”	0.33	0	0
“Door 3”	“Door 2”	0.33	0	0
“Door 3”	“Door 3”	0.33	0	0

Table 3: P(Monty Opens | Prize, First Selection).

The network shown in Figure 1 with nodes having the CPTs of Table 1, 2, and 3 can be constructed very quickly using the Hugin Graphical User Interface. Right after compilation, the Node List Pane of the Hugin Graphical User Interface will appear as in Figure 2, which is an interactive Java applet allowing you to select first your own first selection and then the door opened by Monty. After this, the solution appears from the probabilities of Prize. The functionality of this applet is very similar to the functionality of the Hugin Graphical User Interface.



```
unnamed1
  First Selection
            33.33 Door 1
             0.00 Door 2
            66.67 Door 3
  Monty Opens
                - Door 1
           100.00 Door 2
                - Door 3
  Prize
           100.00 Door 1
                - Door 2
                - Door 3
```

Figure 2: Showing the list of nodes (variables) of the Monty Hall puzzle domain.

Figure 2 shows that it will always be best for you to alter your selection when Monty has opened a door (gives you 66.67% chance of winning the prize).

An easy way to convince yourself that this is true goes like this: At first you have 33.33% chance of choosing the right door and there is 66.67% chance of the prize being somewhere else. You know that Monty is going to open an empty door so when he does, this should not change a thing about your belief of your door being the right one. You still have 33.33% chance of having selected the right door. Thus there must be 66.67% chance of the prize being somewhere else (behind the last door).

6.2 Modeling Examples

In this section, we present a series of examples meant to give you ideas of the connection between a textual description and a network model.

Taenkeboks (page 526) gives a good and thorough description of how you can get from a description of a simple game of dice to an influence diagram that can help find a winning strategy. The network is the result of a 7th semester project at Aalborg University, Denmark, and the documentation is written by one of the participating students.

Asia (page 532) is a small Bayesian network that computes the probability of a patient having tuberculosis, lung cancer or bronchitis respectively based on different factors - for example whether or not the patient has been to Asia recently.

Poker (page 533) is a Bayesian network that models a simplified version of the game of poker. The network can help predict who has the best hand - your or your opponent.

The Mildew examples (page 535) are four different influence diagrams modeling the situation where a farmer has to decide on a treatment with fungicides for a wheatfield. The influence diagrams model different scenarios depending on the information available.

Implementations of these examples come with the installation of any version of HUGIN. If you have lost them or have an older installation of HUGIN, you have the opportunity to download them[22].

If you have made some examples of your own, you are welcome to put them at forum.HUGIN.com[23]. Here you can also find discussions, links and other things related to the field of Bayesian networks.

6.2.1 Introduction

This document describes how to build a (limited-memory) influence diagram (LIMID) to model a simplified version of the dice game called "Tænkeboks". The objective of this modeling is to find a winning strategy against a presumed strategy of the opponent. The influence diagram is designed and afterwards implemented in HUGIN. Finally this implementation is used to determine an optimal counter strategy against the opponent's presumed strategy.

This document has been written by Michael Höhle with the support of Søren Andersen, Lisa Elgaard, Brian Jensen, Brian Kristiansen and Søren Mogensen. The network and it's documentation originate from a DAT3 (7th semester) project worked out by group C1-213 at Department of Computer Science, Aalborg University[24], in the fall of 1997. The supervisor of the DAT3 project was Dennis Nilsson.

The Simplified Tænkeboks

Consider the following simplified version of the dice game Tænkeboks. Two players named Arne and Bente (typical Danish names) participate in the game. Each player holds a two sided die (with the labels "1" and "2").

Initially both players roll their die without showing the result to each other. The goal of the game is to guess how many identical die values are present based upon knowledge of one's own die. The following bids are possible:

- 1 (at least one "1"),
- 2 (at least one "2"),
- 2·1 (both dice show "1"),
- 2·2 (both dice show "2").

Assume Arne starts the game. He has to choose between one of the above 4 bids. After Arne's bid, Bente has to decide whether to believe in Arne's bid and hence bid up or to call. If Bente doesn't call it is Arne's turn to consider Bente's last bid. The game continues until one of the players decides to call. A call results in both players revealing their die thus resolving whether the bid which was called upon was present on the table or not. This process determines winner and loser of the game.

Influence Diagrams and the Simplified Tænkeboks

Imagine that one foggy night at the pub you convince your best friend Arne to play some rounds of the above game. To make things more interesting the loser of each game has to buy the next round of beer. What is a good strategy for Bente?

There are several approaches e.g. one could analyze the above game by game theoretic methods or one could try to use influence diagrams. Since game theory tends to be quite complex and cumbersome we shall concentrate on the latter.

Based on an estimate of Arne's strategy we will solve the influence diagram in order to find an optimal counter strategy. The key problem is how to make a good estimate of Arne's strategy. We assume that you have been playing quite a lot of games of simple Tænkeboks against Arne and hereby achieved a pretty good understanding of his strategy.

[22] http://download.hugin.com/samples

[23] http://forum.hugin.com/

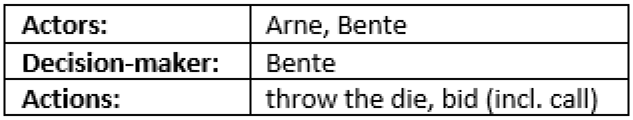
[24] https://www.cs.aau.dk/

An Influence Diagram for Simple Tænkeboks Implemented in HUGIN

For the sake of simplicity we assume from now on that Arne always is in the lead, i.e., he always is the first to bid in a game.

Determining the Chance and Decision Variables

Initially we need to find out which chance and decision variables we need and which states they should have. A prudent way to obtain this is to find the actors in our game and the actions they can perform.



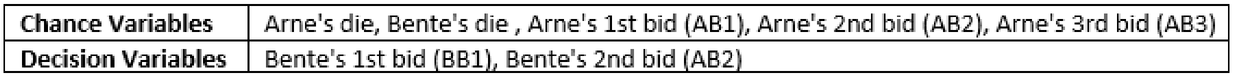
Actors:	Arne, Bente
Decision-maker:	Bente
Actions:	throw the die, bid (incl. call)

By analyzing the maximal sequence of bids in one game we determine the maximum amount of bids for each player.

Arne: 1, Bente: 2, Arne: 2·1, Bente: 2·2, Arne: call

Arne has a maximum of three bids during one game. Bente's maximum is two bids.

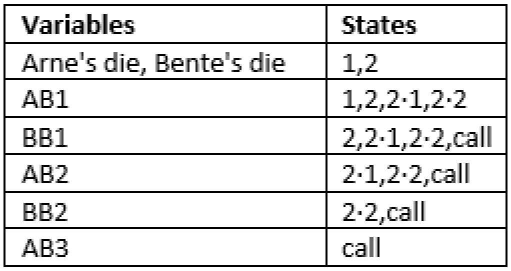
Every action is now converted into either a chance variable or a decision variable according to the following rule: If the action is a decision which has to be made by the decision maker it is converted into a decision variable, otherwise it becomes a chance variable. The result is:



Chance Variables	Arne's die, Bente's die , Arne's 1st bid (AB1), Arne's 2nd bid (AB2), Arne's 3rd bid (AB3)
Decision Variables	Bente's 1st bid (BB1), Bente's 2nd bid (AB2)

Note: the variable "Bente's die" which corresponds to the result of Bente rolling her die becomes a chance variable, since Bente has no influence on the result of rolling the die.

After finding the variables for our influence diagram it is time to consider which states they should have. The states of the die-variables represent the outcome of rolling a die, whereas the states of the bid-variable reflect the possible bids at the point of the bid-variable. Note how the number of states is based on the maximum bid sequence.



Variables	**States**
Arne's die, Bente's die	1,2
AB1	1,2,2·1,2·2
BB1	2,2·1,2·2,call
AB2	2·1,2·2,call
BB2	2·2,call
AB3	call

Causal and Information Dependences

The next task is to establish the dependences - represented by links - between the found variables. Our objective with the links is twofold:

- The links indicate the progress of the game
- a link into one of player X's bid-variables reflects the following: X's bidding strategy is based on information about the state of the variable from which the link comes.

Experience from playing simple Tænkeboks tells us that when bidding you usually only consider your own dice and the opponent's last bid. The two objectives and the above experience result in the following diagram shown in Figure 1.

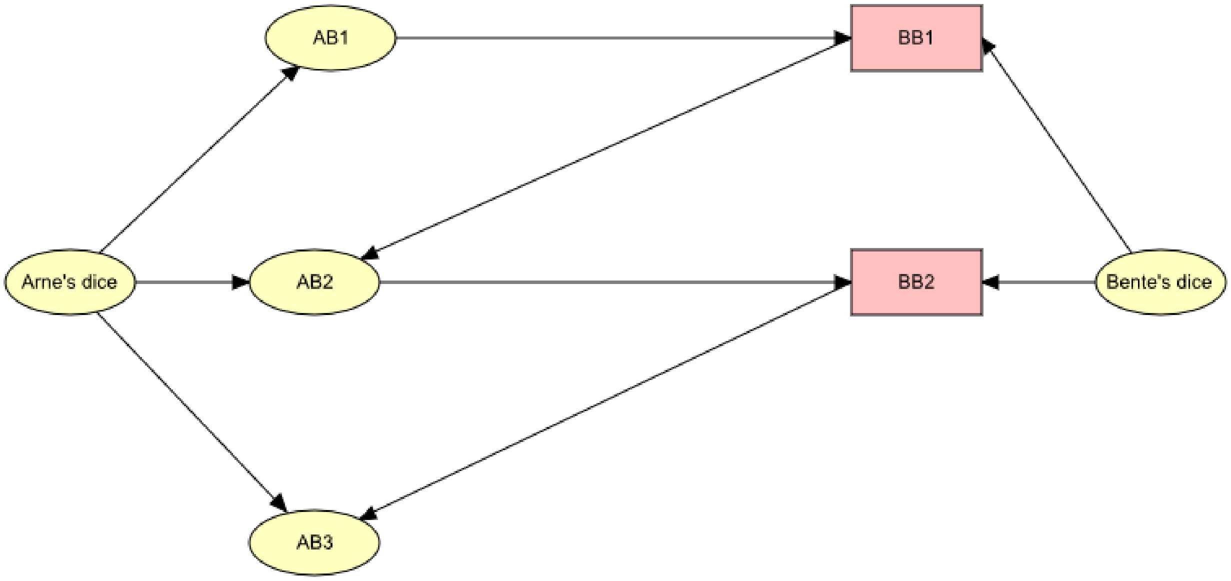


Figure 1: The chance variables and decision variables in the influence diagram together with their dependences.

Note: Arne's strategy on his first bid only depends on the value of his die. We don't exactly know his strategy, but we will use our best guess.

The Utility Functions

After introducing the decision variables it is also necessary to add utility functions which determine the outcome of taking a decision. Our utility functions have two goals

1. To determine whether Bente wins or loses once a player calls.
2. To ensure that Bente only performs valid actions (e.g. restrict bidding 1 after Arne bid 2)

The resulting diagram with all the utilities is displayed in Figure 2. To ensure item 1 utility functions are placed between consecutive bids (e.g. U5 between BB1 and AB2). These functions are connected to both the consecutive bid variables and the 2 die variables. All this information is necessary in order to check if the latter bid was a call and if so whether the bid before the call was present or not. This step result in the utility functions U4,U5,U6 and U7. Item 2 is achieved by putting utility functions between AB1 and BB1 (U1) as well as between AB2 and BB2 (U2). Later when assigning values to the utility functions all the illegal actions will be assigned very low values.

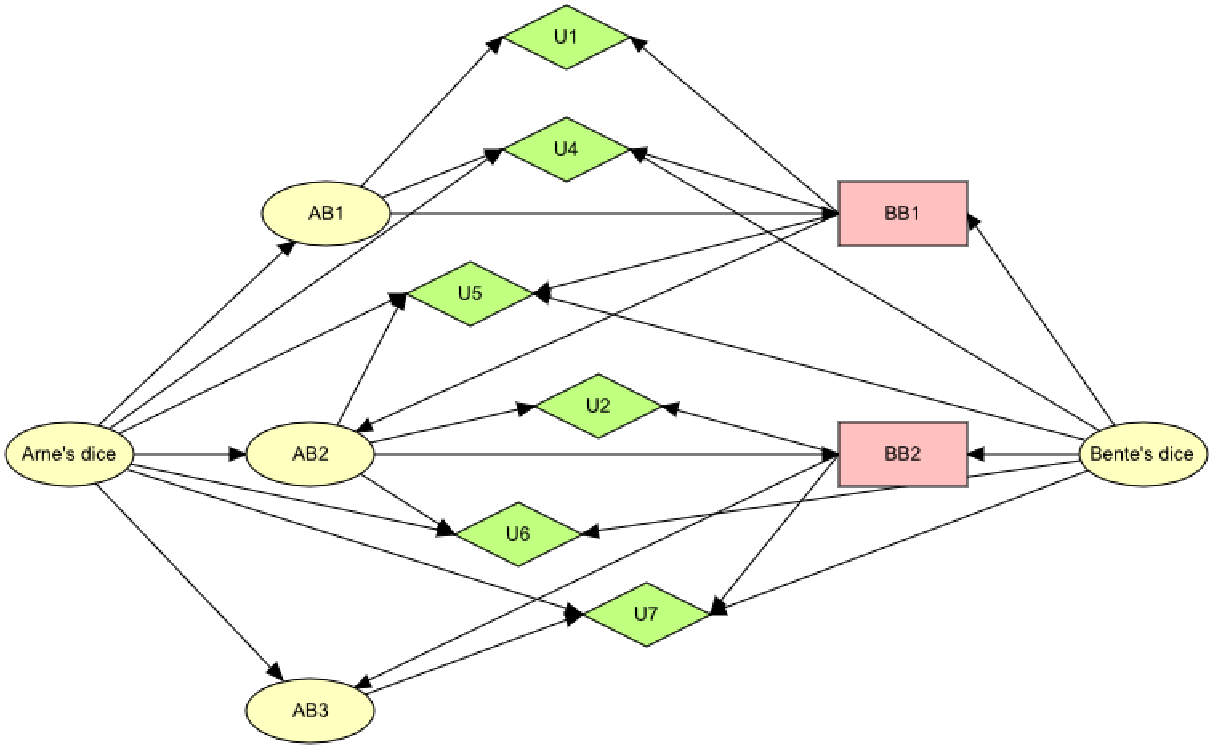


Figure 2: Influence diagram for our model. U1 and U2 guarantee that Bente’s bids are valid and U4, U5, U6, and U7 determine whether Bente wins or not.

Since U1’s set of parents is a subset of U4’s parents the two can be combined in an additive way. This simply means that U1’s utility values are added to U4’s values in the appropriate columns of U4. Similarly U2 and U6 are combined. The resulting influence diagram is displayed in Figure 3.

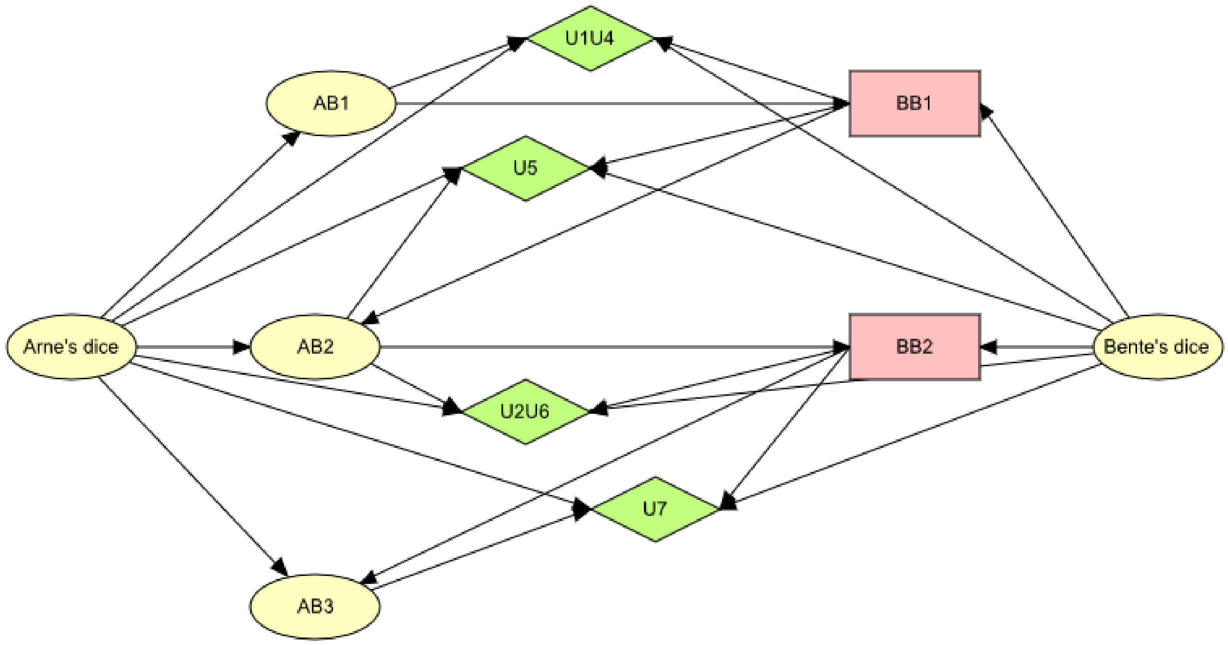


Figure 3: The final influence diagram modeling simple Tænkeboks. Notice how the utility functions have been combined.

Assigning Probabilities and Utility Functions to the Variables

After determining the variables, functions, and their dependences it is necessary to assign the conditional probabilities to the chance variables and utility values to the utility functions. This task can be divided into three subtasks. The gory details of the exact tables implemented in HUGIN are not that important, but are included should you become curious.

1. Assigning probabilities to the two die variables. *[Gory details]* (page 540)
2. Assigning conditional probabilities to Arne's bid variables depending on our estimate of his strategy.

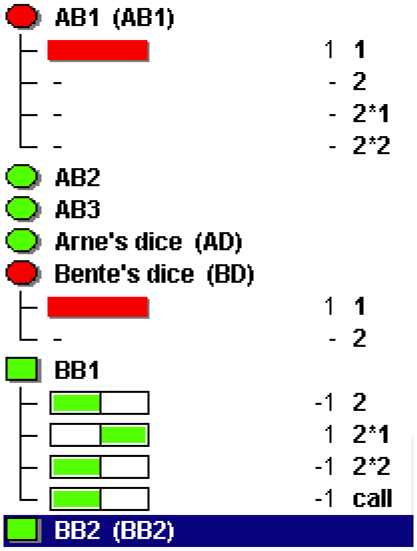
To accomplish this task we need a good guess of Arne's strategy. To keep things simple we assume that he is following the pretty reasonable deterministic strategy:

- When starting he bids what his die shows. Hereby his bid will always be present on the table.
- If Bente bids 2 and he rolls "1" then he calls.
- If Bente bids 2 and he rolls "2" then he bids 2·2.
- If Bente bids 2·1 or 2·2 he always calls.

In a real game it is more likely that Arne will follow a non-deterministic strategy to conceal his real strategy. Once you get the hang of how the optimal counter strategy is found you can use the influence diagram straight away to find counter strategies against non-deterministic strategies as well. *[Gory details]* (page 540) 3. Assigning values to the utility functions so they fulfill the demands from the preceding section. *[Gory details]* (page 540)

Finding the Optimal Strategy

We are ready to use the influence diagram to find an optimal counter strategy for Bente against Arne's strategy.



AB1 (AB1)
1 1
- 2
- 2*1
- 2*2
AB2
AB3
Arne's dice (AD)
Bente's dice (BD)
1 1
- 2
BB1
-1 2
1 2*1
-1 2*2
-1 call
BB2 (BB2)

Figure 4: Screenshot from HUGIN where the evidence BD=1 and AB1=1 was entered and since sum-propagated. The action 2·1 in BB1 has the highest expected utility (expected utility = 1).

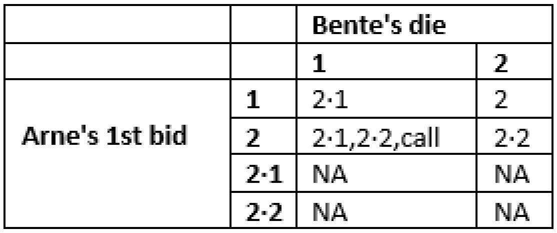
The following algorithm in HUGIN gives us the desired result:

1. Initialize the network
2. **For each** state X of Bente's die variable **do**
    1. enter X as 100% evidence (click on X in run-mode)
    2. **For each** possible state Y of Arne's AB1 variable **do**
        1. enter Y as 100% evidence (click on Y in run-mode)
        2. sum propagate
        3. The best bid for Bente in BB1 is the action in BB1 with the highest expected utility. (see Figure 4)

In step 2.1 we only loop through the possible states of AB1. With Arne's strategy it is for example impossible that Arne bids 2·1 in AB1.

Had the game been longer it would have been necessary to insert the optimal action in BB1 as evidence and loop through the states in AB2 in order to determine the optimal action in BB2. But since Bente never really gets to bid a second time this step isn't included in the algorithm.

By applying this algorithm the following counter strategy for Bente is determined:



		Bente's die	
		1	2
Arne's 1st bid	1	2·1	2
	2	2·1,2·2,call	2·2
	2·1	NA	NA
	2·2	NA	NA

For example, if Bente rolls 2 and Arne bids 2 in his first bid then the strategy tells Bente to bid 2·2. In the situation where Bente rolls a 1 and Arne bids 2 three actions are equally good: 2·1,2·2 and call. To get a deterministic strategy it is sufficient to simply pick one of the possibilities. The NA's in the table symbolize that a situation cannot occur with Arne's choice of strategy. All of Bente's choices are therefore equally good.

The question is how good this counter strategy actually is. By simulating games between the two strategies in the four possible configurations of the dice ([AD=1 and BD=1], [AD=1 and BD=2], [AD=2 and BD=1] or [AD=2 and BD=2]) it becomes obvious that Bente will win 75% of the games! In other words, out of 12 rounds of beer Bente (you) will only have to pay for the 3 while Arne will have to pay for 9. Cheers!

Conclusion

An influence diagram to model a simple version of the game Tænkeboks was designed and since implemented in HUGIN. The resulting diagram is displayed in Figure 3 and the HUGIN implementation can be downloaded from the HUGIN networks page[25].

Arne's strategy was guessed and entered into the tables of the diagram. Using HUGIN to enter evidence and propagate, an optimal counter strategy for Bente was found where she could win 75% of the games.

Perspectives

A game theoretic analysis of the game reveals that the strategy assumed for Arne is not a very good one. He could e.g. prefer a strategy where he always bids 2·(whatever he rolled). Hereby he is certain to win 50% of the games - even against Bente's counter strategy found with the influence diagram! (Game theoretic note: The mentioned strategy combination forms a Nash-equilibrium where both players receive a payoff of zero.)

The assumption that Arne follows a deterministic strategy is pretty unlikely. By changing the values in the tables of Arne's bid variables Arne can easily be modified to play with a randomized strategy. The algorithm described above also works to find a deterministic counter strategy against Arne's randomized strategy.

Simple Tænkeboks is quite a boring game. Possible enhancements to increase the joy could be: more sides on your die, more dice per player, a notion of jokers, or even extra players. The problem is that even small enhancements make the complexity of the influence diagram explode.

This network has been installed on your computer with the HUGIN software. Open the network in the HUGIN Graphical User Interface (Note: not all browsers can open HUGIN directly). You can find the network in the directory where you installed HUGIN (e.g. C:Program FilesHuginHugin LiteSamples). You can also find the samples at the HUGIN download area[26].

6.2.2 Asia (the Chest Clinic)

This example originates from the seminal paper of *Lauritzen & Spiegelhalter (1988)* (page 541).

Shortness-of-breath (dyspnoea) may be due to tuberculosis, lung cancer, bronchitis, more than one of these diseases or none of them. A recent visit to Asia increases the risk of tuberculosis, while smoking is known to be a risk factor for both lung cancer and bronchitis. The results of a single chest X-ray do not discriminate between lung cancer and tuberculosis, as neither does the presence or absence of dyspnoea.

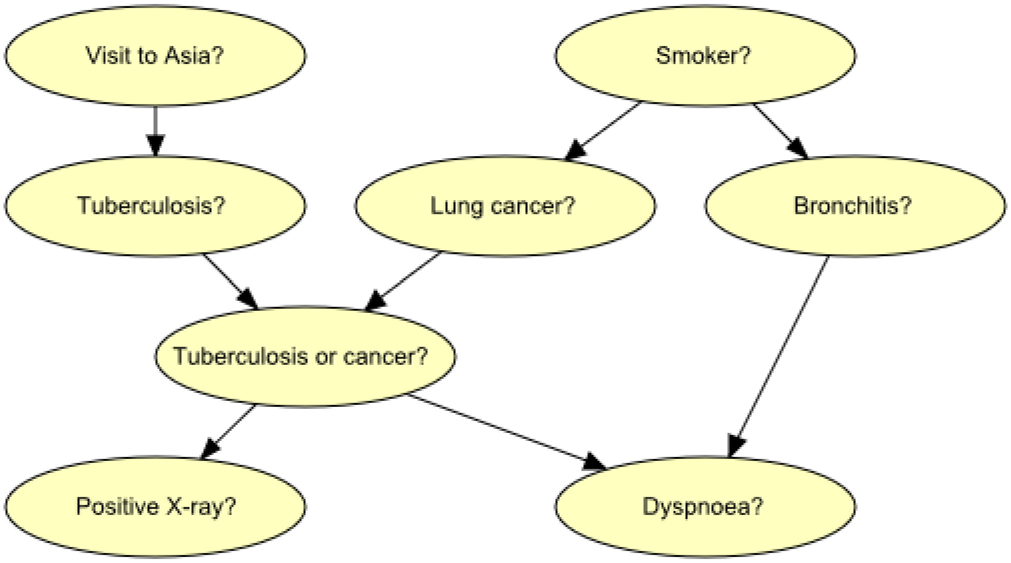
If we learn the fact that a patient is a smoker, we will adjust our beliefs (increased risks) regarding lung cancer and bronchitis. However, our beliefs regarding tuberculosis are unchanged (i.e., tuberculosis is conditionally independent of "smoking" given the empty set of variables). Now, suppose we get a positive X-ray result for the patient. This will affect our beliefs regarding tuberculosis and lung cancer, but not our beliefs regarding bronchitis (i.e., bronchitis is conditionally independent of X-ray given smoking). However, had we also known that the patient suffers from

[25] http://download.hugin.com/samples

[26] http://download.hugin.com/samples

shortness-of-breath, the X-ray result would also have affected our beliefs regarding bronchitis (i.e., "bronchitis" is not conditionally independent of "X-ray" given "smoking" and "dyspnoea").

A Bayesian network for the knowledge described above can look like this:



The above figure shows the node *labels* inside each node. A unique node name is assigned to each node in the network node names are: "Visit to Asis?" has name A, "Smoker?" has name S, "Has tuberculosis" has name T, "Has lung cancer" has name L, "Has bronchitis" has name B, "Tuberculosis or cancer" has name E, "Positive X-ray?" has name X, and "Dyspnoea?" has name D.

This network has been installed on your computer with the HUGIN software. Open the network in the HUGIN Graphical User Interface (Note: not all browsers can open HUGIN directly). You can find the network in the directory where you installed HUGIN (e.g., C:Program FilesHuginHugin LiteSamples).

You can also find the samples at the HUGIN download area[27].

6.2.3 A simplified poker game

This example originates from the textbook *Bayesian Networks and Decision Graphs* (page 541).

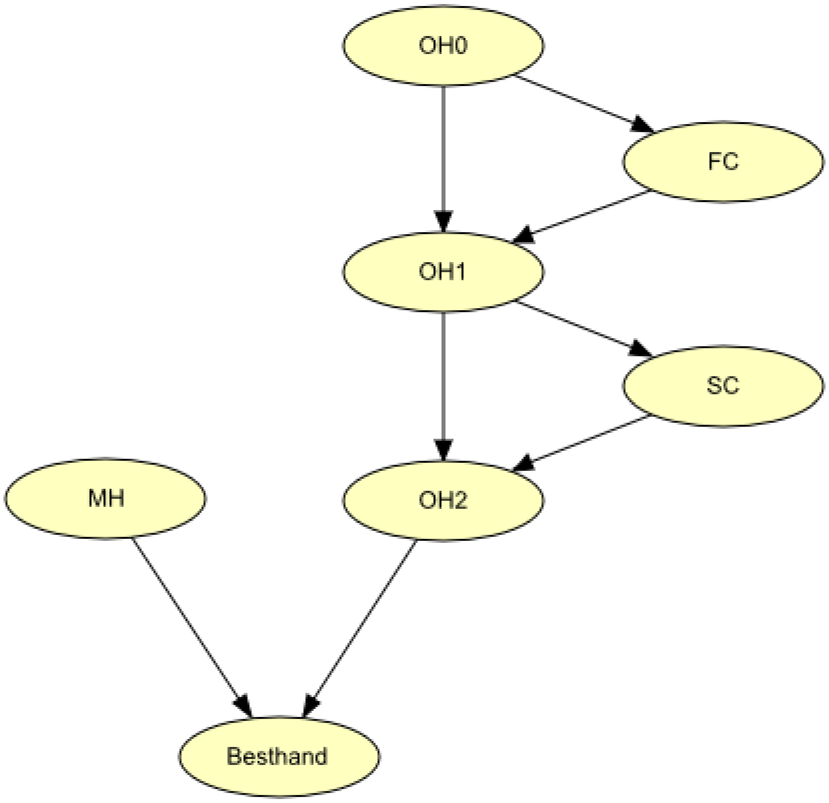
Consider a simplified poker game with the following rules:

Each player receives three cards and is allowed two rounds of changing cards. In the first round you may discard any number of cards from your hand and get replacements from the pack of cards. In the second round you may discard at most two cards.

The various types of hands are classified in the following way (in increasing rank): nothing special, 1 ace, 2 of the same value, 2 aces, flush (3 of a suit), straight (3 of consecutive values), 3 of the same value, straight flush.

The following Bayesian network can compute the probability of you having the best hand after the two ronds of changing, based on the number of cards your opponent has discarded and an assumed strategy of the opponent:

[27] http://download.hugin.com/samples



The nodes OH0, OH1 and OH2 represent the opponents hand initially, after the first change (FC) and after the second change (SC) respectively. MH represents my hand before the bidding starts. Note that we have exact knowledge of the state of this node, so the initial distribution is not important.

Of course the probabilities of the various states of OH2 depend on the presumed strategy of the opponent.

In the network enclosed in the installations of HUGIN, the following strategy for the opponent is assumed:

- If nothing special, then change 3.
- If 1 ace, then keep the ace.
- If 2 of consecutive values or 2 of a suit or 2 of the same value, then discard the third card.
- If 2 of a suit and 2 of consecutive values, then keep 2 of a suit.
- If 2 of a suit and 2 of the same value or 2 of consecutive values and 2 of the same value, then keep the 2 of the same value.
- If flush, straight, 3 of the same value or straight flush, then keep it.

Now you can enter the states of FC, SC and MH and get the probability of you having the best hand.

This network has been installed on your computer with the HUGIN software. Open the network in the HUGIN Graphical User Interface (Note: not all browsers can open HUGIN directly). You can find the network in the directory where you installed HUGIN (e.g. C:Program FilesHuginHugin LiteSamples). You can also find the samples at the HUGIN download area[28].

[28] http://download.hugin.com/samples

6.2.4 The Mildew examples

These examples originates from An *introduction to Bayesian Networks* (page 541).

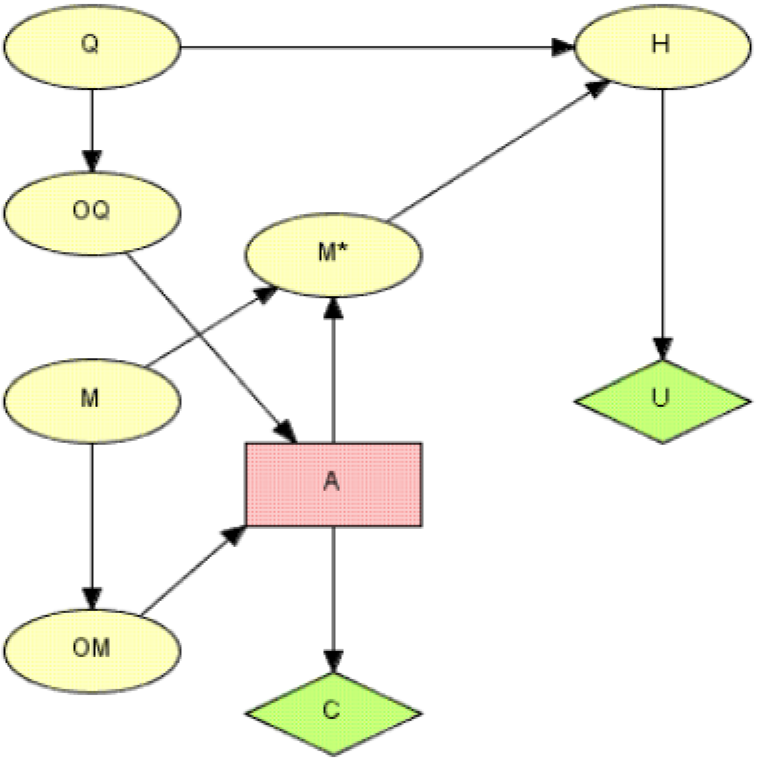
The networks shown in this example have been installed on your computer with the HUGIN software. Open the network in the HUGIN Graphical User Interface (Note: not all browsers can open HUGIN directly). You can find the networks in the directory where you installed HUGIN.

You can also find the samples at the HUGIN download area[29].

The basic situation

Approximately two months before harvesting a wheat field a farmer can observe the state of the crop and he can observe whether it has been attacked by mildew. If there is an attack he should decide on a treatment with fungicides.

The limited memory influence diagram below models this situation:



There are five chance nodes:

- The actual state of the crop, Q with states fair (f), average (a), good (g) and very good (v).
- The actual mildew-situation, M with states no, little (l), moderate (m) and severe (s).
- The state of the crop at harvest time, H with the states from Q plus rotten (r), bad (b) and poor (p).
- The observation, OQ, of Q.
- The observation, OM, of M.

[29] http://download.hugin.com/samples

There is also an action node, A (modeling the fungicide-treatment) with actions no, light (*l*), moderate (*m*) and heavy (*h*).

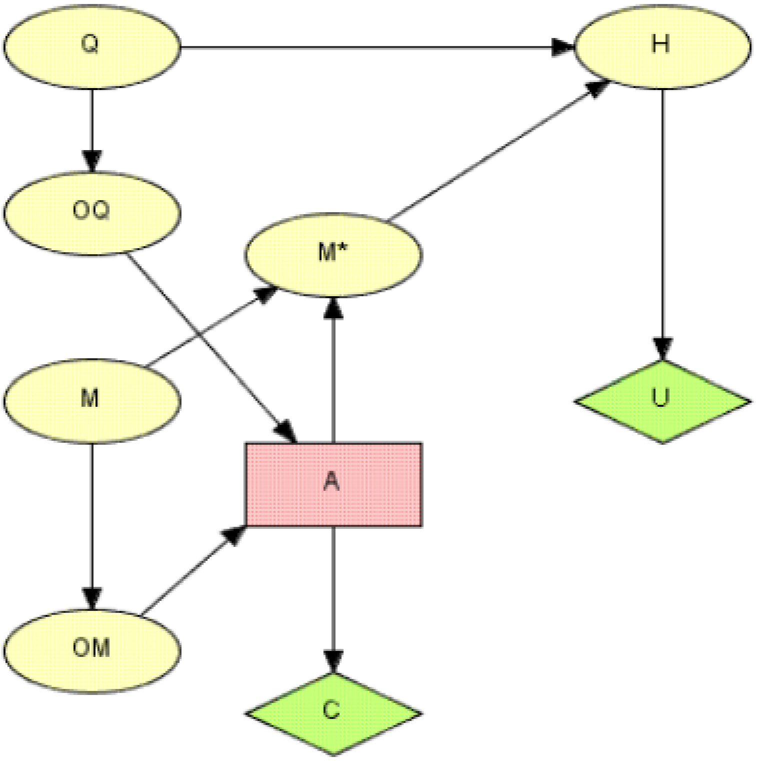
Now we extend the example with a decision on the time of harvest (T). This decision is to be made two months after the decision on fungicides. Note that H models the state of the field at the time of decision T.

When there are more than one decision node in the model, the computation of the expected utility of an action requires knowledge of what will be known when future decisions are made. For each decision node it is important to specify using Informational links the information known to the decision maker at the corresponding decision. A link from node X to decision node D indicates that the state of X is known when the decision on D is made.

The model depends on what information is available at the time of deciding on T. We present four different scenarios:

Scenario 1

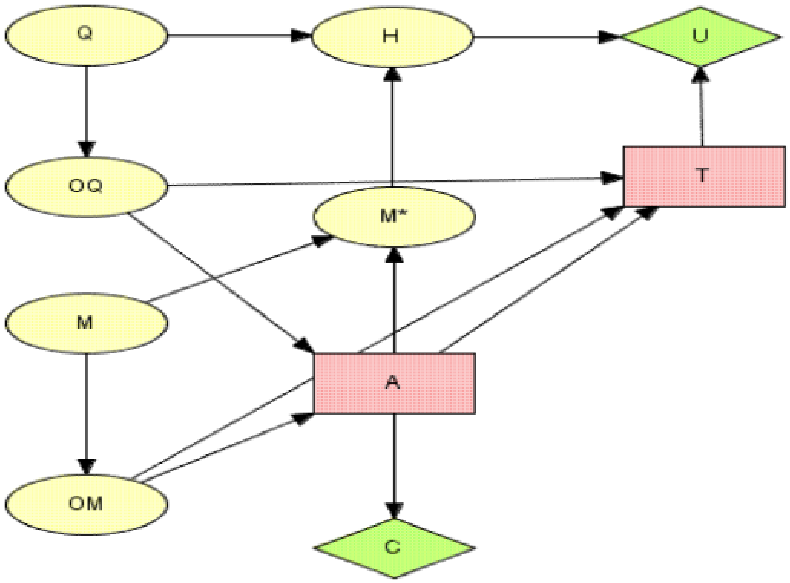
In this scenario there is no further information.



HUGIN computes the (optimal) strategy by Single Policy Updating and the expected utility function and the posterior probability distribution for each node under the strategy.

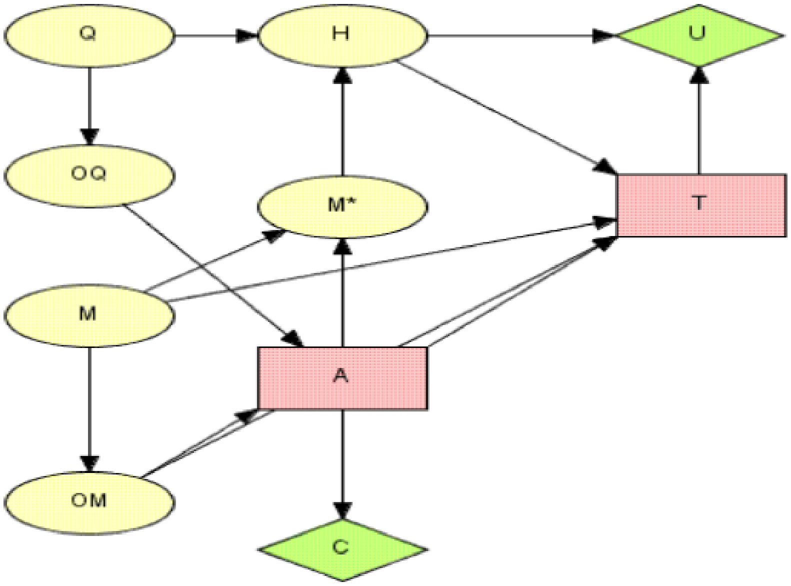
Scenario 2

In this scenario the state of H is known. This means that there is a link from H to T. A link from a chance node to a decision node means that evidence on the chance node has to be available at the time of the decision in order for the utilities to be computed correctly.



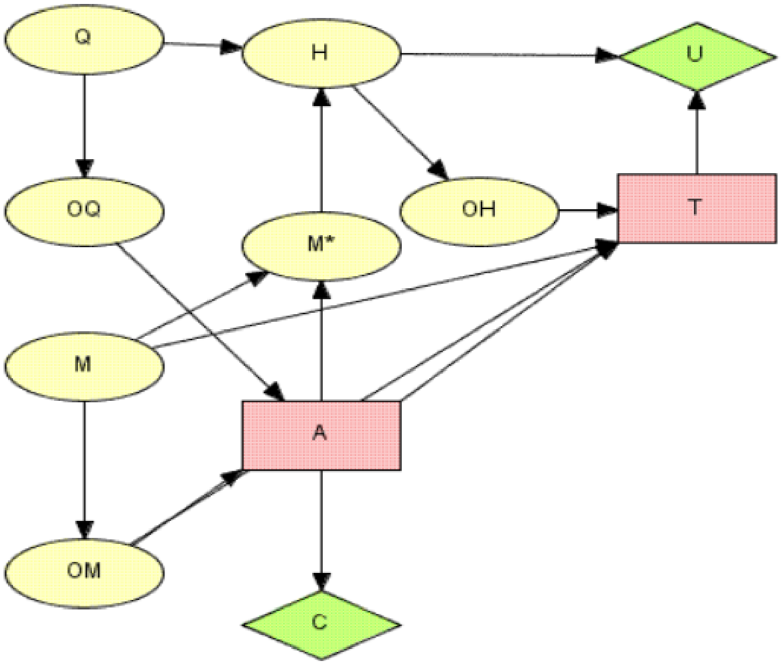
Scenario 3

In this scenario there is an observation on H available, but it is imprecise, so it doesn't necessarily show the actual state of the crop. Still it has an impact on our belief in the state of H and also on the expected utilities of the various possibilities for T.



Scenario 4

In this scenario the farmer does not recall the observations on OQ and OM made prior to decision A nor does he recall the decision made at A. In this example the policy on T is a function of OH only.



6.2.5 Competitive Analysis Bayesian Networks

Author: *Professor Philippe Baumard, University of Versailles, France*

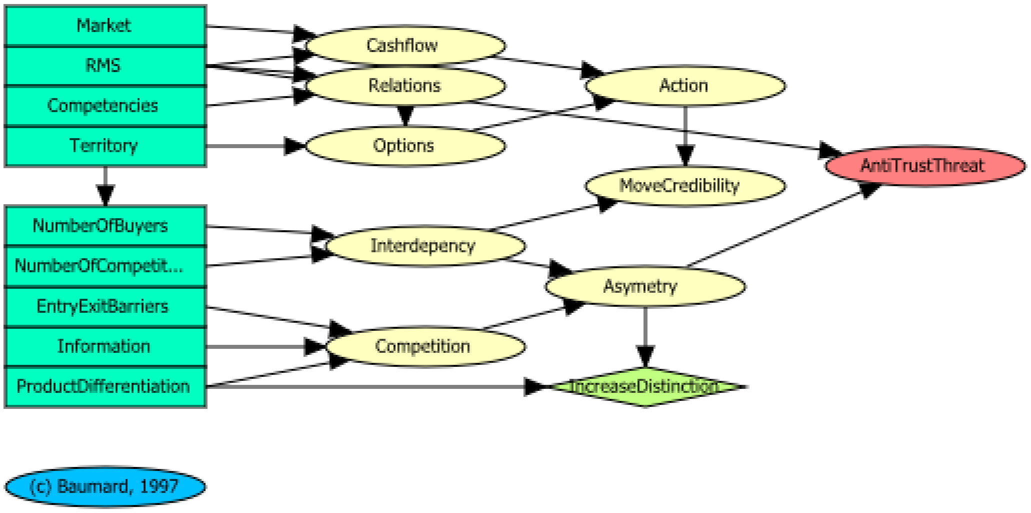
These two Bayesian networks are part of a series of 30 Strategic Analysis Bayesian networks developed for teaching MBA students competitive analysis, at New York University Stern School of Business. They are based on strategic management and industrial economics literature. Probabilities have been entered as to reflect findings of major authors.

“Competitive Asymmetries” assesses the threat of an anti-trust case based on industry structure and the firm’s position in this industry.

“Threat of Entry” assesses the probable entry strategy of a new entrant, based on the industry characteristics.

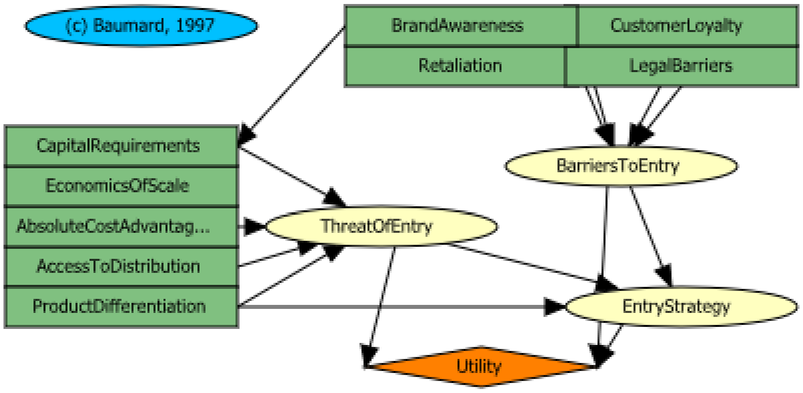
These networks, along with 28 others, are included in a CD-Rom coming along with a book entitled “Competitiveness and Information Systems” (Compétitivé et Systèmes d’Information), forthcoming in October 1998, by Masson/InterEditions in Paris. The book develops a Bayesian-based methodology and tools for assessing the contributions of information technologies and systems to a firm’s competitiveness.

Competitive Asymmetries



This network has been installed on your computer with the HUGIN software. Open the network in the HUGIN Graphical User Interface (Note: not all browsers can open HUGIN directly). You can find the network in the directory where you installed HUGIN (e.g. C:Program FilesHuginHugin LiteSamples). You can also find the samples at the HUGIN download area[30].

Threat of Entry



This network has been installed on your computer with the HUGIN software. Open the network in the HUGIN Graphical User Interface (Note: not all browsers can open HUGIN directly). You can find the network in the directory where you installed HUGIN (e.g. C:Program FilesHuginHugin LiteSamples). You can also find the samples at the HUGIN download area[31].

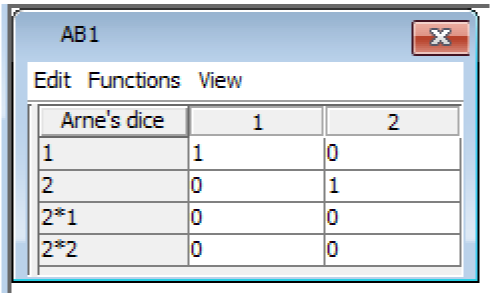
[30] http://download.hugin.com/samples
[31] http://download.hugin.com/samples

6.2.6 The Two Dice Variables

The die variable represents the outcome of rolling a two-sided die. We assume that both Arne and Bente are playing with non-skew dice, so for both variables there is a 50% probability to obtain “1” and 50% for “2”.

Arne’s Bid Variables

It is pretty easy to convert our guess of Arne’s strategy to conditional probabilities. As an example we show the table of AB1:



Arne's dice	1	2
1	1	0
2	0	1
2*1	0	0
2*2	0	0

The only thing that should be noted is that we in AB2 and AB3 for technical reasons respond to a “call” from Bente with a “call”. This is is to ensure that once a game is over no player starts to bid again and no extra utility values are granted for a game that is already won!

The Utility Functions

This task is the most difficult of the three and you are advised to consult net file for the exact tables. The utility functions can be grouped into two:

1. U5 and U7 which determine the winner of a game.
2. U1U4 and U2U6 which ensure that Bente only performs valid actions and determine the winner of a game.

The first ones are pretty straightforward. In both U5 and U7 it is Arne who’s turn it currently is. For each situation where Arne calls the winner of the game is determined from the states of the two die variables. In case Bente wins we assign this situation a utility value of 1, should Bente lose the utility is set to -1.

Situations where Arne doesn’t call are so called “neutral situations” since the game continues and no winner is found. Hence we assign neutral situations the value 0.

With our “call->call” trick the situation where both players called corresponds to an already finished game. Bente’s outcome has therefore already been found, hence this situation is also assigned the value 0.

For U1U4 and U2U6 some extra work has to be done, because it is Bente’s turn. The assigning of values follows the same schema as for U5 and U7: Should Bente decide to call her outcome is determined.

All the other possible bids are either neutral bids since they don’t end the game or illegal bids. Neutral bids are assigned the value 0 whereas illegal bids are assigned the value -10 - a real punishment! (the proper value for this situation would be minus infinity, but -10 will do for our game). We enforce our “call->call” trick by making all other bids than call illegal actions once Arne called.

CHAPTER

SEVEN

REFERENCES

- **S. K. Andersen, K. G. Olesen, F. V. Jensen & F. Jensen (1989).** Hugin – a shell for building Bayesian belief universes for expert systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1080-1085, Detroit, Michigan, Aug. 20-25.
- **I. Beinlich, H. Suermondt, R. Chavez & G. Cooper (1989).** The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. *Proceedings of the second European Conference on Artificial Intelligence in Medicine,* pp 247-256.
- **B. Boerlage (1992).** Link Strength in Bayesian Networks. MSc thesis, Department of Computer Science, University of British Columbia, Canada. Also *Tech. Report 94-17,* Department of Computer Science, University of British Columbia, Canada.
- **O. Cappe & E. Moulines**. Online em algorithm for latent data models. Journal of the Royal Statistical Society Series B (Statistical Methodology), 71(3):593-613, 2009.
- **G. Cooper (1984).** NESTOR: A computer-based medical diagnostic aid that integrates causal and probabilistic knowledge. PhD thesis, Medical Information Sciences, Stanford University, Stanford, CA.
- **R. G. Cowell & A. P. Dawid (1992).** Fast retraction of evidence in a probabilistic expert system. *Statistics and Computing,* 2:37-40.
- **A. P. Dawid (1992).** Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing,* 2:25-36.
- **D. Heckerman, J. Breese & K. Rommelse (1994).** Troubleshooting under Uncertainty. *Technical report msr-tr-94-07.* Microsoft Research, Advanced Technology Division, Microsoft Corporation.
- **F. Jensen (1994).** Implementation aspects of various propagation algorithms in Hugin. Research Report R-94-2014, Department of Mathematics and Computer Science, Aalborg University, Denmark.
- **F. Jensen & S. K. Andersen (1990).** Approximations in Bayesian belief universes for knowledge-based systems. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence,* pages 162-169, Cambridge, Massachusetts, July 27-29.
- **F. Jensen, F. V. Jensen & S. L. Dittmer (1994).** From influence diagrams to junction trees. In R. L. de Mantaras and D. Poole, editors, *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence,* pages 367-373, Seattle, Washington, July 29-31. Morgan Kaufmann, San Mateo, California.
- **F. V. Jensen, S. L. Lauritzen & K. G. Olesen (1990(1)).** Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly,* 4:269-282.
- **F. V. Jensen, K. G. Olesen & S. K. Andersen (1990(2)).** An algebra of Bayesian belief universes for knowledge-based systems. *Networks,* 20(5):637-659. Special Issue on Influence Diagrams.
- **F. V. Jensen, B. Chamberlain, T. Nordahl & F. Jensen (1991).** Analysis in Hugin of data conflict. In P. P. Bonissone, M. Henrion, L. N. Kanal, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence,* volume 6, pages 519-528. Elsevier Science Publishers, Amsterdam, The Netherlands.

- **F. V. Jensen (2001).** *Bayesian Networks and Decision Graphs,* Springer.
- **F. V. Jensen (1996).** *An Introduction to Bayesian Networks,* Springer.
- **U. Kjærulff (1990).** Triangulation of graphs - algorithms giving small total state space. *Research Report R-90-09.* Department of Mathematics and Computer Science, Aalborg University, Denmark.
- **S. L. Lauritzen (1992).** Propagation of probabilities, means, and variances in mixed graphical models. *Journal of the American Statistical Association (Theory and Methods),* 87(420):1098-1108.
- **S. L. Lauritzen (1995).** The EM algorithm for graphical association models with missing data. *Computational Statistics & Data Analysis,* 19:191-201.
- **S. L. Lauritzen, A. P. Dawid, B. N. Larsen & H.-G. Leimer (1990).** Independence properties of directed Markov fields. *Networks,* 20(5):491-505. Special Issue on Influence Diagrams.
- **S. L. Lauritzen and D. Nilsson.** Representing and solving decision problems with limited information. Management Science, 47(9):1235–1251, Sept. 2001.
- **S. L. Lauritzen & D. J. Spiegelhalter (1988).** Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B (Methodological),* 50(2):157-224.
- **J. Matheson (1990).** Using Influence diagrams to value information and control. *Influence Diagrams, Belief Networks and Decision Analysis.*
- **R. Neapolitan (1990).** Probabilistic Reasoning in Expert Systems: Theory and Algorithms. John Wiley & Sons, New York.
- **K. G. Olesen, S. L. Lauritzen & F. V. Jensen (1992).** aHugin: A system creating adaptive causal probabilistic networks. In D. Dubois, M. P. Wellman, B. D'Ambrosio, and P. Smets, editors, *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence,* pages 223-229, Stanford, California, July 17-19. Morgan Kaufmann, San Mateo, California.
- **J. Pearl (1988).** *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Mateo, CA.
- **J. Pearl (2000).** Causality: Models, Reasoning, and Inference. *Cambridge University Press, UK.*
- **D. Poole & E. Neufeld (1988).** Sound probabilistic inference in Prolog: An executable specification of influence diagrams.
- **R. Qi (1994).** Decision graphs: Algorithms and applications to influence diagram evaluation and high-level path planning under uncertainty. PhD thesis, Department of Computer Science, University of British Columbia, Canada. *Also Tech. Report 94-27,* Department of Computer Science, University of British Columbia, Canada.
- **H. Raiffa (1968).** *Decision Analysis, Introductory Lectures on Choices under Uncertainty.* Addison-Wesley, Reading, Massachusetts.
- **L. K. Rasmussen (1995(1)).** Bayesian network for blood typing and parentage verification of cattle. *Dina research report no. 38.* Department of Mathematics and Computer Science, Aalborg University, Denmark.
- **L. K. Rasmussen (1995(2)**). BOBLO: an expert system based on Bayesian networks to blood group determination of cattle. *Research report 16.* Research Center Foulum, Denmark, PB 23, 8830 Tjele, Denmark.
- **J. Smith, S. Holtzman & J. Matheson (1993).** Structuring conditional relationships in influence diagrams *Operations Research,* 41(2):280-297.
- **D. J. Spiegelhalter & S. L. Lauritzen.** Sequential updating of conditional probabilities on directed graphical structures. *Networks*, 20(5):579-605, Aug. 1990. Special Issue on Influence Diagrams.
- **P. Spirtes, C. Glymour & R. Scheines (2000).** Causation, Prediction, and Search. *MIT Press*, Adaptive Computation and Machine Learning, second edition.

- **L. Zhang (1993).** A computational theory of decision networks. PhD thesis, Department of Computer Science, University of British Columbia, Canada. *Also Tech. Report 94-8,* Department of Computer Science, University of British Columbia, Canada.
- **S. L. Lauritzen and D. Nilsson. (2001)** Representing and solving decision problems with limited information. *Management Science,* 47(9):1235 - 1251, Sept. 2001.

INDEX