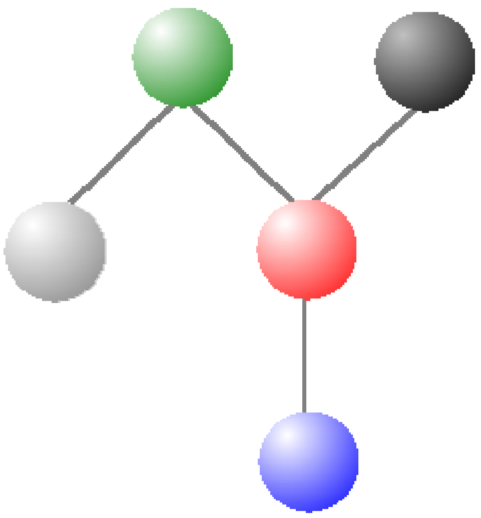


IMPERIAL
COLLEGE

DEPARTMENT · OF · COMPUTING

Bayesian Network Toolkit

4th Year Individual Project

~ 2001 ~

Mark Gowans

Supervised by:

Duncan Gillies

Foreword

The aim of this project is briefly to:

- Produce a toolkit enabling the display and manipulation of Bayesian Networks.
- Utilise the new .BIF standard for representing networks in XML
- Act as a foundation and stimulus for further research into practical extensions of Bayesian Network utilities.

A software toolkit is required to maintain, build and analyse the agreed Bayesian Interchange file format. The software would comprise the following components:

- **Core:** To produce a utility to parse the .BIF file format and to calculate the probability matrices and propagation's contained within its structure.

Building on from the above, the project will take on the following directions:

- **Direction 1:** To produce a graphical tool for the construction and manipulation of .bif files
- **Direction 2:** To enable networks to be constructed from data.
- **Direction 3:** To generate data conforming to a given network.

Each direction will be tied together with the core component and other directions to produce a general toolkit. The project is not be restricted to the above scope and incorporates research into the algorithms required for direction 2 and for optimising generated data sets.

Acknowledgements

I would like to thank the following people for their assistance and inspiration throughout this project:

Duncan Gillies: My project supervisor. It was through Duncan's Intelligent Data course that the idea for this project first came to me. I entered the 4th year with the intention of avoiding as many mathematical based courses as possible. This project, therefore serves as a testament to the excellent quality of teaching offered by Duncan and the department in general as not only did I take the Intelligent Data option, but enthusiastically proposed a project with a core mathematical component. Additionally, throughout the project, Duncan has been a source of inspiration, reacting encouragingly to my ideas and providing a reality-check to ensure the project stayed on the rails throughout its development.

Simon Clarke (sjc97): For teaching me everything I know about Microsoft Visual C++. Without Simon the development of network display code would have been a very much more painful process. My thanks to him for producing the early versions of the required *diagram.ocx* control to my ever-changing specification!

Table of Contents

FOREWORD **2**

ACKNOWLEDGEMENTS **3**

INTRODUCTION **6**

What is a Bayesian Network? 6
Why Bayesian Networks? 6
Motivation for the project 6
What the project is about 7

BACKGROUND **8**

The Dog Barking Problem 8
Required Bayes Theory 9
Initial Propagation *9*
Instantiating Nodes *10*
Propagation Example *11*
Existing Software 13
The FloUE Project *13*
JavaBayes *15*
Microsoft Belief Networks *16*
Belief Network (BN) PowerConstructor *18*
Bayesian Network Representations 19
The .bif File Format *19*
Rival File Formats *22*
The Hepatitis C data source and other suggested file formats 24
The .dfg File Format *24*

SPECIFICATION **26**

Core Component 26
Definition of Terms (Core Component): *26*
Extension 1: Graphical Manipulation and Display of .bif files 27
Explanation of Terms (Extension 1): *27*
Extension 2: Generation of Bayesian Networks from Data 28
Extension 3: Generation of Data from a Bayesian Network 29
Notes and Limitations: 29

IMPLEMENTING A SOFTWARE SOLUTION **30**

Selecting a Language 30
Java *30*
(Visual) C++ *31*
Visual BASIC *31*
Learning Algorithms 35
Naïve Bayes *35*
Cheng, Bell, Liu 3 Parse Algorithm *36*
The Spanning Tree Algorithm *37*
Generating Data 39
Test 1: Instantiating Root Nodes *39*
Test 2: Instantiating All Non-Root Nodes *40*
Test 3: Instantiating All Nodes *40*
Test 4: Instantiating 3 Nodes *40*
Test 5: Instantiating All Non-Root Nodes with 1000 lines of data *41*
Moving out of Access – Prototyping in Action 41
Modifying the Network 43
Additional Testing 44
File Handling *44*
Further testing of the Learning Algorithm *44*
Testing Instantiations *45*

CONCLUSIONS AND EVALUATION **46**
EVALUATION WITH REGARD TO THE SPECIFICATION 46
Core Component *46*
Extension 1 .... *47*
Extension 2 .... *48*
Extension 3 .... *49*
EVALUATION AS A PIECE OF SOFTWARE… 49
The Bayesian Network Toolkit Vs. *50*
BNT as a teaching aid *50*
EVALUATION OF SOFTWARE DESIGN PROCESS 51
FINAL WORDS… 51
THE FUTURE **52**
MULTIPLY CONNECTED NETWORKS 52
IMPROVED FILE FORMAT SUPPORT 52
Read / Write Formats *52*
Data Generator Formats *52*
IMPROVED PROBABILITY / STATE DISPLAY 53
HIDDEN NODE INVESTIGATION 53
IMPROVED HELP ET AL 53
TEACHING AID IMPROVEMENTS 53
MDL METRIC OR OTHER MEASURE OR FIT 54
BIBLIOGRAPHY AND LINKOLOGY **55**
APPENDIX A - EXAMPLE FILES **56**
.BIF REPRESENTATION OF THE DOG BARKING PROBLEM 57
0.3 Compliant *57*
0.1(5) Compliant *58*
.DSC REPRESENTATION OF THE DOG BARKING PROBLEM 59
APPENDIX B - USER DOCUMENTATION **61**
APPENDIX C - TESTING **73**
TESTING OF THE DATA GENERATOR 74
Test 1: Instantiating Root Nodes *74*
Test 2: Instantiating All Non-Root Nodes *74*
Test 3: Instantiating All Nodes *74*
Test 4: Instantiating 3 Nodes *74*
Test 5: Instantiating All Non-Root Nodes with 1000 lines of data *74*
APPENDIX D - HEPATITIS C INVESTIGATION RESULTS **75**
BAYESIAN NETWORK REPRESENTATION OF THE DATASET 76

Introduction

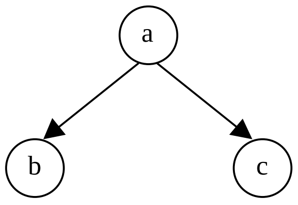
Many practical tasks can be reduced to the problem of classification. Fault diagnostics, pattern recognition, forecasting are but 3 examples. Learning and using these classifiers is considered a hot topic amongst data mining groups, research councils, in Artificial Intelligence applications and in many other interesting communities. A Bayesian network helps tackle the problem of classification.

What is a Bayesian Network?

Bayesian networks are a combination of two different mathematical areas: graph theory and probability theory. It is a representation of a joint probability distribution defined on a finite set of discrete random variables.

The representation is a directed acyclic graph consisting of nodes, which correspond to random variables and arcs, which correspond to probabilistic dependencies between the variables.

A conditional probability distribution is associated with each node and describes the dependency between the node and its parents.



The diagram to the left illustrates an example Bayesian Network. It is a directed acyclic graph since each arc is directed and there are no loops. It contains 3 nodes (discrete random variables), *a*, *b* and *c*. From it we can infer that *b and c are conditionally* ***dependant*** *on a.* Similarly we can note that *b is conditionally* ***independent*** *from c*. I.e.

$$P(a \mid b , c) = P(a \mid b).$$

The joint probability distribution of all the variables is therefore:

$$P(a , b , c) = P(b \mid a) P(a) P (c \mid a)$$

But let us not get bogged down in the mathematics at this stage…

Why Bayesian Networks?

When used in conjunction with statistical techniques, a graphical model such as a Bayesian network, has several advantages for data analysis.

One, because the model encodes dependencies among all variables, it readily handles situations where some data entries are missing.

Two, a Bayesian network can be used to learn causal relationships, and hence can be used to gain understanding about a problem domain and to predict the consequences of intervention.

Three, because the model has both a causal and probabilistic semantics, it is an ideal representation for combining prior knowledge (which often comes in causal form) and data.

Four, Bayesian statistical methods in conjunction with Bayesian networks offer an efficient and principled approach for avoiding the overfitting of data.

Motivation for the project

The Bayesian Network Toolkit was a self-proposed project inspired by the 4th year Intelligent Data and Probabilistic Inference course offered at Imperial.

It provides both a demanding implementation challenge as well as including scope for interesting and potentially groundbreaking research.

What the project is about

We have seen that Bayesian Networks are an established method for analysing data and a convenient way to represent relationships and dependencies between variables in a given scenario. It is surprising therefore that so little exists in the way of software products that utilise this theory.

The purpose of this project is split into 3 main goals.

It is hoped that a usable utility will be authored to enable people to explore and use Bayesian Networks in a practical way. It is paramount that the toolkit isn't *just another software offering*, instead it must succeed as a stand-alone product as well as offering practical advantages over existing products targeting the same user base. In order to begin to achieve this goal, the software must:

- Fill a niche in the Market. To build an application offering features and functionality not offered by existing Bayesian Network software products.
- Address the shortcomings, inaccuracies and limitations present in existing Bayesian Network products.

Additionally, a standard XML representation for Bayesian Networks has been proposed and agreed. However, no existing software currently supports the format, so this project will be designed to utilise the new standard.

The second goal of the project is to produce a utility that can be utilised as a teaching aid. It will be focussed toward the 4th Year Intelligent Data course offered at Imperial, but should in no way should this be a restriction. In order to fulfil this goal, the software will be tied closely into theory taught in the course wherever possible. Test cases and examples will utilise material and data sets, such as the Hepatitis C data, familiar to students taking that option.

As well as producing a software product, the 3rd aim of this project is that of researching further useful tools that can accompany such a product. The proposed data generator (See *Specification*) is a totally original concept not present in any existing software product[1] designed for use with Bayesian Networks. Research is often hindered by unreliable data, caused by freak values, data corruption, etc. Often, what little reliable data exists is only available at high financial cost. It is therefore hoped that this aspect to the project will be able to reliably expand data sets with potential uses and cost savings across all disciplines.

More formally, the project can be broken up into 4 distinct components: a core module and three extensions. Each section and its individual targets are discussed in detail in the *Specification*.

A further discussion of the theory and motivation behind the project can be found in the following section – *Background*.

[1] Reviewed or that could be found at the time of writing.

Background

The Dog Barking Problem

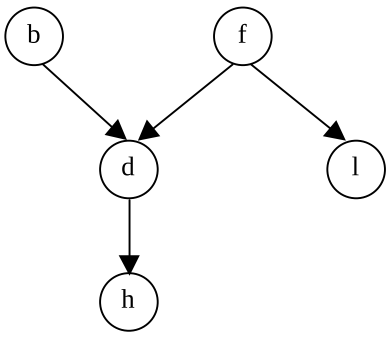
The Dog Barking Problem is the canonical example used when discussing Bayesian Networks. It was originally published by E. Charniak, in Bayesian Networks without Tears, AI Magazine, 1991. Throughout this document we will refer back to this problem so it is important to be aware of its salient points.

We have a world where we only care about a dog and its owners. The dog is known to suffer now and again from bowel problems, but as an outsider we're not sure exactly when it suffers from this condition. Often when the dog has bowel problems, the owners leave it outside. But equally so do they sometimes when they go out. We are able to notice or would perhaps like to predict the following observations:

- Whether the dog has bowel problems or not
- If the owners are at home or if they are out.
- Whether the dog is inside the house or out in the yard.
- If the lights are on in the house
- Whether we can hear the dog barking.

By observing some of the above facts we would like to know what the state of the other observations would most likely be. For example, we might note that the lights are on inside the house and we can hear the dog barking. From this we might like to deduce whether we think the dog is inside the house or not.

Clearly there is some relationship between each of our observations in the toy world. We discussed one such relationship above in noting that the dog is often out when it has bowel problems or its owners are out. The other such relationships are clearest represented graphically in the form of a Bayesian Network as pictured below:



Legend:

b	- bowel-problems
f	- family-out
d	- dog-out
l	- lights-on
h	- hear-bark

Each state is Boolean and can take either *normal* or *abnormal* states. The probabilities and relationships are described in the table below.

	P(f)	P(b)	P(l|f)	P(l|!f)	P(d|f,b)	P(d|f,!b)	P(d|!f,b)	P(d|!f,!b)	P(h|d)	P(h|!d)
Normal	*0.15*	*0.01*	*0.60*	*0.05*	*0.99*	*0.90*	*0.97*	*0.30*	*0.70*	*0.01*
Abnormal	*0.85*	*0.99*	*0.40*	*0.95*	*0.01*	*0.10*	*0.03*	*0.70*	*0.30*	*0.99*

Required Bayes Theory

In order to calculate the correct probabilities at each state of each node we use a process of Lambda and Pi message propagation for singly connected networks. They theory and algorithms behind this are discussed below by example, referenced to the dog problem network:

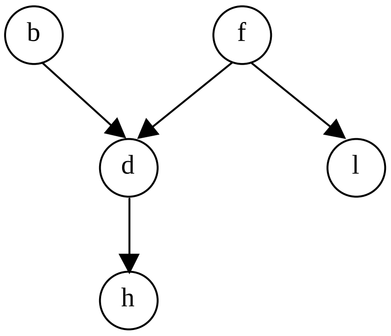
Initially we are given the prior probabilities of the root nodes and link matrices expressing the probabilities of child nodes in terms of conditional probabilities to their parents.

When loading the network we must determine the probabilities for each node / state which can be done trivially using the following algorithm:

Initial Propagation

1. All λ and π messages and normalised[2] evidence at each node is set to 1.

2. The root node state probabilities are given to us.



3. In the dog problem, the prior probabilities for the root nodes (*b* and *f*) would therefore be set as:

	P(f)	P(b)
Normal	*0.15*	*0.01*
Abnormal	*0.85*	*0.99*

4. Next we are able to calculate the node/state probabilities for each of the direct children of the root nodes using the link matrices. We are able to do this by means of *propagating* π messages.

The mathematical formula for a π message is given as:

The π message

For 1 Parent: π message at *c* in state *i* with parent *p* =

$$\sum_{j=1}^{\text{No of states of } p} P(c_i \mid p_j)\, P(p_j)$$

For Multiple Parents: π message at *c* in state *i* with parents *a..n* =

$$\sum \sum .. \sum_{j,k..z=1}^{\text{No of states of a..n}} P(c_i \mid a_j \,\&\, b_k \,\&\, \ldots \,\&\, n_z)\, P(a_j)\, P(b_k) \,..\, P(n_x)$$

5. The probability for each state of the node is simply the normalised sum of the π messages.

For the initial propagation's – this may be easier expressed in words:

The initial probabilities for a node with 1 parent: For a node *c* in state *i* we simply have:

The probability = The normalised sum over all the parent states of: the conditional probability (from the link matrix) of *c* in state *i* given the state of the parent node, multiplied by the initial probability of the parent node.

[2] Evidence at each node need not be normalised, although for the purposes of the utility, the code may be simplified by ensuring that it is.

In our example, therefore, the initial probability for node *l* being in state *i* would be:

$$\text{Normalised}\left(P\left(l_i \mid f_1\right) P\left(f_1\right) + P\left(l_i \mid f_2\right) P\left(f_2\right)\right)$$

The initial probabilities for a node with multiple parents: For a node *c* in state *i* we have:

An expanded form of the base case above except we are summing over each state of **each** parent.

In our example this would relate to dog_out (node *d*). The probability of node *d* being in state *i* would therefore be:

$$\text{Normalised}\left(P\left(d_i \mid f_1 \,\&\, b_1\right) P\left(f_1\right) P\left(b_1\right) + P\left(d_i \mid f_1 \,\&\, b_2\right) P\left(f_1\right) P\left(b_2\right) + P\left(d_i \mid f_2 \,\&\, b_1\right) P\left(f_2\right) P\left(b_1\right) + P\left(d_i \mid f_2 \,\&\, b_2\right) P\left(f_2\right) P\left(b_2\right)\right)$$

6. Next we are able to repeat the process from stage 4 propagating down the network in the same way until all nodes have been calculated.

Once all the initial calculations are complete we are able to present the network to the user of the system ready for instantiation.

Instantiating Nodes

When a node is instantiated into one of its states:

1. The lambda evidence for that state is set to 1.

2. The probability for that state is set to 1.

3. The probability for every other state of that node is set to 0.

4. λ and π messages are sent to its parents and children respectively.

π messages are of the same form as with the initial propagation.

λ messages are represented by the formula:

The λ message

For 1 Parent: λ messages at *P* in state *k* from child node *c* =

$$\sum_{j=1}^{\text{No of states of } c} P(c_j \mid a_k)\, \lambda(c_j)$$

Note: $\lambda(c_j)$ = lambda **evidence** for node *c* in state *j*. For instantiated nodes this will equal 1 for the instantiated state and 0 for all others.

For multiple Parents: λ messages to *P* in state *k* from child node *c* =

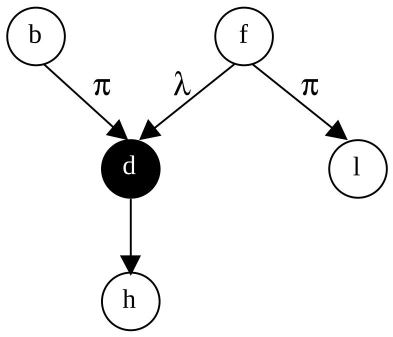
$$\sum_{i=1}^{\text{No of States of } P} p(p_k) \sum_{j=1}^{\text{No of states of } c} P(c_j \mid p_k \,\&\, a_i \,\&\, \ldots \,\&\, n_i)\, \lambda\,(c_j)$$

Note: $p(p_k)$ is the **pi evidence** for *p* in state *k*, 1 if instantiated 0 if instantiated in some state other than k.

5. The message propagation's continue as from stage 4 recursively but compliant to the following rules

Propagating Messages

1. π messages cannot be propagated **through** instantiated nodes. I.e.



Suppose at some time in the past we instantiated *dog_out.* Now if we instantiate *bowel_problem*, we send a π message to d, but not further on to h. Similarly if *d* had more children than just *h* **-** π messages would not be sent to **any** of the child nodes of d.

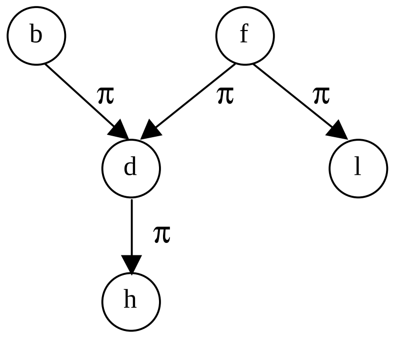
2. λ messages are **not** blocked by instantiation as can be seen in the above example. They are, however, blocked by nodes at converging paths with no λ evidence. I.e. if *dog_out* or and child of *dog_out* were not instantiated, then no λ message would be sent to f.

Let us illustrate these principles with the aid of an example:

Propagation Example

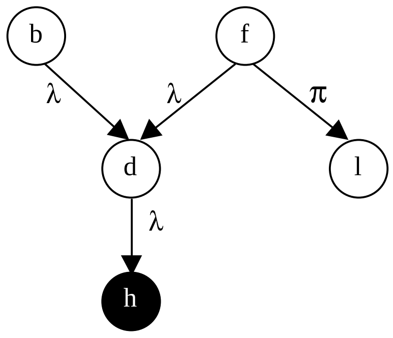
For this example we will use the familiar dog problem network.

1. Initially we must propagate all the π messages to set-up the network.



Note: No λ message is sent from *d* to *b* or *f* since there it is blocked by a lack of λ evidence.

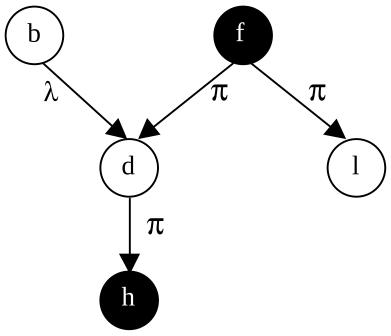
2. If we now instantiate *hear_bark*:



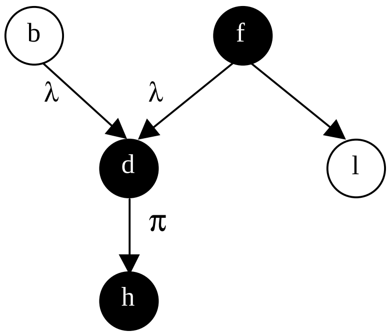
- A λ message is sent from *h* to *dog_out*
- The posterior probability of *d* is recalculated based on the new evidence
- Messages are sent to all relations **except back to *h*.**
- *b, f* and *l* all recalculate their probabilities
- *f* sends a π message on to *light_on*, but nothing **back** to *dog_out.*
- *l* recalculates its probability.

3. Suppose now that we wish to instantiate *family_out:*

- *f* sends π messages to *dog_out* and *light_on*
- *l* and *d* recalculate their probabilities.
- Since d has λ evidence (from *hear_bark*) it sends a λ message up to *bowel_problem* as well as a π message to *hear_bark*
- Since *h* is already instantiated it does nothing.
- *b* recalculates its probability based on the message from *d.*



4. If we finally instantiate *dog_out:*



- λ messages are sent to *family_out* and *bowel_problem*. *A* π message is sent to *hear_bark*.
- *h* and *f* remain unchanged by the new evidence but *b* is recalculated.
- *f* does not send a π message to *light_on* since the propagation's are blocked by *f's* instantiation.

Existing Software

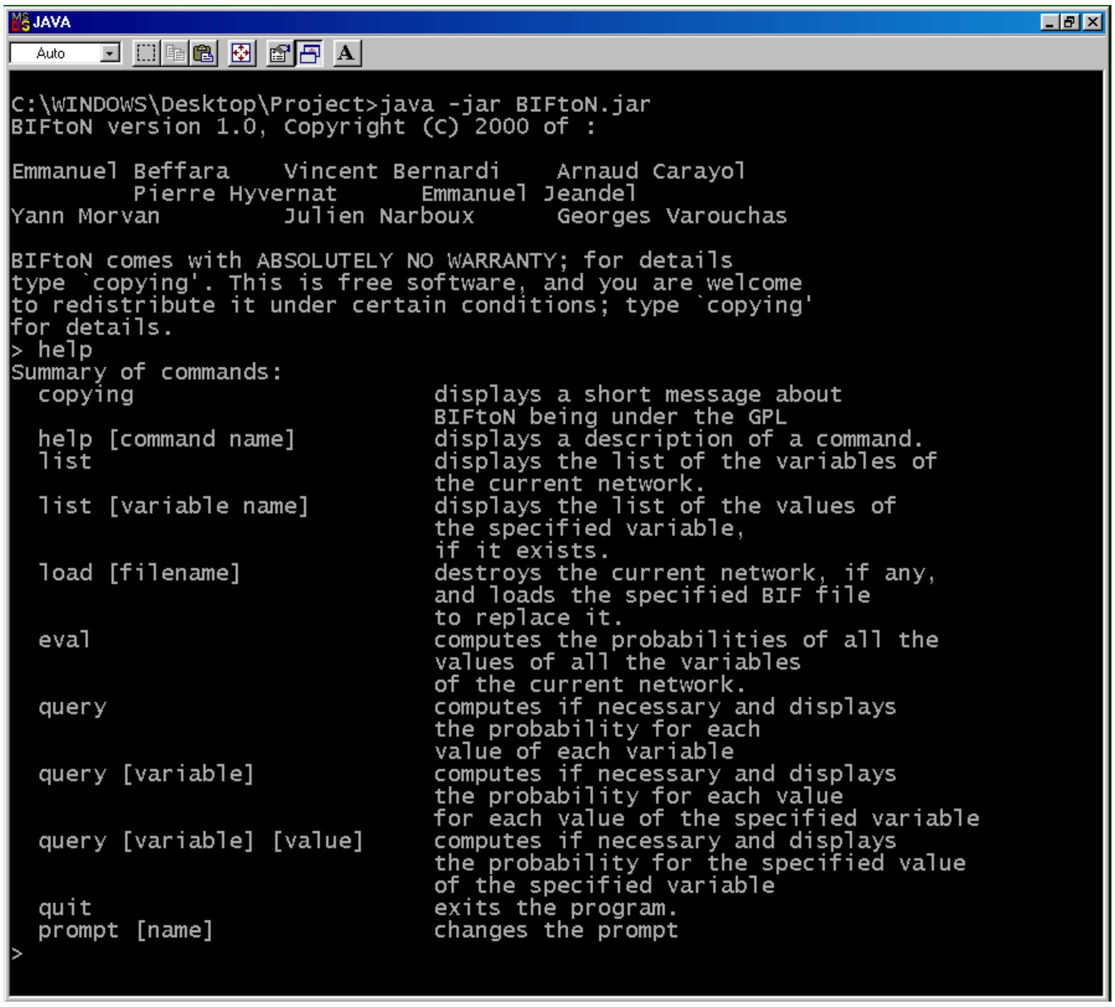
The FloUE Project

http://www.ens-lyon.fr/~jnarboux/Floue/

The purpose of the FLoUE project is to develop a program for evaluating Bayesian networks.

The complete software package is available for download at the above address, although no accompanying source-code has been provided. It would appear as though no work has been done on the project or site for quite some time and although the downloadable software works well, it is very limited in its functionality. The web site is widely linked to from the Bayesian eCommunity

Invoked by running the BIFtoN.jar file, the FloUE software uses a command line interface. (See below)



```
C:\WINDOWS\Desktop\Project>java -jar BIFtoN.jar
BIFtoN version 1.0, Copyright (C) 2000 of :

Emmanuel Beffara    Vincent Bernardi    Arnaud Carayol
         Pierre Hyvernat       Emmanuel Jeandel
Yann Morvan          Julien Narboux      Georges Varouchas

BIFtoN comes with ABSOLUTELY NO WARRANTY; for details
type `copying'. This is free software, and you are welcome
to redistribute it under certain conditions; type `copying'
for details.
> help
Summary of commands:
  copying                       displays a short message about
                                BIFtoN being under the GPL
  help [command name]           displays a description of a command.
  list                          displays the list of the variables of
                                the current network.
  list [variable name]          displays the list of the values of
                                the specified variable,
                                if it exists.
  load [filename]               destroys the current network, if any,
                                and loads the specified BIF file
                                to replace it.
  eval                          computes the probabilities of all the
                                values of all the variables
                                of the current network.
  query                         computes if necessary and displays
                                the probability for each
                                value of each variable
  query [variable]              computes if necessary and displays
                                the probability for each value
                                for each value of the specified variable
  query [variable] [value]      computes if necessary and displays
                                the probability for the specified value
                                of the specified variable
  quit                          exits the program.
  prompt [name]                 changes the prompt
>
```

Screen 1: The FLoUE Project interface

The above screenshot of the FloUE summarises all the commands available to the end user. The software will read *.bif* (not XML) files using the load command, although the authors note that the supported format may deviate from other .bif formats used by other available applications. A complete discussion of the required file format is available from the user documentation page (see link above).

Using the software with the dog barking problem:

Once loaded, the individual nodes may be viewed using the list command:

```
> list
family-out bowel-problem hear-bark dog-out light-on
```

The individual states of each node can then be examined with the list [variable] command:

```
> list hear-bark
true false
```

The probabilities of each node can be viewed using the query command to display either all the probabilities or, with the optional parameters, the individual node probabilities.

```
> query hear-bark
true: 0.2831227
false: 0.7168773
```

The provided file BIFLint.jar acts as a parser for the *.bif* format returning a simple boolean value as to whether the file is compliant or not.

No other functionality is provided as working.

A selection of *valid .bif* files for use with the software have been provided and are linked to from the project home page. User documentation has also been provided on the site, although offers little more help / information than this document.

Although very simple in its functionality – the software forms a good benchmark for the core component of the utility to be authored. The user interface is simple to use and implements much of the required functionality of the core. The supplied parser may be of use in the project, even if used as-is. Error trapping has not been developed in FloUE. Entering invalid commands sensibly produces an *invalid command* error, yet invalid parameters supplied to a valid command cause the program to crash with an unhandled exception.

It seems strange that the program should be authored in Java and freely available, yet is distributed without source-code. As a consequence of this – the software should be used as a benchmark for the functionality / usability of the core component of this project only as opposed to being a valuable source of information.

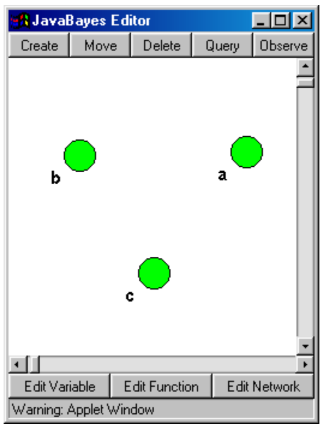
JavaBayes

http://www.cs.cmu.edu/~javabayes/Home/index.html

JavaBayes *is a system that handles Bayesian networks: it calculates marginal probabilities and expectations, produces explanations, performs robustness analysis, and allows the user to import, create, modify and export networks.*

The authors claim "JavaBayes *is the first full implementation of Bayesian networks in Java.*" and include a brief discussion of the merits of using Java as an implementation language. 2 Versions of the software exist, a *lite* web-applet and the fully functional desktop application. Both are distributed freely and including source-code under the GNU license.

Using the web applet

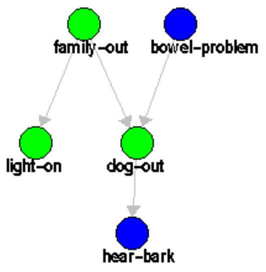


The web applet is designed to replicate all the functionality of the *editor* of the main software component. Its use and operation is fully documented from the project home page, although in reality it appears to be only partially functional.

As can be seen from the screenshot on the left, 3 nodes have been created in a new network being edited in the JavaBayes editor. The documentation describes how links can be created between nodes and the how probabilities can be assigned etc. although this appears not to work in practice.

The web applet offers no support for opening or saving work. In terms of a piece of software the applet is of little use for anybody wishing to conduct serious research into Bayesian nets. The point and click interface however provides a good foundation for the development of *Extension 1* in this project.

Using the main software



Again, although provided fully documented and with source, it would appear as though the supplied files do not compile or run. It is claimed that the software will open all standard versions of the *.bif* format including the new XML representation to be used by this project. As such, the supplied code may be of use when designing the core component of the utility.

The diagram to the right is provided as a screenshot[3] of the main application representing the *dog-barking problem*. In theory the main application could be used as a benchmark for the *Core* and *Extension 1* components of this project. However, no facility has been provided for saving back to file once a network has been created or modified. According to the documentation it would appear as though the utility uses extra comment fields in the *.bif* files to place the nodes on-screen as opposed to calculating positions at run-time. As an effect of this – the utility may not be able to display networks created using a different program – even if they are fully *.bif* compliant.

[3] Taken from the Using JavaBayes documentation: http://www.cs.cmu.edu/~javabayes/Home/node6.html#SECTION00610000000000000000

Microsoft Belief Networks

http://www.research.microsoft.com/msbn/default.htm

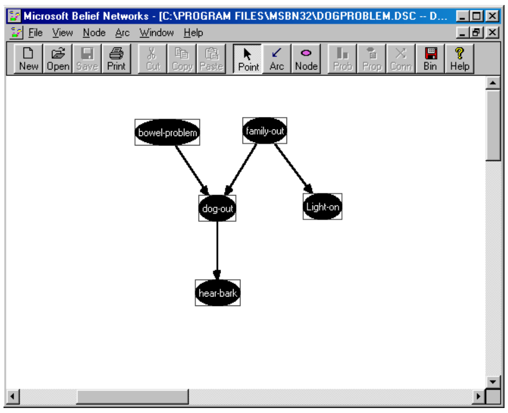
*Members of the Decision Theory & Adaptive Systems Group within Microsoft Research have created a Windows application called **MSBN32.EXE** which allows the creation, assessment and evaluation of Bayesian belief networks.*

The software is freely downloadable from the above link although is subject to Microsoft's non-commercial licensing. Supplied as a compiled Win32 executable, the accompanying DLL with associated API is also provided offering the following functionality:

- loading and storing of belief networks in textual form
- creation and modification of networks through the addition of nodes and arcs assessment of discrete probabilities
- evaluation of belief networks using exact clique-tree propagation methods
- decision-theoretic troubleshooting and recommendations
- asymmetric assessment single-decision influence diagrams

As such, the above functionality can be implemented directly, using the Microsoft engine, in Visual Basic 4.0, C or C++. Full documentation of the API is provided with the standard download. It should be noted, however, that the API is highly optimised for Microsoft's *.dsc* file format and so would require a great deal of manipulation in order to deal with the *.bif* standard.

Using the software:


Microsoft Belief Networks - [C:\PROGRAM FILES\MSBN32\DOGPROBLEM.DSC -- D...
File View Node Arc Window Help
New Open Save Print Cut Copy Paste Point Arc Node Prob Prop Conn Bin Help
bowel-problem
family-out
dog-out
Light-on
hear-bark

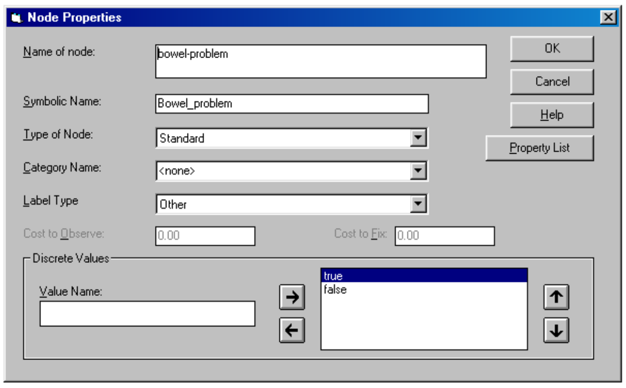
A sample application is provided to demonstrate the use of the API (pictured left). It is able to load and save .dsc files as well as interfacing with Microsoft's JET engine. As such, networks can be created and manipulated independently to the application using Microsoft Access or similar.

The software, pictured here displaying the *dog barking problem* network implements a simple point and click interface. Nodes have to be manually placed, although the length of their name determines their size. Relationships between nodes are easy to add and a "snap to grid" style interface is provided. Moving nodes with relationships automatically updates the link-lines as required.

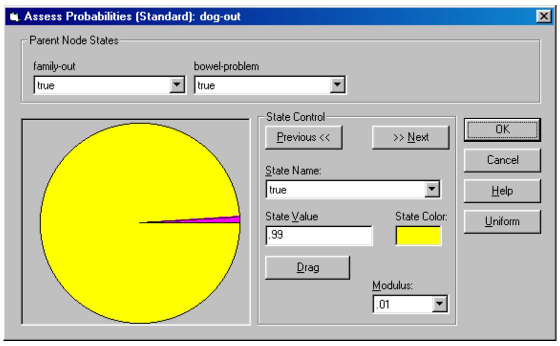
The properties of a node can be accessed by a simple right click and offers the following functionality:

Each node is allowed to take 0 or more 'Discrete Values' (States) – each of which can be assigned a unique meaningful name. Each node may be assigned to be a *Standard* node or causally independent.

Probability assignments are similarly accessed via the right click (see figure over). Pull down boxes allow the user to select between each node state and of the states of the parent nodes. Once all the required probability assignments have been made the node turns from red to white on the main screen.



Name of node: bowel-problem
Symbolic Name: Bowel_problem
Type of Node: Standard
Category Name: <none>
Label Type: Other
Cost to Observe: 0.00
Cost to Fix: 0.00
Discrete Values
Value Name:
true
false



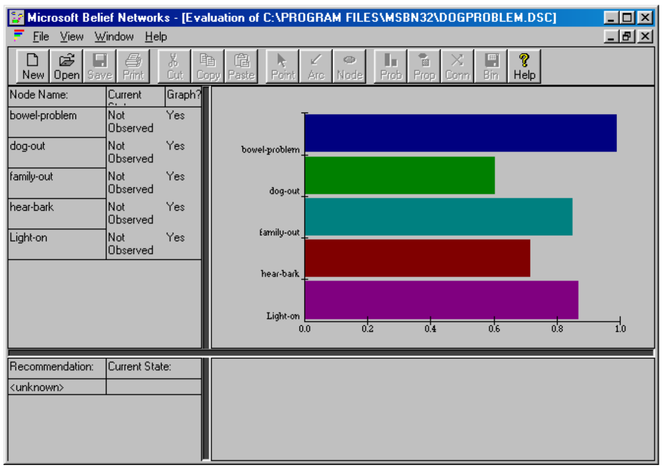
Parent Node States
family-out: true
bowel-problem: true

State Control
State Name: true
State Value: .99
State Color: 🟨
Modulus: .01

Once the network is complete the *evaluate* option brings up the screen below to enable the network to be queried.

The bar chart clearly shows the probabilities of each of the nodes in either each of their states or in their designated 'abnormal' state (as pictured below).

Nodes may be added to or removed from the graph. Their state may be *not observed* or set to a specific value. In this way it is easy to examine the effects of various observations. For example, setting the node *hear-bark* to *true* – it is immediately obvious that the most likely *cause* for this phenomenon is *bowel-problem*.



Node Name:	Current St...	Graph?
bowel-problem	Not Observed	Yes
dog-out	Not Observed	Yes
family-out	Not Observed	Yes
hear-bark	Not Observed	Yes
Light-on	Not Observed	Yes

Recommendation:	Current State:
<unknown>	

The interface is generally easy to use and the software as a whole forms an excellent benchmark for *Extension 1* of this project.

Unfortunately as MSBN is unable to read/write to the standard *.bif* format and offers no facility to learn from data it falls far short of the serious research tool that is the intended outcome of this project.

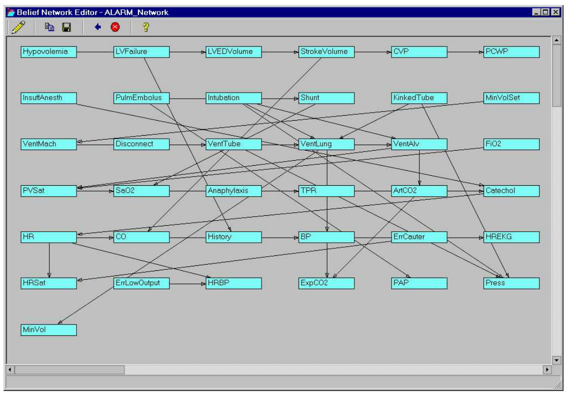
Belief Network (BN) PowerConstructor
http://www.cs.ualberta.ca/~jcheng/bnpc.htm

This appears to be the only freely available tool to construct Bayesian networks from a data source. The utility connects to any ODBC compliant database or will read in from a text file. Although distributed without source code, a DLL and API is provided which *implements Chow and Liu's tree construction algorithm [Chow and Liu, 1968]* – ready for use with Visual Basic or similar.

Running the software:

The software includes only 1 executable, which once run takes the form of a *wizard*, guiding the user through each stage of the network construction process. Although the software appears not to work, crashing out with unexplained errors, the supplied documentation describes each of the steps as follows:

1. Select the database format of your data set.
2. Enter the database name of a desktop database or enter the connection information to the database server if ODBC (Remote) was chosen in Step 1.
3. Select the data set from your database and the fields of the data set that you want to include.
4. Enter the domain knowledge. Complete ordering and Partial Ordering of the nodes may be specified as may be and causes and effects. The user is also able to stipulate and relationships that must not be formed during the network constructions
5. Starts the construction process…
6. Finally the finished network may be viewed as pictured here.



Under stage 6 – the software makes no attempt to "tidy" the graph, instead opting for regular ordering of the nodes with lines representing relationships as applicable. Once compiled, the network may be exported as: (*.dne or *.dnet), Hugin format (*.net) and .bif (not XML).

Although offering much of the functionality to be implemented by *Extension 1* of the project – the BN software has some shortcomings, which this project intends to address. Namely:

- Only 1 inference algorithm has been implemented and no facility is provided to allow the user to tweak it during or after the construction process.
- Once constructed there is no facility within the same application to reason about the network or view the associated probabilities.
- No measure of fit is provided between the data and the produced network.

The user interface provided by the software is extremely user friendly and would form a good basis for *Extension 1*. In order for *Extension 1* to be deemed useful – it should at least implement all the functionality provided by this software.

The accompanying web-site provides a useful resource for starting a similar project and includes references and links to each component of the software.

Bayesian Network Representations

The .bif File Format

Background

Following the AUI conference in 1996 the .bif file format was used and began to be widely used by many of the research projects into Bayesian Networks. However many shortcomings were discovered with this file format and following the 1998 conference an XML version of the .bif file format was proposed. An idea originally outlined by Fabio Cozman[4] of CMU and key members of the GeNie project[5], it has become the international standard for the representation of Bayesian Networks.

Despite being an international standard only few of the existing Bayesian products are able to interface with the file format. This project will use version 0.3 of the xmlbif format as its primary input file format.

Format of the File

More in-depth information into the workings of the file format can be found at: http://www.cs.cmu.edu/afs/cs/user/fgcozman/www/Research/InterchangeFormat/ although, for the purposes of the utility, this document should provide all the required information to complete the project.

Version 0.3 of the file discussed below is fully XML[6] compliant according to the w3c standards and as such can be parsed by any readily available XML parser.

The mandatory[7] XML header of the file is:

```
<?xml version="1.0"?>
```

The file format is described in the following DTD:

```
<!DOCTYPE BIF [
  <!ELEMENT BIF ( NETWORK )*>
    <!ATTLIST BIF VERSION CDATA #REQUIRED>
  <!ELEMENT NETWORK ( NAME, ( PROPERTY | VARIABLE | DEFINITION )* )>
  <!ELEMENT NAME (#PCDATA)>
  <!ELEMENT VARIABLE ( NAME, ( OUTCOME | PROPERTY )* ) >
    <!ATTLIST VARIABLE TYPE (nature|decision|utility) "nature">
  <!ELEMENT OUTCOME (#PCDATA)>
  <!ELEMENT DEFINITION ( FOR | GIVEN | TABLE | PROPERTY )* >
  <!ELEMENT FOR (#PCDATA)>
  <!ELEMENT GIVEN (#PCDATA)>
  <!ELEMENT TABLE (#PCDATA)>
  <!ELEMENT PROPERTY (#PCDATA)>
]>
```

[4] Fabio Cozman: www.cs.cmu.edu/~fgcozman/home.html
[5] The GeNie project: www2.sis.pitt.edu/~genie
[6] See: www.w3c.org/XML/ for further information on XML.
[7] Only the *version* field is mandatory, but other attributes described in the XML standards are also permissible.

Since DTD is a proprietary language and is not XML compliant, many freely available XML parsers do not accept it as a valid definition. Therefore the w3c compliant schema equivalent to the above DTD follows for reference purposes:

```
<!DOCTYPE xsd:schema PUBLIC "-//W3C//DTD XMLSCHEMA 19991216//EN" "" [
  <!ENTITY % p 'xsd:'>
  <!ENTITY % s ':xsd'>
]>
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <xsd:element name="BIF">
    <xsd:complexType content="elementOnly">
     <xsd:sequence minOccurs="0" maxOccurs="unbounded">
       <xsd:element name="NETWORK" type="NETWORKType"/>
     </xsd:sequence>
     <xsd:attribute name="VERSION" type="xsd:string" use="required"/>
     <xsd:attribute name="xmlns:xsi" type="xsd:uriReference"
use="default" value="http://www.w3.org/1999/XMLSchema-instance"/>
     <xsd:attribute name="xsi:noNamespaceSchemaLocation"
type="xsd:string"/>
     <xsd:attribute name="xsi:schemaLocation" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
    <xsd:complexType name="DEFINITIONType" content="elementOnly">
     <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="FOR" type="xsd:string"/>
      <xsd:element name="GIVEN" type="xsd:string"/>
      <xsd:element name="TABLE" type="xsd:string"/>
      <xsd:element name="PROPERTY" type="xsd:string"/>
     </xsd:choice>
    </xsd:complexType>
    <xsd:complexType name="NETWORKType" content="elementOnly">
     <xsd:sequence>
      <xsd:element name="NAME" type="xsd:string"/>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="PROPERTY" type="xsd:string"/>
        <xsd:element name="VARIABLE" type="VARIABLEType"/>
        <xsd:element name="DEFINITION" type="DEFINITIONType"/>
      </xsd:choice>
     </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="VARIABLEType" content="elementOnly">
     <xsd:sequence>
      <xsd:element name="NAME" type="xsd:string"/>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="OUTCOME" type="xsd:string"/>
        <xsd:element name="PROPERTY" type="xsd:string"/>
      </xsd:choice>
     </xsd:sequence>
     <xsd:attribute name="TYPE" use="default" value="nature">
      <xsd:simpleType base="xsd:NMTOKEN">
        <xsd:enumeration value="nature"/>
        <xsd:enumeration value="decision"/>
        <xsd:enumeration value="utility"/>
      </xsd:simpleType>
     </xsd:attribute>
    </xsd:complexType>
</xsd:schema>
```

The following tags must encompass each collection of networks in a .bif file:
`<BIF VERSION="0.3">` {Network(s)} `</BIF>`

The Network Tags must encompass each network within a collection of BIF format networks:
`<NETWORK>` Network Description `</NETWORK>`

The elements described in the DTD should be used as follows:

Network

- **Name:** A freeform text field to meaningfully describe the network
- **(Property):** Zero or more freeform text fields to describe attributes of the network if required.

E.g.

```
<NETWORK>
      <NAME>Network Example</NAME>
      <PROPERTY>date 05 January 2000</PROPERTY>
      <PROPERTY>author Mark Gowans</PROPERTY>
      …
</NETWORK>
```

Variable

- **Name:** A freeform text field to meaningfully describe the variable (node).
- **(Property):** As For Network
- **Outcome:** One or more freeform text fields to **uniquely** identify state names that the variable may take.

E.g.

```
<VARIABLE TYPE = “nature”>[8]
      <NAME>Variable Example</NAME>
      <PROPERTY>A simple example</PROPERTY>
      <OUTCOME>State 1</OUTCOME>
      <OUTCOME>State 2</OUTCOME>
      <OUTCOME> …
      …
</VARIABLE>
```

Definition

Used to describe the probabilities associated with a given variable.

- **For:** The name of a previously defined *Variable*[9] to which the probability block is relevant.
- **(Given):** Zero or more names of previously defined *Variables* which the current *For Variable* is conditional upon.
- **(Property):** As for Network.
- **Table:** The probabilities expressed as decimal percentages (E.g. 10% = 0.1). Probabilities are listed separated by white space. The probabilities are listed by enumerating the **Given** variables first in the order they are listed and then the **For** variables.

E.g. (taken from F.G.Cozman’s web page)

```
<DEFINITION>
    <FOR>GasGauge</FOR>
    <GIVEN>BatteryPower</GIVEN>
    <GIVEN>GasInTank</GIVEN>
    <TABLE>1.0 0.0 0.0 1.0 0.2 0.8 0.0 1.0 </TABLE>
</DEFINITION>
```

[8] The TYPE attribute must be the keyword either *nature, decision* or *utility*.

[9] Note this condition cannot be stipulated in the DTD but **must** be enforced.

The *GasGauge*, *BatteryPower* and *GasInTank* variables are all boolean. Let us abbreviate them to GG, BP and GIT respectively. Assuming the convention that !<name> indicates the false boolean state of a variable (i.e. P(GG & !GG) = 0) then the probabilities listed in the above table tag represent (in order as they appear):

P(GG | BP & GIT) P(!GG | BP & GIT) P(GG | BP & !GIT) P(!GG | BP & !GIT)
P(GG | !BP & GIT) P(!GG | !BP & GIT) P(GG | !BP & !GIT) P(!GG | !BP & !GIT)

An example .bif file is located in *Appendix A*.

Rival File Formats

.bif (not XML)

Previous versions of the bif format are still widely used throughout the Bayesian community. It is based on the original proposals for a standardised file format by Microsoft who proposed a .BNIF format.

2 Versions of the original .bif format were produced, 0.1 and 0.15. For the purposes of this document we shall focus on revision 0.15 as it is the most widely used. Information on 0.1 of the format can be found at: http://www.cs.cmu.edu/~javabayes/Home/node7.html#SECTION00740000000000000000

Version 0.15 of the .bif format is a human readable text file, formatted using the constraints as described below:

The basic unit of information is a *block*: a piece of text, which starts with a keyword and ends with the end of an attribute list (see below). Text appearing outside a block is regarded as a comment.

Blocks: are formatted as follows:

```
block-type block-name {
        attribute-name  attribute-value;
        attribute-name  attribute-value;
        attribute-name  attribute-value;
}
```

Semi-colons are mandatory after each attribute line. *Block-name* is a free form text string contained within quotes (")

Block-types:

Network: Describes the network and defines its name. *Property* is the only valid attribute-name and may take a free form text to describe properties of the network. E.g.

```
network "Network-Block-Example" {
      property Just for illustration;
      property author Mark Gowans;
}
```

Variable: Describes the variables or nodes within the network. Block-name must be a uniquely identifying free form text string. Valid attribute-names are *property* as for *network* and *type*. The only type we are with is discrete. The proper format for type is as follows:

```
Type discrete[<number of states>] { "state 1"  "state 2"  etc…}
```

Probability: Describes the conditional probability tables and relationships between the nodes. Block-name must be formatted as follows:

("<defined variable name>")
or
("<variable>" | "<variable1>", "<variable2>"…)

,'s are used to indicate logical ands. Attribute-names do not begin with a keyword, but instead represent a list of states and the associated probabilities in the following format:

("<variable{1..n} states>") <probability for variable in state{1..n}> ;

This is best illustrated by means of an example[10]:

```
probability("GasGauge" | "Gas", "BatteryPower") {
            ("yes", "high") 0.999 0.001;
            ("yes", "low") 0.850 0.150;
            ("yes", "medium") 0.000 1.000;
            ("no", "high") 0.000 1.000;
            ("no", "low") 0.000 1.000;
            ("no", "medium") 0.000 1.000;
}
```

Here it can be seen that *Gas* takes 2 states, *yes* or *no*. *Battery Power* takes 3, *high*, *medium* and *low*. *GasGuage* takes 2 states – the names of which could be found in the corresponding variable block.

For the complete reference guide to this format see the JavaBayes documentation. (See useful resources).

.dsc

Microsoft's format used by their MSBN[11] tool. They have not released the formal specification of the .dsc file format yet the creation tool is publicly available and via reverse engineering the files produced it should be possible the exact format of the file. Microsoft claim the format is based on their original .BNIF format, similar to the .bif (not XML) as described above. As an example, the *Dog Barking Problem* as represented in .dsc is included in *Appendix A*. Other .dsc files are included with the standard download of MSBN and can be used for reference.

[10] Taken from the JavaBayes documentation (http://www.cs.cmu.edu/~javabayes/Home/)
[11] See Section: Background – "Microsoft Belief Networks"

The Hepatitis C data source and other suggested file formats

For the purposes of this project we have been provided with a real life data source relating to the Hepatitis C disease. This data file can be downloaded from:

http://www.doc.ic.ac.uk/~dfg/ProbabilisticInference/Coursework/HepatitisC.txt .

The format for this file is as follows[12] and will herein after be referred to as the *.dfg* file format.

The .dfg File Format

```
<* Number of Variables *>
<* Number of Root nodes *>
for each variable <* The number of states of each variable *>
<* Number of Data Points *>
for each data point
      for each variable <* the state encoded as an integer *>
```

For example[16]:

4	the data file has four variable
2	there are two root nodes (the root nodes will always be placed first in the data)
5 2 4 2	variable 1 has 5 states, variable 2 has 2 states etc
7	there are seven data points
2 2 3 1	data point 1 variable 1 (a root) is in state 2, variable 2 (a root) is in state 2 etc.
4 2 3 1	data point 2…
4 1 4 1	
5 1 4 2	
3 2 2 1	
1 1 2 1	
2 2 2 1	

It would make sense to adopt the above described file format as that to be used. If a more general utility were to be authored, however, a tool would need to be produced to convert the supplied data files into the desired format.

An alternative file format that should be considered would be the use of an SQL accessible database, such as Microsoft's JET engine or similar. By storing data points in a table structure much of the file handling could be removed from the utility replaced by a pre-existing database interface. SQL Queries could then be used to access and filter the data points. A discussion of the impact that this would have in terms of the data that could be handled follows:

- **Improved Data File Format:** Data contained in a central database is a preferable method of data access over individual data files. In practice a database could contain several orders of magnitude more data than a straight text file, as concerns about its distribution become less significant.
- **Simplified Program Structure:** Using a pre-written Database Access Component removes code having to deal with non-core functionality from the utility. File handling operations can be replaced with intuitive SQL statements to make both coding and understanding the code easier.
- **Increased Speed:** Many of the popular database packages will cache frequently accessed records reducing the time taken to locate and sort the relevant data points from the source.
- **Improved Format Support:** Using a generic database access component or DAO, ODBC or similar enables a wide variety of data sources to be accessed without the need for additional coding.
- **Decreased Speed and Increased System Requirements:** Many of the available tools to connect to database sources include functionality that would not be required by the utility being authored in this context. As such – a large overhead would have to be carried in terms of the system requirements on the target machine. The overall performance of the utility would also be decreased

[12] Taken from: http://www.doc.ic.ac.uk/~dfg/ProbabilisticInference/IDAPICoursework.pdf

as SQL statements would have to be translated by the third party access tool – adding an extra delay into the process of accessing a single piece of data.

It is the conclusion of this section that it is recommended to use a generic database package for the storage of the observation data from which the Bayesian networks are to be constructed. Additionally, however, an import or similar routine should be provided to enable the existing Hepatitis C file and other similarly formatted data files to be used by the utility.

Specification

Brief descriptions of each following subsection can be found in the foreword.

The decomposition of the project into these sections is designed to be indicative of the intended design / implementation order. The specification is intended to be a minimum functionality requirements list, left deliberately open-ended to allow for further improvements / refinements at implementation-time. Where features in the project are designed to address issues raised by existing software products, these are discussed in the review of existing software, located in the *background* section.

Core Component

1. A utility must be authored that is able to *reason* about a Bayesian network described in an external .bif format file[13].

A textual or graphical interface must be provided that provides the end user with the following functionality:

- **Load a file**: To specify a file containing information relating to a Bayesian Network in a suitable file format.
- **Instantiate Nodes**: To be able to enter "observations" into the network (manually or otherwise). That is to say that for any node *n* contained within the loaded file (above), the user should be able to specify / choose the state of *n*, over-riding the prior probability matrix representing that node specified in the file (above). Conversely it should be possible for instantiated nodes to be uninstantiated – returning the prior probability of each state of that node to that specified in the original file as loaded (above).
- **Query Nodes**: To be able to view the probability of a given node being in a specified state.

Meaningful error reporting must also be implemented into the utility. That is to say, any dialogues relating to abnormal program operation, displayed at runtime, must relate meaningfully and unambiguously to the causal problem.

Online help or accompanying documentation must be provided for the utility detailing with examples (where applicable) each of the features it implements and explaining normal use of the system.

Definition of Terms (Core Component):

Reason
In order to *reason* about a Bayesian network comprising *n* nodes, the posterior probabilities for every node *1..n* must be calculated taking into account *information* (if any) contained elsewhere within the network and the prior probabilities of that node.

Information
Information contained within a Bayesian Network must take into account every node within that network, each of which can only be of 1 of 2 types:

1. A given node may be instantiated – that is to say the state of that node is explicitly stated.
2. A given node may be unobserved – that is to say the states of that node are represented by their representative prior probabilities.

[13] See Section: Background – "The .bif File Format"

Extension 1: Graphical Manipulation and Display of .bif files

The same utility as authored under the core component (described above) should have its UI modified to a graphical form whereby a 'point and click' interface is provided. A graphical representation of the network described by the loaded file should be displayed in a meaningful and *clear* fashion. That is to say:

- A circle or similar appropriate graphical symbol should represent an individual node. Instantiated nodes should be visually differentiable from non-instantiated nodes by means of colour, a point and click interface or similar.
- Relationships and dependencies between nodes should be represented by lines joining the 2 related nodes or in some other easily identifiable way. An indication of the direction of the relationship should also be provided by means of arrowheads or similar.
- The prior probabilities for each node should be constantly displayed or accessible in an intuitive way by a point and click style interface.

The graphical display of the network should be presented with minimal or no user interaction required for its construction. The network should be displayed on screen without unreasonable delay. A wait of 1 minute or greater is considered unreasonable. This delay may depend on the size of the network and for abnormally[14] large networks an initial delay of longer than 1 minute **may** be considered tolerable.

Functions provided under the core component that effects the display should do so without unreasonable delay in screen update. Node instantiation may cause a longer delay if deemed necessary but should not effect the usability of the system.

Additionally the utility should provide functionality to manipulate both existing and created networks. Namely:

- **Export to .bif:** It should be possible to save a finalised network representation in .bif format.
- **Modify Network Data:** It should be possible to edit / manipulate key items relevant to the network, for example: renaming nodes, editing probability matrices etc.

Explanation of Terms (Extension 1):

Clear
An attempt should be made to ensure the display of the network remains as uncluttered as possible. If lines are used to represent relationships between nodes:

- Overlapping and crossing should be prevented wherever possible.
- Lines should be straight without kink or bend wherever possible.
- Angles should be acute and uniform wherever possible.

If relationships are represented in some other way, uniformity and consistency should be ensured wherever possible.

Placement of nodes on the screen should be sensible. That is to say there should be uniformity and consistency in their location.

Information relating to the probabilities or instantiation of an individual node should not obscure other information on the display wherever possible.

[14] It is difficult to define abnormal so common sense should be used. Practical applications of Bayesian networks tend to be applied to networks of size <25 nodes. Networks of an order of magnitude bigger than this may be deemed abnormal.

Extension 2: Generation of Bayesian Networks from Data

A utility should be authored to automatically construct a Bayesian network from a given data set.

The utility should:

- **Interface with a data source:** The exact format of the data file is open to some flexibility. It is fair to expect data files to be complete and valid. That is to say, they must contain data relating to observations of state in a system. Each observation must contain data for every variable in the system or an assumption made and stated to handle cases of missing variable information. Data files cannot be expected to contain information relating to the structure of the final Bayesian network representation. I.e. Variable ordering and relationships must not need be present in the data files. As this extension is primarily focused on research and discovering new knowledge, the utility should interface with the provided Hepatitis C database as described in the Background Section[15].
- **User Specifiable Calculations:** The utility should enable the end user to have some degree of control over the computation of the Bayesian network. The following options should be available:
  - **Explicit relationship information:** The end user should be able to specify relationships between some or all of the variables in the data source. Explicit relationships should be able to be set and calculated relationships should be able to be broken. The inference engine should then be able to recalculate the network given the new information.
  - **View / Modify the Link Matrices:** The end user should be able to explicitly over-ride some or all of the probabilities found and calculated by the engine from the data source.
- **Display the calculated Bayesian Network:** Once a network has been computed it should be outputted graphically or textually to screen. It should then be possible to *reason* about the network. Ideally this utility would interface with the core component / other extensions to save to .bif format.

As with the core component, meaningful error reporting must also be implemented into the utility and online help.

Online help or accompanying documentation must be provided for the utility detailing with examples (where applicable) each of the features it implements and explaining normal use of the system.

In addition, a study of the Hepatitis C data set should be considered with the aid of the utility described above.

- The final 'best fit' Bayesian Network produced from the data set.
- A discussion of the accuracy of the findings should be produced taking into account inaccuracies in the original data file, inaccuracies in the algorithms used and assumptions that were made.

[15] See Section: Background – "The Hepatitis C data source and other suggested file formats"

Extension 3: Generation of Data from a Bayesian Network

The utility should be able to generate data sets related (in some way) to a provided or generated Bayesian Network.

The utility should:

- **Output to an importable file format.** That is to say, it should be possible (under extension 2) to import the dataset generated by the utility for analysis.
- **Provide End-User Customisability:** The end user should have some control over relevant options to help manipulate the generated data. For example:
    - **Alter the number of data lines generated**
    - **Customisable control over the method of data generation**

As with other areas of the software, operating speed should not hinder the functionality of the component. Data files should be generated in a timely fashion. It is considered appropriate to wait several minutes (maybe hours) to generate a dataset. A run-time in excess of 12 hours is considered intolerable even for very large network/dataset combinations. If this should not prove feasible, then some form of warning should be displayed to the end user via an *estimation of time remaining* or similar.

Notes and Limitations:

The core component need only cope with singly connected Bayesian networks.

There should be no restrictive limit placed on the size of Bayesian network the utility can deal with. A restrictive limit is deemed to be a size of less than 100 nodes, not differentiating between root and non-root nodes. A smaller size restriction may be governed by available resources on the end user's machine, although a typical target platform should be regarded as that of the lab machines provided by Imperial College's Department of Computing.

A similar restriction may be placed on the number of states any given node may take, but should not be less than 15. Any restriction must be clearly stated.

No special hardware should be required to use the utility. That is to say the utility should be accessible on a typical lab machine as provided by the Department of Computing at Imperial College.

Extension 1 should not violate any of the notes and limitations of the core component of the utility.

Implementing a Software Solution

It was clear from the outset of the project that some compromise would have to be reached, as with all software of this nature, between functionality and usability.

In choosing a development model to follow during the lifecycle of the project, two key points were noted about the specification:

- Many of the core components were envisaged as being computationally expensive.
- Some of the points to be explored were left open ended and therefore would prove difficult to assign to a fixed project time line.

Taking these into account it was decided to adopt a **rapid prototyping** style approach. In this way the direction of the project could be readdressed as each milestone in the development was reached. Additionally, the approach would enable functional, yet unstable, versions of the product to be demonstrated on a regular basis so feedback could be gained from the project supervisors and other interested parties.

When choosing a programming language to implement the project in, certain key decisions had to be made with regard to how the specification would be implemented whilst bearing in mind the suitability for a prototyping development model.

The criteria that was used for selection is detailed below:

Selecting a Language

The main choices for implementation language that were considered and their salient points are detailed below:

Java

Advantages:

An Object Oriented language such as java would lend its self excellently to the envisaged display of networks on the screen. Node symbols and relationship lines could each be treated as separate instances of node and relationship objects enabling the drawing code to be a relatively straightforward implementation. Additionally the envisaged data structures required to hold the necessary internal information such as lists, queues etc would be easier implemented using OO methodology.

Past experience in Java would reduce the time that would have to be dedicated to *learning a language*.

Using swing and similar available java concepts, the final product could be made both Unix and Windows compatible with a familiar look and feel for both.

Java is considered by many to be *the language of the future* permitting a longer life expectancy for the software as future developments in the language would enable the maintenance tasks to be simplified.

Supported by the CSG and available on lab machines.

Disadvantages:

Although not formally tested, there were concerns raised over the operating speed of the language.

Potentially a disproportionate amount of time would have to be spent developing a UI in Java than would be otherwise ideal under a prototyping development model.

If the utility would be tied into a database product for the data learning component, Java would lend its self less well than the other alternatives with only emerging support for SQL databases etc (via JDBC).

(Visual) C++

Advantages:

As with Java, the advantages of using an Object Oriented language apply.

Microsoft's implementation of VC++ includes a fully integrated development suite supporting rapid development of a UI, conducive to the prototyping methodology.

The compiler included in the development suite would enable a potentially faster program execution time since code would not have to be *run* as interpreted code. Indeed the general speed concerns raised if using Java would not be applicable for software developed in VC++.

Support is available for database connectivity via tried and tested Microsoft COM objects.

Available on home machine.

Disadvantages:

No previous experience in the language would lead to a large *language learning* time overhead.

The final software would be confined to Microsoft Windows machines.

Not available in DoC.

VC++ has a relatively high overhead in ensuring that the code conforms properly to the language specification, for example – preventing memory leaks etc.

Visual BASIC

Advantages:

Authored as a *user-friendly* language abstracting the development process up one level from VC++ - since the coder need be less concerned with the *nitty-gritty* of the language.

Previous experience in the language ensures that no time would have to be spent *learning the syntax*.

Even more so that with VC++ , Microsoft's development suite allows extremely rapid prototyping of the UI.

Supports Microsoft COM objects for database connectivity etc.

Available on home machine.

Disadvantages:

Non OO language – raising potential difficulties with display code and internal data structures.

There are slight concerns that could be raised as to the running speed of VB code.

The final software would be confined to Microsoft Windows machines.

Not available in DoC.

Having evaluated each of the possibilities the further design decision was reached that it would not be a limitation if the final software were confined to a Microsoft Windows machine.

As a result of careful consideration it was decided that the primary implementation language would be Visual BASIC because it permitted the emphasis of the project to be shifted from that of implementation to that of exploration.

The graphics / display code would be unnecessarily complicated to implement in VB, so it was decided that a Microsoft ActiveX control would be authored in Visual C++ and then *plugged-in* to the main application as an OCX.

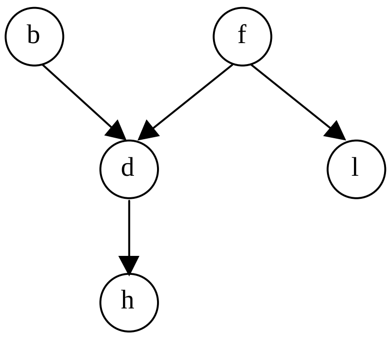
At the initial stages it was decided that the code **should** be tied into a database of sorts since:

- It would permit greater flexibility for the learning functionality etc during the middle stages of the development
- Internal data structures could simply be stored as tables at the beginning of the development process enabling the focus to shift toward algorithm design.

As a result of this it was decided to adopt a potentially *throwaway prototyping* model for development. For ease of design, initial code was produced using VBA for Microsoft Access, enabling the application to be tied very closely to an SQL implementation.

The first task was to design the OCX control for the display of network.

It was hoped that the display would resemble the diagrams used in this document to illustrate the dog-problem network.



The following functionality would therefore be required to **display** the graph:

- `Drawnode(label,position,colour)`
- `DrawLine(Parent Node, Child Node)`

Additionally functionality would also be required to:
- Change the properties of a node; label, position, colour etc,
- Remove Nodes
- Remove Lines

Finally since the display would permit the user to have a certain amount of interaction, a hit test would have to be implemented to return where the user had clicked on the screen. Through experimentation it was also deemed to be practical to let the OCX control deal with users rearranging the position of objects on the screen, so the main VB application would be totally unconcerned with all aspects of the display apart from when a user clicked on the screen.

Design of the OCX control was straightforward, since most of the required drawing primitives: Circles and Lines, were implemented by Microsoft. Arrow heads proved to be a slightly more challenging task since no implementation code could be found anywhere available for download. It was decided in the end to opt for arrow direction indicators to be placed at the midpoint of the line. The direction and gradient of the line was calculated from the `DrawLine` method (based on the position of the start and end nodes). A standard graphic (▼) was then rotated using standard MS methods and placed at the midpoint of the line. The result, although not perfect, suffices for the intended application. It may be noted that sometimes on perfectly vertical / horizontal lines the direction indicator appears at a slight angle as a result of rounding errors.

Once the display code was authored, the next obvious step would be to write the file loading code to enable the networks to be displayed on screen.

Since Version 3 of the .bif format is XML compliant, the possibility of using a standard XML parser to load the file was considered in some depth. However, Version 1 of the format was not in XML and since the utility need only deal with this tiny formatting subset of all possible XML files, it was decided that an in-house parser would be written from scratch.

Visual BASIC contains a `LineInput` method, returning the entire line of a text file from current position to Microsoft newline. Each line could then be *tokenised* using the XML begin and end tags as delimiters. Once tokenised, separate functions to handle *Definitions*, *Variables* and *Networks* were able to load the file into the internal data structures.

The format of the internal data structures was initially as follows:

Node States	
`NodeID`	`(Int)`
`StateName`	`(String)`
`Probability`	`(float)`

Nodes	
`NodeID`	`(Int)`
`Description`	`(String)`
`NoOfStates`	`(Int)`

Node Types	
`NodeID`	`(Int)`
`RootNode`	`(Bool)`
`LeadNode`	`(Bool)`

Relationships	
`NodeID`	`(Int)`
`ParentIDs`	`(String)`
`PMatrix`	`(String)`

Since the data was being held in a database, a unique **NodeID** was generated for each variable in the network. This ID was then used as a key field on each of the other tables to enable faster access to the required section of the data. The tables described above roughly reflect the data that is contained in a .bif file stored in a way to attempt to improve access speed to frequently accessed components:

- **Nodes** is designed to act as a mapping between the meaningful variable names present in the .bif file and the internally reference. SQL searches on integer key values is considerably faster than using string keys.
- **Node States** holds as a list each state against each node with its calculated probability. Although slightly wasteful on space (repeating the NodeID) – this structure would ensure fast access the required probability values – a figure that was envisaged as needing frequent reference.
- **Relationships**. This table reflects almost identically the structure of the *<definition>* data in the .bif file. The probability matrix is held as a string as in the file.
- **Node Types** is / was a slightly redundant table, enabling fast access to determining whether a node is of a special type. The root node field equates to `isnull(!ParentIDs)` from the correct subset of the relationships table.

As a result of the drawing OCX, file parser and data structures – a demonstrable prototype (pictured right) was up and running by the end of week 2.

Notice how the development in VBA/Access necessitates the main Access application to be running and visible in the background. At this stage, the display was divided into 2 primary parts, the network view canvas and a node list to the left.

The next major feature to implement would be the probability propagation's.

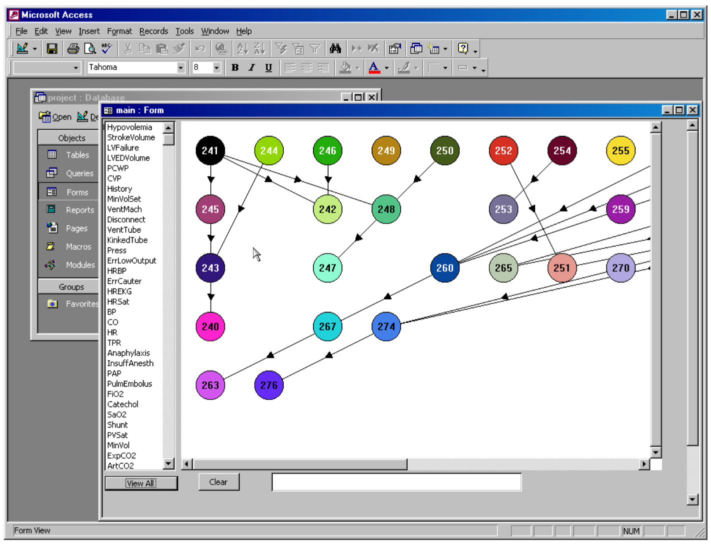
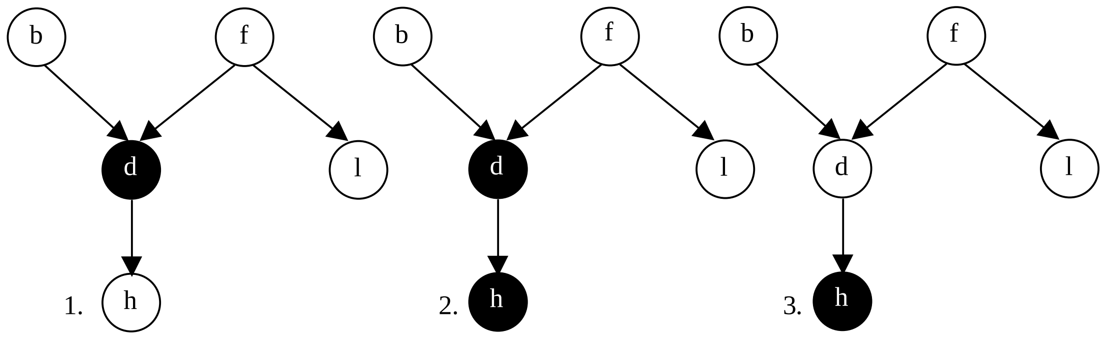


Fig I1. BNT as of Feb. 2001

The pseudocode algorithm to implement the propagation functionality appears relatively straightforward, requiring only 3 functions to be authored:

- `CalculateProbability`
- `PropagateLambdaTo`
- `PropagatePiTo`

Initially the individual Lambda and Pi messages and evidence was lost, with messages being directly normalised into the posterior probability at each node. Although initially all seemed well, testing revealed that the theory just didn't work. The dog problem network was loaded into the BNT as well as Microsoft's MSBN. Probabilities were compared at each stage and found not to correspond. The results of this initial testing have not been included in this report since they only highlight an error in the intended algorithm. A refinement of the original data structure lead to *Pi_Evidence* and *Lambda_Evidence* fields being added to the *Node States* table. This is again a slightly unacceptable solution since it necessitates a hack in uninstantiating nodes. During testing of the revised situation – the following case was found to cause error:



If some node were instantiated all probabilities would be calculated correctly. Similarly, if that one node were then uninstantiated, all would be fine. Instantiation of some middle node (step 1) followed by a child node of that node being instantiated (step 2) would sneakily let all *seem* fine. However, the problem with the data structure comes secretly into play. The problem is only noticeable if the first node is then uninstantiated (step 3) – when the probabilities are incorrectly calculated. The problem is caused by instantiated nodes having their lambda evidence set to 1. Lambda message propagation is blocked through that node and so the messages are in effect lost since the Lambda evidence at the instantiated node is not updated. To overcome this, the BNT uses a re-instantiation hack. Upon uninstantiation of a node with a parent instantiated; **all** nodes are uninstantiated, the network is reset and each previously instantiated node is reinstantiated, omitting the node to be uninstantiated. Although in practice this works accurately, it does have a significant execution time overhead. However, since testing has shown that probability propagation times in the final version are ~ 0.1second, this overhead is not deemed to have a noticeable detrimental effect on the overall performance of the system.

With propagation code in place, the main screen required updating to present the probabilities to the user.

Once stabilised the next feature for implementation was the learning algorithm.

The exact file format of the data source was debated, but it was decided to adopt, initially at least, the dfg file format as described in the background. It was decided to hold the entire data file in an Access table, which was dynamically generated but of the form:

Data	
`Observation1`	`(int)`
`Observation2`	`(int)`
…	
`ObservationN`	`(int)`

In using such a structure, calculating probabilities could be simplified no matter how complex a conditional probability was required. Dynamically generated SQL queries could be produced in code to produce recordsets directly containing the relevant data to calculate probabilities. For example, to calculate the probability of *Variable1* in *state 1*, *2* in *2*, *3* in *3* up to *n-1* in *n-1* given *n* to be in state *n* we trivially generate the SQL query:

```
(Select * from (select * from Data where VariableN = n) where
Variable1 = 1 AND Variable2 = 2… AND VariableN-1 = n-1).recordcount
```

By Bayes:

$$P(A \cap B) = p(A \mid B)\, p(B)$$

We can divide our answer to the SQL query by p(n) to give us our conditional probability. In this way – no matter how many variables we need on the conditional side, a single line function is able to return the correct probability we need.

p(n) trivially reduces to:

```
(Select * from Data where VariableN = N).recordcount/data.recordcount
```

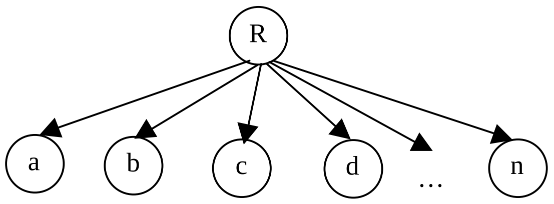
With the required data structures in place and a parser for the dfg files written the choice of which Learning algorithm to use had to be made.

Learning Algorithms

A great deal of study into the effectiveness of various learning algorithms has been conducted and it is beyond the scope of this project to continue such work. However, a choice must be made as to which algorithm should be implemented in the BNT.

Out of the multitude of possibilities available, three distinct algorithms are worthy of closer inspection for the purposes of this project.

Naïve Bayes



The naïve Bayesian classification algorithm simply expresses each non-root node as a direct child of each of the root nodes, as shown above. No other connections are allowed. In this way, the network will never have a depth of greater than 2.

This approach would tie in well with the utility being authored, since by definition, the produced network will always be singly connected.

The Naïve-Bayes approach may be initially dismissed as being too trivial, although tests have shown[16] that it has often proved more accurate than many more sophisticated classifiers. Indeed in the case where the variables in a dataset are not strongly correlated, the Naïve-Bayes produces the most accurate representation of that scenario.

Two further advantages of the Naïve-Bayes approach are:

- Fast construction time, due to a minimisation in the number of required complex calculations
- Fast classification time, since probability propagation's are also minimised.

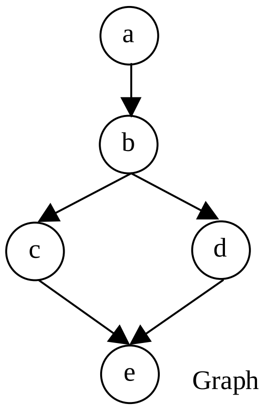
Various modifications / enhancements to the Naïve-Bayes classifier are discussed in *Learning Bayesian Belief Network Classifiers: Algorithms and Systems – Cheng and Greiner* but are not relevant to this application.

[16] Langley et al. 1992

Cheng, Bell, Liu 3 Parse Algorithm

Taken from the paper: *An Algorithm for Bayesian Belief Network Construction From Data – Cheng, Bell, Liu.*

The algorithm consists of 3 phases, *Drafting, Thickening* and *Thinning.* For our explanation, let us consider the example network shown below.



Graph l1

In order to compute the network from a compliant dataset, the node ordering must be given (i.e. ABCDE indicating that nodes must be at the same or higher level than variables appearing after them in the ordering).

Phase 1 (Drafting):

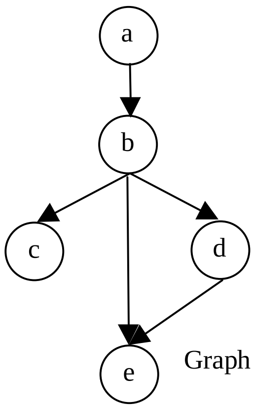
- For each pair of variables in the dataset, compute the mutual information I(a,b) using[17]:

$$\sum_{ai,\,bj} P(ai,bj) \log \frac{P(ai,bj)}{P(ai)\,P(bj)}$$

- If the mutual information I is greater than some threshold value ε, then add them to an ordered set S.
- Take the first 2 pairs of nodes from S and add them (with arcs between) to the graph so far. The direction of the graph is given from the node ordering.
- Remove the next pair of nodes from S. If there is no open[18] path between the nodes, add the corresponding arc to the graph. Otherwise add the pair of nodes to another ordered set R.
- Repeat the last step until S is empty.

Let us assume with our example, that after the first phase, our graph looks like that below.

From this we can see that an arc has been incorrectly added between *b* and *e*. Additionally the arc *c* to *e* is missing. The disparity from the original is caused by the open path condition of the first phase and forms the basis of phase 2.



Graph l2

Phase II (Thickening):

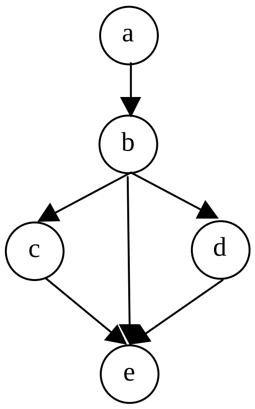
- Remove the first pair of nodes from R
- Find a block set that blocks each open path between this pair of nodes by the minimum number of nodes. Conduct a CI test using the formula:

$$\sum_{ai,\,bj,\,c} P(ai,bj,c) \log \frac{P(ai,bj \mid c)}{P(ai \mid c)\,P(bj \mid c)}$$

- If the nodes are still dependent on each other, add them to the graph.
- Repeat until R is empty.

[17] From Information Theory.
[18] D-B-E and D-B-C-E are examples of an *open* path in graph l2.

The third step in the thickening stage will add the required arc *c* to *e* in our graph, so we have:



The final stage, in our example, will attempt to remove the incorrectly added relation *b* to *e*.

Phase III (Thinning)

- For each arc in the graph, remove it iff there are open paths between the nodes.

- Conduct a CI test on the removed nodes **relative** to the graph so far.

- If the two nodes are still dependent, add them back to the graph. Discard them otherwise.

The final stage should return our graph back to its original form, since the conditional independence test for *b* to *e* given *b* to *c* to *e* and *b* to *d* to *e* should show the nodes to be independent.

The main advantage of this algorithm is that it is believed to construct a graph as close to the original as is conceivably possible. Computationally however, it is very expensive, requiring a large number of I and CI tests.

The third choice of algorithm to consider is that covered in the DoC Intelligent Data course:

The Spanning Tree Algorithm

The idea behind the spanning tree algorithm is to start with a set of nodes to which we add arcs until a complete network is created[19].

- For every pair of variables in the dataset:
    - Calculate the dependency between them.
    - Add the pair and their dependency score to a set R

- Order the set R

- Add the first pair in set R to our graph connected by an arc, assuming it does not create a loop.

- Repeat the last step until each node is connected or R is empty.

Although appearing relatively straightforward, like the *3 Parse Algorithm*, the *Spanning Tree Algorithm* is computationally expensive, requiring O(n^2) calculations.

It is clear that all except the naïve Bayes approach are computationally difficult. As it is intended that the users should have some degree of control over the learning algorithm by being able, for example, to explicitly set relationships between nodes, it was decided that the naïve Bayes approach would not be an acceptable implementation of a learning algorithm. Additionally, with this functionality – it would be possible to force a naïve classification manually, over-riding the implemented algorithm. Therefore, the choice was reduced to the *Spanning Tree* or *3 Parse* algorithms.

Although the *3 Parse* approach may theoretically produce more *accurate* results, it was decided to implement the *spanning tree algorithm* for the following primary reasons:

- The algorithm inherently produces a singly connected network – an agreed limitation of the BNT.
- The algorithm is tied closer in to the 4th year Intelligent Data Course at Imperial, promoting the *Learning Aid* aspect of the project.

[19] Duncan Gillies - IDaPI

In an attempt to reduce the computational expense of the *Spanning Tree* algorithm, the effects of different dependency measures were examined.

The *Mutual Entropy* measure of dependence as used in the *3 parse algorithm* (defined above) uses a log calculation. Other measures of dependency using simpler computational operators were examined. Specifically:

Weighted L1 Metric:

$$\sum_{ai,\, bj} P(ai,bj) \times | P(ai,bj)-P(ai)P(bj) |$$

Weighted L2 Metric:

$$\sum_{ai,\, bj} (P(ai,bj)-P(ai)P(bj))^2$$

Since the difference in expense between the *L1* and *L2* metric is minimal, tests were conducted to compare the effectiveness of the *L1 metric* vs. the *Mutual Entropy* measure. The findings of testing are not included in this document, since there was no discernible difference in the results of the 2. In all test cases, the graphs produced using both metrics were the same. The performance of the *L1* was marginally superior to that of the *ME* and so has been used in the final implementation. Code remains implemented, although not used, for the *ME* metric should it wish to be reverted to.

With a choice of learning algorithm made an implementation was produced. The pseudocode implementation follows the step by step outline of the *Spanning Tree algorithm* almost exactly and so has not been included in this document.

Once implemented, the utility was tested with the Hepatitis C data set.

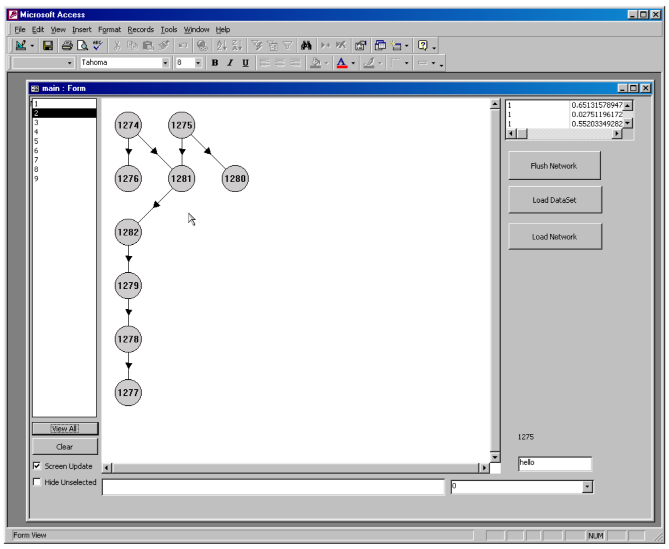


Fig I2. The 2nd Major Release of BNT

Figure I2 shows the toolkit at this next milestone during its development. Displayed in the *Network View Canvas* is the graph generated from the Hepatitis C data set, using the implemented *spanning tree* algorithm. More detailed results can be found in Appendix D.

Note the subtle changes required since Fig I1, namely:

- The addition of a combo box toward the bottom of the screen and list box top right, to display probability information for debugging and testing purposes.
- Load Dataset, Load Network and debugging Flush Network buttons.
- The use of colour-coded nodes has been abolished and replaced with a less confusing, uniform grey.

Having compared results of the algorithm to other independently authored implementations of the algorithm, results appear correct. Slightly disappointing was a learn time of 8 minutes for the Hepatitis C dataset. This was deemed acceptable however.

With a learn time for the Hepatitis data set well into the *boring* category, the save (export to bif) functionality was next to be implemented.

Since the implementation of save was to overcome the time overheads of re-learning data, it was decided to implement only 1 file-save format at this stage. The most sensible format was deemed to be Version 3 .bif (XML) since this would be the most useful for future development of the project.

As the internal data structures closely represented the desired output file format, writing the necessary save routines was not problematic.

It was decided that the output XML file should be XML version 1.0 compliant and include a standard header and BIF DTD.

Saved files were fed back into the system to check for validity. The Hepatitis C network represented in XML is included in Appendix D.

In order to complete minimal functionality under each section of the specification, the Data Generator still needed to be implemented.

Generating Data

Planning the development of the data generator was complicated by the total lack of any existing software products offering similar functionality to use as a benchmark.

The theory behind the data generator appears straightforward:

- Instantiate 1 or more nodes in a network.
- Propagate the probabilities.
- Output the most likely states of each node.

The difficulty comes in deciding how many nodes to initially instantiate. In order to maintain flexibility the generator was initially authored as a function taking as one of its parameters a list of nodes to instantiate.

Calculating the initial states of each node was done dynamically using VB's internal random number generator. Tests showed that the number generator behaved as desired, producing an approximately linear distribution of numbers between 0 and 1. Comparing the randomly generated number to the probabilities at each node, a corresponding state could be chosen.

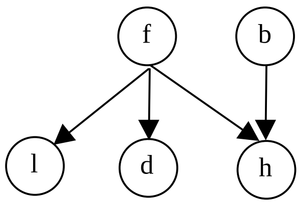
Trials were initially conducted by instantiating just the root nodes. Unfortunately the data generator proved too slow to be practical.

Probability propagation's took around 1 second to complete during testing with the *dog problem* network. Generating data for this network takes 2 instantiations (1 per root node) and a network reset (equivalent to 2 uninstantiations). Total time taken therefore to generate 1 line of data (including general overheads) was approx. 5 seconds. This was deemed to be unacceptable and prompted a migration of the application out of Microsoft Access (see *Moving out of Access – Prototyping in Action).*

With the speed issue addressed investigations into the effects of instantiating different nodes was conducted. For each of the tests, data was generated intending to conform to the *dog problem* network. For each of the tests, probability information can be found in the output files for each case, located in Appendix C.

Test 1: Instantiating Root Nodes

Instantiating just the root nodes of the network and generating 500 lines of data, the network (once fed back into the utility) produced was as below:



Prior probabilities for the root nodes *family_out* and *bowel_problem* appear correct (to 1SF) to the original *dog problem* network.

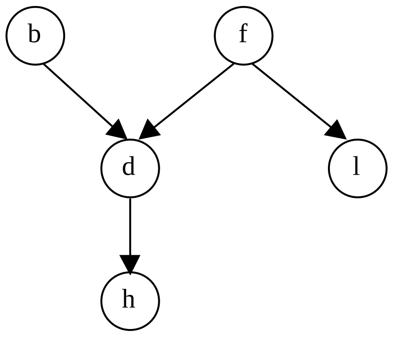
Interestingly *hear_bark* has remained a child of both root nodes, although has become a direct child – no longer a child of *dog_out*.

Light on remains a child of *family out*, but *dog_out* now appears to be no longer related to *bowel_problem*.

Test 2: Instantiating All Non-Root Nodes

Again generating 500 lines of data, but this time instantiating everything **but** the root nodes.

The generated network looks a lot more like the original *dog problem* network.



The prior probabilities of the root nodes are less accurate than under *Test 1* coming in at:

	P(f)	P(b)
Normal	*0.28*	*0.32*
Abnormal	*0.72*	*0.68*

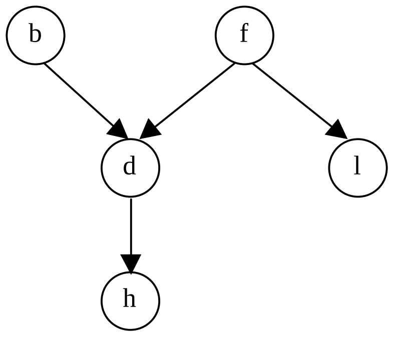
Compared to the original at:

	P(f)	P(b)
Normal	*0.15*	*0.01*
Abnormal	*0.85*	*0.99*

The link matrices are also noticeably different to the original, yet remain with the bias on the correct state.

Test 3: Instantiating All Nodes

Instantiating all nodes and generating 500 lines of data again, like the Non-Root Node instantiation produces a familiar looking graph.



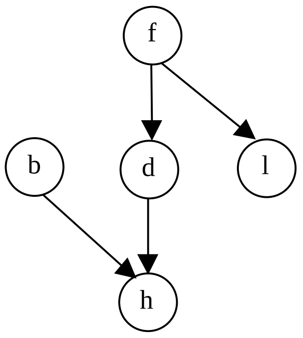
The prior probabilities of the root nodes are not much closer to the original when compared to *Test 2* and appear to have *bowel_problem* and *family_out* reversed. Results are:

	P(f)	P(b)
Normal	*0.02*	*0.17*
Abnormal	*0.98*	*0.83*

Test 4: Instantiating 3 Nodes

To see whether the major factor in produced model accuracy was the number of instantiated nodes or if root node instantiation played a more influencing role, test 4 was produced. Again, 500 lines of data were generated, but this time *family out*, *lights_on* and *hear_bark* were instantiated.

The results are interesting. The produced network bears more than a passing resemblance to the original. The only change is that *bowel_problem* has become a direct parent of *hear_bark* and not of *dog_out*. The same effect can be observed in *Test 1*.



The prior probabilities for the root nodes (displayed below) still appear to differ greatly from the original.

	P(f)	P(b)
Normal	*0.15*	*0.4*
Abnormal	*0.85*	*0.6*

Test 5: Instantiating All Non-Root Nodes with 1000 lines of data

As a final test, *Test 1* (the least accurate results) was rerun with double the amount of data generated to investigate the effects of the quantity of data.

$$\begin{tikzcd}
 & f \arrow[dl] \arrow[d] \arrow[dr] & b \arrow[d] \\
l & d & h
\end{tikzcd}$$

The results were very similar to that of *Test 1* with an identical network resulting. The prior probabilities had not changed significantly, still correct to 1SF.

As a conclusion to the tests – it would appear as though the accuracy of the generated model is more dependent on the choice of instantiated nodes than the amount of data generated. Instantiating non-root nodes appears to produce the most accurate data although further testing would show whether instantiating all nodes has an overall improved result.

Moving out of Access – Prototyping in Action

As discussed briefly above, it became necessary to throw away the Access implementation. Execution time was too slow and so it was decided to port the code out of VBA into VB with the consequence that the support for SQL databases would be lost. In light of the performance increase resulting from this action, this was deemed an acceptable trade off.

Since the code was previously in VBA, the task of porting to full-blown VB was simplified. All the data-structures and associated routines however, had to be completely rewritten. In order to minimise the work required, new data structures were devised to represent the previous table structure. They key structures are:

```
Type NS_State
    probability As Double
    lambda_evidence As Double
    pi_evidence As Double
    state_name As String
End Type

Type NodeStates
    States(30) As NS_State
End Type

Dim NodeStatesTable(100) As NodeStates
Dim NodeTypesTable(100) As NodeTypes
Dim RelationshipsTable(100) As
Relationships
Dim datatable() As Double
```

```
Type NodeTypes
    node As Integer
    rootnode As Boolean
    leafnode As Boolean
    description As String
    NoOfStates As Integer
    Index As Integer
End Type

 Type Relationships
    node As Integer
    Parents As String
    ProbabilityMatrix     As
String
    Instantiated As Boolean
    State As String
End Type
```

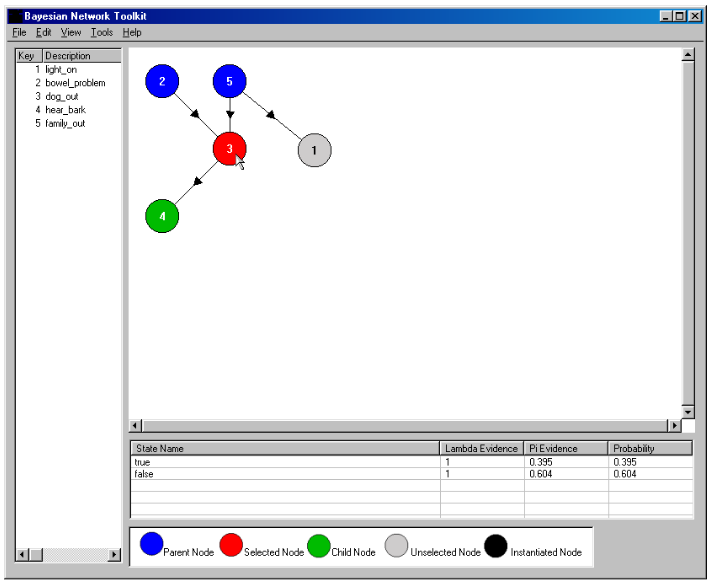
The three primary arrays dimensioned to 100 correspond to the 3 primary tables as designed in the VBA/Access implementation. Note that `datatable` remains undimensioned since VB permits the dynamic redimensioning of arrays, enabling the structure to work in much the same way as the dynamically created data table in the original version. Also, note the restriction of 30 states per node placed in the `NodeStates` definition.

Much of the work required to permit the change over was involved with removing the inline and dynamically generated SQL queries. The functions associated with conditional probabilities were worst effected and prompted the creation of a pseudo-SQL generator and parser.

As a result of the changeover, propagation speeds for the *dog problem* network increased from 1 propagation / second to around 20/second.

Access does not permit the export of *Forms* to VB, so this provided a good opportunity to address the front end of the toolkit.

Design of the UI



Key	Description
1	light_on
2	bowel_problem
3	dog_out
4	hear_bark
5	family_out

State Name	Lambda Evidence	Pi Evidence	Probability
true	1	0.395	0.395
false	1	0.604	0.604

Fig. I3 – The final front end

Fig I3 shows the newly designed front end. Note the many similarities between this and the first version.

In building a front end – it was important to bear in mind the following user needs:

- Some networks will be very large and as such will not be entirely viewable on one screen. The display area should therefore be large enough to display a significant proportion of most loaded networks and at least big enough to display the parents and children of the currently selected node in the standard case.
- Since a key is generated to use as a label on the nodes, a lookup table should always be clearly visible to enable the user to translate the node label into the more meaningful node name.
- State information should be displayed at all times for the currently active node in a way that does not obscure the view of the network.
- In line with MSBN, it should be possible to view the node/state probabilities for all nodes in a network, although this need not always be visible. It would be useful for such a feature to allow the user to be able to display both the probabilities and the network at the same time to permit cross-referencing.

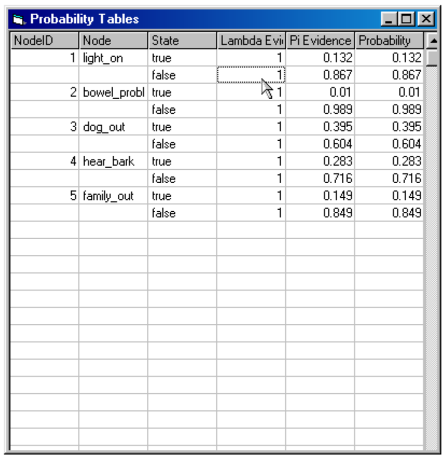
As a result, the screen has been divided into 3 primary areas to reflect the most commonly viewed components of the application.

The largest portion of the display is dedicated to the *Network View Canvas* – which displays the currently loaded network. Resizing the overall program window increases the size of the *NVC* proportionally, increasing the visible portion of the currently displayed network.

Since the circles representing the nodes use a key to indicate which variable they represent, the *Node List* fills the next largest portion of the screen. It displays the original node names next to the automatically generated key, sorted by the key to enable the desired lookup to be easily performed.

Probability information for each state of the currently selected node is displayed in the *State Information Table*, positioned directly beneath the *NVC*.

A system of colour coding of nodes has been implemented so the currently selected node can easily be identified. Instantiated nodes appear in black to enable them to be differentiated. The immediate family of the currently selected node is also colour coded to help identify the directly influenced nodes in the event of a complex network being displayed.



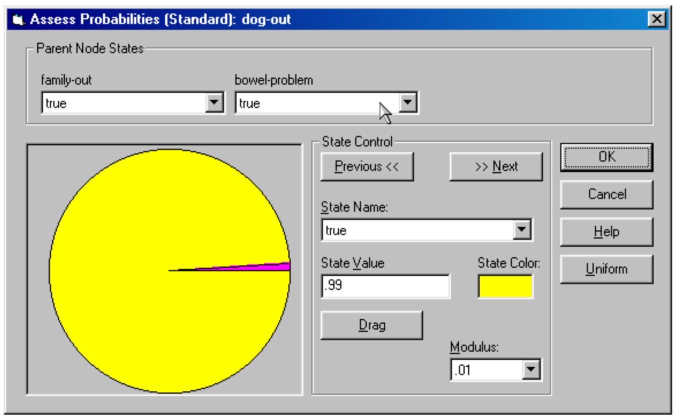
NodeID	Node	State	Lambda Evi	Pi Evidence	Probability
1	light_on	true	1	0.132	0.132
		false	1	0.867	0.867
2	bowel_probl	true	1	0.01	0.01
		false	1	0.989	0.989
3	dog_out	true	1	0.395	0.395
		false	1	0.604	0.604
4	hear_bark	true	1	0.283	0.283
		false	1	0.716	0.716
5	family_out	true	1	0.149	0.149
		false	1	0.849	0.849

A separate window (see above) is provided, that can be resized as necessary, to display the current node/state probabilities for the entire network. The tables are updated automatically every time a modification is made to the network.

With the software reaching maturity, it was necessary to implement the end-user functionality for network modification.

Modifying the Network

One of the first issues to address was how to display the probability matrices for a given node in a way that would permit the user to modify them.



family-out: true
bowel-problem: true
State Name: true
State Value: .99
Modulus: .01

Fig I4 – MSBN's View Probabilities Window

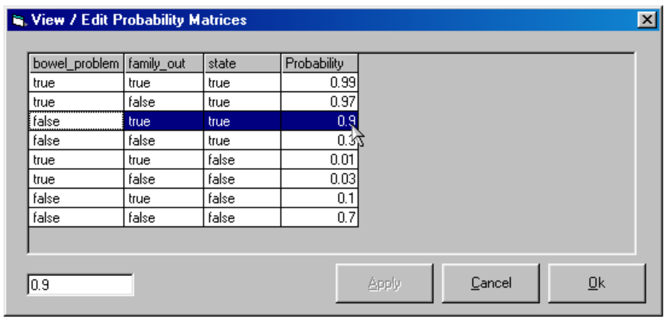
Microsoft's approach (pictured left) was used as a guide to give pointers into the design of the probability matrix display.

Fig I4 displays the equivalent of the probability matrix for the node *dog_out* in the classic *dog problem* network. Note how combo-boxes are used for each parents so the user can enumerate through each state of the parents. For nodes such as this where there are only two states, the software automatically sets the unedited matrix value to 1-the edited one. A graph displays the proportion of the state allocations per parent combo.

While the Microsoft approach appears very pretty, a few lessons can be learned to aid in the design of the equivalent for the BNT. The automatic allocation of probabilities to unedited states (1-edited) goes very wrong on nodes with more than two states. As a result, a seemingly useful feature is in fact, upon closer inspection, a hindrance. More seriously, however, the rationale behind using separate combo boxes for each parent has to be questioned, since concerns must be raised for nodes with large numbers of parents. The BNT in theory would support up to 99 parents of any 1 single node.

Taking these points into consideration, it was decided that the best way to display / modify the probability matrices would be via some form of enumerated table.

Fig I5 (right) shows the BNT equivalent of the MSBN window in Fig I4. A table is generated by the software has columns for each parent node – ensuring that no matter how many parents a node has, the information will still be comprehensible. All the state combinations have been permutated for the user and are displayed in a uniform way to aid in locating the correct value to modify. The toolkit performs a check when the user closes the window to ensure that the probabilities of the node, for each parent combination, sum to either 1 or 0. Should a discrepancy be found, the user is asked whether he/she would like to return and modify the probabilities manually again. Alternatively, the software is able to *auto normalise* the probabilities, automatically ensuring they add to 1, based on the current probability ratios for that combination.



bowel_problem	family_out	state	Probability
true	true	true	0.99
true	false	true	0.97
false	true	true	0.9
false	false	true	0.3
true	true	false	0.01
true	false	false	0.03
false	true	false	0.1
false	false	false	0.7

0.9

Fig I5 – BNT's View Probabilities Window

Lastly, the finishing touches were added to the software, namely:

- A utility to enable users to rename nodes. This feature was really necessitated by the learning component of the toolkit, which merely assigns numerical state names.
- During testing, it was found that the probabilities were often displayed to too great a number of significant figures. A user option was added to allow the user to modify the number of displayed decimal places, without altering the accuracy of the internal calculations. A bug has been noted, however, that Visual Basic's divide function returns (0.5 *100)/100 as 0.49. An issue that is still being addressed.
- The BNT logo was designed to add an icon for the program – giving the software a more professional feel.

Testing was a continual process throughout the development of the product. The more significant tests have been detailed in their relevant place in this chapter. Additional noteworthy tests were carried out that have been detailed below, since they would have distracted from the development process as detailed above.

Additional Testing

File Handling

An interesting problem was noted during development caused by Unix saved text files, whereby the entire file would be read in by a single `LineInput` call. Although seemingly not a problem, the in-built VB function `Trim` was used to remove leading whitespace from a line or token so it can correctly be identified. When read in, Unix */n* new line characters appear as a non whitespace character and so initially lead to problems reading in the file. A replacement function for `Trim` was authored to overcome this problem.

Further testing of the Learning Algorithm

To investigate the effectiveness of the learning algorithm, a completely random dataset was produced with:

- 5000 Observations
- 4 Root Nodes
- 18 Variables
- Each of which has 3 states

The code to produce this dataset has been included in Appendix C.

The dataset should theoretically have conformed to:

- Nodes 6 to 18 should be children of node 1.
- Nodes 2 to 4 should be islands.
- All nodes should have a probability distribution of approx. 0.3, 0.3, 0.3 across their states.

When fed into the learning component, the results were pleasing.

- Node 9 was a child of node 5, in turn a child of node 2.
- Node 6 was a child of nodes 2, 3 and 4.
- All other nodes were children of node 1.
- The probability distributions were approximately as desired, with the highest noted probability at 0.41 and the lowest at 0.26

In essence, only 3 nodes were theoretically wrongly classified (5,2 and 6). Considering an imperfect data generator, the results were very exciting.

Testing Instantiations

The dog problem network was used to test the instantiation / probability propagation accuracy of the toolkit. Since there are only 5 nodes, each of which can be instantiated or not, only 32 test cases were required to test every possibly combination of instantiation.

The results of the tests were compared to the results of the same test run using the MSBN application. Results were found to be correct in all cases – with the BNT often exceeding the MSBN in accuracy.[20]

[20] Where a discrepancy between the MSBN and BNT results was found, the calculations were performed by hand to verify which was correct.

Conclusions and Evaluation

It is often difficult to appraise a project of this nature in terms of a 'degree of success'. Similarly with such a healthy '*future extensions*' section, care must be taken not to criticise the software for work not done that appears in the aforementioned chapter. However, every successful and worthwhile project shares a common feature, in that it should not be complete, instead providing a solid and interesting basis for future development. The Bayesian Network Toolkit, I believe, certainly comes into this category.

In evaluating such a project one must first begin by discussing how the final product compares to the initial intended specification. As with the introduction, we shall break the software down into its 4 constituent parts and examine each independently.

Evaluation with regard to the specification

Core Component

The specified functionality of the core component in terms of *features* has been completely implemented in the BNT.

Load a File:

The utility is able to parse correctly formatted Version 1 and 3 BIF compliant files and in doing so, hopefully will go some way to establishing the .bif standard. Indeed the BNT is believed to be the only software product currently able to meaningfully utilise the file format. Conversely, however, this adoption of the new standard does pose a slight limit on the utility, since users of competing systems are likely to have their networks saved in other proprietary file formats, such as Microsoft's DSC format, not compatible with the BNT. The utility is also bad at handling corrupt or ill-formatted files, erring on the side of caution and failing to load files that don't meet the standard in the minutest detail. Additionally, since a proprietary XML parser (suitable only for use with the .bif formats) has been authored for the toolkit, future modifications to the file format will necessitate changes to the program code.

Instantiate Nodes:

A particularly user-friendly feature of the utility is the node instantiation functionality. Testing has shown the process to work accurately and in a timely fashion. Although not having a noticeable effect on functionality, the uninstantiate node option is slightly less than satisfactory. As discussed earlier in the report, the adopted solution to uninstantiating nodes could be considered a *hack* and is cumbersome at best. Tests into the timeliness of this feature for very large networks with a high number of instantiated nodes have not been conducted and it is feasible that that system may become *visibly slow* in this case. However, since the feature is implemented, and works acceptably in the typical scenario, this poor implementation should not reflect detrimentally to the overall usability of the system.

Query Nodes:

The *State Information Table* as displayed on the main screen of the utility combined with the *View Probability Table* options means that the node-state probabilities are / can be constantly displayed and updated as necessary without the need to *query* a specific node. The information is presented in tabular format, which was intended to give at-a-glance usability to the toolkit and indeed the implementation does go most of the way to providing this. There are, however, a few drawbacks to this approach which users of other products, particularly Microsoft's MSBN, might pick up on. Namely, with large numbers of nodes it can become difficult to keep track of the required state information in the *View Probability Table* view. The potential problem is that of information overload – there are just too many figures to keep an eye on, and it becomes easily to loose track of which numbers are important. For a discussion of how this situation could be improved upon, please refer to the '*The Future*' chapter. Despite its drawbacks, the *query nodes* functionality achieves everything and more than it was intended to do.

Extension 1

As with the core component, each separate section discussed in the specification of extension 1 has been addressed and implemented. The extension can be broken down into 2 distinct areas for evaluation:

The utility was intended to have a GUI attached to the core component providing a *point and click* style interface. This has been achieved. However, criticism of a GUI is always subjective and end-user testing would have gone some way to assessing the effectiveness of the interface. However, it was intended that the display should be meaningful and clear, so let us attempt to qualitatively rate the success of the BNT in providing this.

Clarity of the UI:

Nodes:
As originally suggested, a circle is used in the toolkit to represent a node. Colour coding combined with a uniform size for each node enables the core network features to be displayed, to be clearly and unambiguously differentiated between.

Using a key system for labelling the nodes enables the network as displayed, to remain clutter free. It may be argued that with a key, it is not always obvious which node any given circle (as displayed) represents. Indeed since the key is generated by the utility, some users may be initially confused by the concept. However, when we compare the BNT display to that of other products, the advantages of the adopted approach become apparent. Microsoft's convention for placing the name of each node actually **in** the displayed symbol removes any ambiguity as to which node each symbol actually represents. However, it does lead to each node having a different sized symbol representing it (the larger the name, the larger the symbol), which leads to some nodes appearing visibly more important / significant than – others – which is simply not the case. The JavaBayes approach places the name for each node underneath the displayed symbol, thereby (as with the BNT) guaranteeing each node is represented by a uniform sized symbol. However, with text as well as *drawing objects* appearing on the display, it can quickly become cluttered. For example, if 2 nodes are placed close enough to each other, the text displaying each of their names overlaps in a confusing fashion. Colour coding has also been suggested as a method for differentiating nodes, although with support for a large number of variables, the colours required would become difficult to differentiate by the human eye. To conclude, therefore, the BNT successfully addresses many of the problems associated with the display of nodes and in doing so becomes one of the most visually clear products within its group.

Relationships:
As with the convention for displaying graphs, relationships between nodes are represented in the utility by means of a line with arrowhead. Although the utility goes some way to addressing the problem of keeping the display clear by ensuring that as generated, all relationship lines run *down* the page (parent->child), for the display can often become difficult to comprehend. The problem of overlapping and crossing lines has not been. Since it was deemed sensible to ensure that relationship lines were always displayed as straight lines, it has not been possible / practical to ensure that angles on the display are always uniform and acute. As a result of this, the display can often appear cluttered and random, and it is not always possible to establish the structure of the displayed network just by examining the screen. The *dynamic colour coding* of nodes, whereby when a node is selected, it becomes highlighted in red with its parents and children changing to blue and green respectively, goes most of the way to help reduce the ambiguity of the display. The *hide unselected nodes* feature is a unique concept, not offered by any existing product, which enables the complexity of the display to be dramatically reduced by hiding all *irrelevant* parts of the network. Overall, the BNT display of relationships is better than that of any existing software product.

Outside the display of networks, the UI of the BNT conforms to the Microsoft Windows standards and maintains a familiar look and feel for Windows users throughout.

Prior Probabilities:
As discussed for the core component.

Export to .bif:

The BNT enables networks to be saved to .bif (Version 3). It is the only known utility to implement this feature. The facility works as desired, but as with the load function discussed in the *core component*, is currently somewhat limiting. Export to .dsc, .hugin or similar would have been useful options. However, the feature enables version 1 .bif files to be converted to version 3, networks generated from data to be loaded back into the toolkit at a later-date without re-learning from data and permits user-modification of networks already saved in the .bif format. The real power of this feature, however, will materialise when the .bif format is more widely adopted, enabling the BNT to be directly compatible with other software products.

Modify Network Data:

Functionality has been provided to rename nodes and edit the associated probability matrices. Although this meets all the targets initially set out in the specification, competing products go further, enabling users to add and delete nodes etc. Interestingly however, since Version 3 .bif is an XML representation, it should be relatively straightforward for a user of the BNT to add or delete nodes by editing the file directly in notepad or similar. The edit probability matrices option provided by the utility can be regarded as a *secret weapon* of the BNT. Representing the parent-state – state – probability data in tabular form can be argued as being susceptible to confusion, in that it is not always immediately obvious which / where the desired probability entry is. However, in terms of functionality, using this approach allows a far greater deal of flexibility than any other existing product. With the utility supporting up to 100 nodes, the tabular format easily adapts to having a node with anything up to 99 parents. Microsoft's approach of using drop-down-list-boxes to represent parent states may seem initially a neater solution, but as the number of parent node increases, the 1-control-per-parent approach becomes more and more un-user-friendly since the user is required (in the worst case) to enumerate all possible states of the parents manually (potentially 20^{99} combinations). The BNT computes all possible states and displays them enumerated automatically.

Extension 2

Users wishing to generate Bayesian Networks from data would traditionally have used the *Belief Network PowerConstructor*. Comparing the extension 2 results to this product, the BNT emerges very favourably yet almost as a complementary as opposed to competing product. To understand this conclusion we must break extension 2 down into its components and compare and contrast the 2 products.

Interface with a data source:

The BNT is able to produce sensible bayesian networks from a given dataset. To this extent the utility performs well, however on the surface it does have its limitations. The BNPowerConstructor is able to interface with a wide range of data sources, ranging from basic text file through to ODBC compliant databases. In contrast to this, the BNT is only able to *learn* from .dfg formatted text files. In its defence, however, the dfg file format is human readable and data-sources of a manageable size should be able to be converted (by hand) into .dfg without any great hassle. For data-sources of an unmanageable size, however, i.e. those requiring a complete database solution to hold them etc, the BNT learning component is of no use. In practice, however, since each single line of data generally costs money, the majority of targeted users of the system would have a manageable sized data source. Larger data quantity users would probably not consider using Bayesian techniques to analyse their data in the first place – so the system remains eminently usable by its target user-group.

User Specifiable Calculations:

The BNT enables the user to enforce or prevent certain relationships between nodes in a data set. This is a very practical and usable feature not offered by the BN PowerConstructor. Combined with Extension 1, which permits the modification of the link matrices, the BNT is a very versatile tool for learning from data. Like the BN PowerConstructor however, the BNT only implements one learning algorithm, which some users may deem limiting. Additionally, since Extension 2 has been tied in with the rest of the utility, the ability to only generate **singly connected** networks may subjectively render the feature useless for certain applications. In its favour, however, implementing the *spanning tree algorithm* means the BNT acts as an excellent teaching / example tool – particularly for the Intelligent Data course offered in DoC.

Display of calculated networks:

Again, since the learning component has been tied in with the existing UI of the BNT, display of the generated network is deemed to be an improvement over that offered by the BN PowerConstructor.

Extension 3

The data generator provides a potentially very useful facility not offered by any existing Bayesian software. The feature its-self provides an excellent simulator. Users could simply create a .bif (XML) file specifying the relationships they require between variables and then use the generator to produce conforming data sets. Another potential use would be for enlarging existing data sets. Fed initially through the learning algorithm and then concatenated with the results of the data-generator, data-sets can be enlarged to whatever size the user desired as long as the data adapted well to a bayesian representation. We will discuss the impact of this feature later in the evaluation, but let us first consider how the produced results compare to the initial specification.

Output to an importable file format:

The data-generator outputs to the .dfg file format, which although proprietary to the BNT, is human-readable and so can be easily massaged into a format suitable for another analysis tool. A drawback to the .dfg format is the information loss that is caused when saving too it. Node and state names are lost, instead replaced with numerical identifiers.

Customisability:

The most obvious feature for any data-generating product is to be able to modify the number of lines output, the BNT implements this. Flexibility has also been increased by the ability to set which nodes are to be instantiated per data line generated. The usefulness of this option is somewhat open to debate, since the results of the data-generator are difficult to *test* /*verify*.

Evaluation as a piece of software…

Although it is very well to evaluate the BNT in terms of how it compares to what it set out to do, we must also examine the overall usefulness of the system in its targeted market niche.

There are two important subsections of the specification that we have not already discussed which we must do so here. They relate to the robustness / usability of the toolkit and provide a measure of how the toolkit would perform as a finished product offered to the market. They are:

1. Error Reporting / Stability
As a piece of code almost every error that could be raised has been trapped[21] ensuring the software should not (and does not in practice) crash in an uncontrolled way. The vast majority of **implemented** error messages are meaningful to the developer / maintainers of the software and if displayed may confuse or alarm an end user. During testing (and believed to be in the case in practice), however, all error messages that were displayed are meaningful but not always helpful. For example, should a network fail to load for whatever reason, the user is simply told “Incompatible file.” In terms of stability the toolkit fares well, with no undocumented problem arising during testing. In terms of accuracy, the *testing* chapter should be referenced.

[21] Visual Basic’s *OnError* command ensures that every error occurring in every function will be trapped somewhere. The code has been designed to ensure that wherever possible each of these errors are directed to a relevant error handler.

2. Speed

As with accuracy, the *testing* section discusses the issue of speed in greater depth, although it is worth pointing out that compared to the tested competing products[22], the BNT fares very favourably. Indeed no merit points should be lost because of the response / execution time of any part of the system.

From the above we can see that as a stand-alone offering the BNT performs well. It achieves every goal that was specified and as a piece of software, remains usable. But let us consider closer, how the toolkit performs against its rivals…

The Bayesian Network Toolkit Vs.

Although offering a wider range of functionality than any of the existing products, it might be argued with credence that the BNT scratches the surface of a wide variety of different surfaces without covering any one in great depth.

Most fundamentally, the limitation in the scope to singly connected networks raises the most concern. While this ensures the product remains relevant to the DoC Intelligent Data course, it places a big question mark over the toolkit in terms of who would actually use it. Existing products are able to *cope* with multiply connected networks meaning that the vast majority of available / downloadable networks are **not** singly connected. The impact of this depends on your school of thought with regard to Bayesian Nets and a discussion into this is outside the scope of this report. The product was designed with the manipulation / display of *.bif* files at its core and so perhaps as a result of this combined with the singly connected limitation, the toolkit can be regarded as an outstanding array of peripheral functionality with a hollow core. For an eminently usable product, however, this appraisal is probably a little harsh, so let us not dwell but consider some of the other factors of the toolkit.

The BNT is able to deal with networks / datasets of up to 100 nodes each of which may have up to 20 different states. While it has not been possible to verify the fact, it is believed that this offers a greater deal of flexibility than any other existing Bayesian network software.

The Data Generator is a unique offering amongst the Bayesian software community and as such the BNT may be considered **groundbreaking**. It is beyond the scope of this report to compare the BNT to other generic data-generators / simulators / modellers that are available, but within the scope of Bayesian Networks, the generator is a very valuable and usable tool.

The BNT is also unique in that it combines so many different useful peripheral functions into one utility. This surely must act as a roadmap for future application development within the field.

Trying to place the BNT amongst its competitors is a very difficult task. If an end user was happy with the singly connected network limitation, the BNT would almost certainly **be the software of choice.**

BNT as a teaching aid

The toolkit conforms to much of the theory taught in the 4th year Intelligent Data course as intended and as such can certainly be used as intended (if required) as a teaching aid to a certain extent. Lambda and Pi evidence for each node is displayed in the *State Information Table* and *Probability Tables* in the application, which ties well in with the course. Unfortunately, however, due to the implementation of the probability propagation algorithm and internal data structures, the individual Lambda and Pi messages are lost and so not displayed.

[22] See Background

Evaluation of software design process

In a project of this nature and size, it would not suffice just to evaluate the finished product. Time should be taken to reflect on lessons that have been learned during the software development process. This section in effect provides a critique of the project management.

The software was completed before the pre-agreed deadline and addresses every issue in the original specification. An interim report "*Project Outsourcing*" was also produced on time to discuss the future direction of the project as well as highlight important areas of research that had been conducted at that time. The decision to follow a rapid prototyping model appeared well founded with the constant visible results acting as a driving force at each stage of the project. As is the danger with most projects following this design model, the requirement of complete throwaway prototypes was not adhered to. Instead, much of the code in the final product was taken directly from previous *prototypes*. Against this, however, it should be noted that much of the code was completely rewritten when the implementation language changed from VBA to VB. Conducive to good programming style, the new data structures and associated code were designed taking into account lessons that had been learned from the previous version. Code modularity and reuse was encouraged throughout with meaningful distinct modules handling their discrete sections of the program code. Unfortunately, however, when transferred to VB, some of the original module design had to be combined to prevent the use of *global variables*. Every function requiring access to the internal data structures has been combined into one *data* module – against the grain the previous strict separation. The result; a messy module containing code referencing many different aspects of the implementation. Outside this module, however, sections of code have remained well formed and distinct – intended to permit ease of future maintenance.

Testing was conducted throughout the development of the project with careful attention to ensure that each separate function was tested thoroughly at the time of authoring. Pre and Post conditions have been included in every function to ensure that its intended use can be understood. The changing requirements of certain aspects of the project, however, has lead to a certain degree of *code and hack* occurring within some of the more popular functions. Testing at this stage was only minimal, although it is believed (through general test cases) that no serious bugs should have resulted as an effect of this bad programming practice that would have an effect the overall stability of the program.

User interface design was conducted in a strategic way, drawing on the experience of competing software products. No managed end-user testing however was conducted of the design.

Close liaison with the project supervisor was maintained throughout the lifecycle of the project with meetings occurring on a weekly basis to assess progress. In this way, the stakeholders were continually informed of developments and setbacks as they arose.

The workload throughout the project remained high but constant. No final rush ensured the quality of both the software and the accompanying documentation.

With hindsight, of course there are always things that could have been improved upon. With more time, the project would debatably be *better.* However, given the knowledge and resources available it would seem fair to assess the overall project management process as a success.

Final Words…

We have seen discussed earlier in this chapter that the product produced at the end of the day was both a success in terms of the initial specification and in terms of usefulness to the marketplace. We have seen terms such as *ground breaking* and *the software of choice* but we must regard these with some degree of caution as it is clear the software would have been *better* if able to deal with multiply connected networks. Outside its immediate target area, the BNT is of little or no use to those wishing to reason about Bayesian networks. However, the data-generator combined with the software's ability to read / write to the .bif standard, begins to redeem the toolkit even outside its initial scope. As a basis for future expansion, the toolkit provides an excellent base and so perhaps for this reason alone the project as a whole may be deemed a triumph.

The Future

As discussed in the conclusions and evaluation, the BNT project provides an excellent base for future expansion. Some of the possibilities are detailed below:

Multiply Connected Networks

The probability propagation's code has been kept self-contained within the source of the project enabling fast access and simplified modification. The work required to implement a reasoning algorithm able to cope with more than just singly-connected networks is deemed to be minimal and would not have any direct knock on effects to any other aspect of the program. This single modification would undoubtedly make the BNT the most versatile Bayesian Network tool available. The 2 relevant functions that would need to be modified are:

```
PropagateLambdaTo(Node List, from) as void
PropagatePiTo(Node List, from) as void
```

Improved File Format support

This can be broken down further into 2 separate areas:

Read / Write Formats

As it currently stands, the limitation to only importing from and exporting to the *.bif* standard may be regarded as a limitation. However, as the load and save functionality is handled solely by two distinct functions, *read_bif(filename)* and *save_bif(filename)*. The work required to implement support for different file formats reduces purely to investigating the specification of that format. Using bold to indicate the functions that would need to be written, the code required to load different file formats would be:

```
For each node in GetNodeNames(filename)
   Addnode(node, number of states)
   For each state of node
      AddNodeStates(node, state)
   next
Next

For each node in getnodes()
   Addrelationships(node,getparents(node,filename),_
                     getprobabilitymatrix(node,filename))
Next
```

Clearly, just 3 functions are required per new file format, *GetNodeNames*, *Get Parents* and *Get Probability Matrix* each of which should be no more complex than trivial file handling. Similarly for output, the following functions have already been implemented, ensuring that the code requiring authoring would be minimal:

```
GetAllNodes() as a list
GetRootNodes() as a list
GetProbabilityMatrix(node) as a string
GetParentsOf(node) as a list
GetChildrenOf(node) as a list
GetNodeStates(node) as a list
GetProbability(node,state) as a float
SimulateSql(node,state,GivenList,GivenListStates)as a float
```

(returns the probability of a node being in the specified state given the parents in the corresponding states)

Data Generator Formats

Tying the data generator into different file formats would require a little more work. Authoring a utility to convert text file data-sources into .dfg files would be trivial. Providing a link to ODBC compliant databases etc would require greater amounts of code, yet using Microsoft's components, OCX controls etc may reduce the problem closer to plug and play. Internal data structures and associated handling code may have to be re-written since the entire loaded data source is currently held in memory in a data structure of dynamic sized type: `Data(row,column)`. The only access to this data structure is however only used for probability calculations namely: `prob(node,state)`, `proband(node1, state of node1, node2, state of node 2)`and a more complicated function for calculating conditional probabilities (detailed in code comments). It should be noted however, that all the code remains commented out for converting the utility to run off an SQL compatible database that could simply be reused.

Improved probability / state display

A usability study into the *probability tables* feature would be usefully conducted to help reduce the information-overload that can potentially be present when using the toolkit. The probability table code uses a standard `MSHFlexGrid` control to display data and so can easily be modified to display (a subset of?) the data in a different format if required. Some graphical element would be a welcome addition, similar to that offered by the MSBN tool.

Hidden Node Investigation

Originally part of the project proposal, an implementation of user customisable hidden-nodes would prove a powerful addition to the software. The singly connected network limitation would have to be removed first. The idea to include a hidden node investigation in this project was abandoned due to its inapplicability to the network generated from the hepatitis c data set. Such a feature would help introduce more of a research tool component to the software.

Improved Help et al

While the BNT serves as a stable base for expansion, online help and improved error reporting would turn the software into a marketable product. It would be feasible to examine the code function by function and re-write the error handlers, but a more practical approach would be to conduct user tests on the software and makes the changes as error messages are displayed. An area particularly in need of attention is the error handling for bif file loading.

Teaching Aid Improvements

Individual lambda and pi messages are currently lost in the implementation. A slight modification to the data structures and probability calculation algorithms would restore this information and boost the teaching aid functionality of the toolkit. The modifications required to implement the required changes sound worse than they probably would be in practice. The relevant existing data that would need to be changed is (bold shows suggested modification):

```
NodeStatesTable(DYNAMIC 100) as NS_State

Type NS_State
      probability As Double
      lambda_evidence As Double
      pi_evidence() As Double
      Lambda_Messages(DYNAMIC) as tMessage
      Pi_Messages(DYNAMIC) as tMessage
      state_name As String
End Type

Type tMessage
      OriginatorID as Integer
      Value as Double
End Type
```

Code would be needed to ensure that lambda and pi evidence values in `NS_State` was the sum of each of the relevant messages. The function `SetProbability` would need to be modified to store lambda and pi messages instead of directly normalising and storing them as evidence. Finally `PropagatePiTo` and `PropagateLambdaTo` would need minor modification to ensure that messages were sent to instantiated nodes.

MDL Metric or other measure or fit

A measure of fit would be an excellent addition to the toolkit. Currently there is no real way of verifying the networks produced by the learning algorithm. A measure of fit using the MDL metric or similar would help turn the toolkit into a research tool.

Bibliography and Linkology

"Bayes nets – Probably a useful tool, maybe" by Joe Tullio.
http://ebirah.cc.gatech.edu/~jtullio/cs7470/Bayestut.htm

The Intelligent Data and Probabilistic Inference Course Lecture Notes of the Department of Computing at London's Imperial College:
http://www.doc.ic.ac.uk/~dfg/ProbabilisticInference/Bayesian.html

Introduction to Artificial Intelligence
http://www-cse.uta.edu/~cook/ai1/lectures/p.html

The Los Angeles Times: "Improbable Inspiration" – October 28th 1996.
http://www.latimes.com

The Los Alamos Centre for Bayesian Methods: "What is Bayesian Inference?"
http://drambuie.lanl.gov/~bayes/tutorial.htm

"A Tutorial on Learning with Bayesian Networks" Microsoft.
Available at: http://www.research.microsoft.com/scripts/pubDB/pubsasp.asp?RecordID=63

A selection of research papers by Jie Cheng
Available at: http://www.cs.ualberta.ca/~jcheng/lab.htm

The Bayesian Network Repository
http://www-nt.cs.berkeley.edu/home/nir/public_html/Repository/index.htm

JavaBayes Documentation
http://www.cs.cmu.edu/~javabayes/Home/index.html

Appendix A - Example Files

.BIF Representation of the Dog Barking Problem

The following example file, taken from F.G.Cozman's home page, describes the *Dog Barking Problem*, represented in Version 0.3 of the BIF standard.

0.3 Compliant

```
<BIF VERSION="0.3">
<NETWORK>
  <NAME>Dog-Problem</NAME>
<!-- Variables -->
  <VARIABLE TYPE="nature">
    <NAME>light-on</NAME>
    <OUTCOME>true</OUTCOME>
    <OUTCOME>false</OUTCOME>
    <PROPERTY>position = (73, 165)</PROPERTY>
  </VARIABLE>

  <VARIABLE TYPE="nature">
    <NAME>bowel-problem</NAME>
    <OUTCOME>true</OUTCOME>
    <OUTCOME>false</OUTCOME>
    <PROPERTY>position = (190, 69)</PROPERTY>
  </VARIABLE>

  <VARIABLE TYPE="nature">
    <NAME>dog-out</NAME>
    <OUTCOME>true</OUTCOME>
    <OUTCOME>false</OUTCOME>
    <PROPERTY>position = (155, 165)</PROPERTY>
  </VARIABLE>

  <VARIABLE TYPE="nature">
    <NAME>hear-bark</NAME>
    <OUTCOME>true</OUTCOME>
    <OUTCOME>false</OUTCOME>
    <PROPERTY>position = (154, 241)</PROPERTY>
  </VARIABLE>

  <VARIABLE TYPE="nature">
    <NAME>family-out</NAME>
    <OUTCOME>true</OUTCOME>
    <OUTCOME>false</OUTCOME>
    <PROPERTY>position = (112, 69)</PROPERTY>
  </VARIABLE>

<!-- Probability distributions -->

  <DEFINITION>
    <FOR>light-on</FOR>
    <GIVEN>family-out</GIVEN>
    <TABLE>0.6 0.4 0.05 0.95 </TABLE>
  </DEFINITION>

  <DEFINITION>
    <FOR>bowel-problem</FOR>
    <TABLE>0.01 0.99 </TABLE>
  </DEFINITION>

  <DEFINITION>
```

```
    <FOR>dog-out</FOR>
    <GIVEN>bowel-problem</GIVEN>
    <GIVEN>family-out</GIVEN>
    <TABLE>0.99 0.01 0.97 0.03 0.9 0.1 0.3 0.7 </TABLE>
  </DEFINITION>

  <DEFINITION>
    <FOR>hear-bark</FOR>
    <GIVEN>dog-out</GIVEN>
    <TABLE>0.7 0.3 0.01 0.99 </TABLE>
  </DEFINITION>

  <DEFINITION>
    <FOR>family-out</FOR>
    <TABLE>0.15 0.85 </TABLE>
  </DEFINITION>
  </NETWORK>
</BIF>
```

0.1(5) Compliant

Represented in version 0.15 of the .bif format, the file would be:

```
network "Dog-Problem" {
      property "credal-set constant-density-bounded 1.1" ;
}

variable "light-on" { //2 values
      type discrete[2] { "true" "false" };
      property "position = (218, 195)" ;
}
variable "bowel-problem" { //2 values
      type discrete[2] { "true" "false" };
      property "position = (335, 99)" ;
}
variable "dog-out" { //2 values
      type discrete[2] { "true" "false" };
      property "position = (300, 195)" ;
}
variable "hear-bark" { //2 values
      type discrete[2] { "true" "false" };
      property "position = (296, 268)" ;
}
variable "family-out" { //2 values
      type discrete[2] { "true" "false" };
      property "position = (257, 99)" ;
}

probability ( "light-on" "family-out" ) {
      table 0.6 0.05 0.4 0.95 ;
}
probability ( "bowel-problem" ) {
      table 0.01 0.99 ;
}
probability ( "dog-out" "bowel-problem" "family-out" )
      table 0.99 0.97 0.9 0.3 0.01 0.03 0.1 0.7 ;
}
probability ( "hear-bark" "dog-out" ) {
      table 0.7 0.01 0.3 0.99 ;
}
probability ( "family-out" ) {
```

```
                table 0.15 0.85 ;
        }
```

The above description has been abbreviated, but should still enable the utility being authored to deal with basic .bif (not XML) files.

.DSC Representation of the Dog Barking Problem

Represented in Microsoft's .dsc file format, the Dog Problem network is as follows:

```
        diagnostic network ""
        node Light_on
        {
                name: "Light-on";
                type: discrete[2] =
                {
                        "true",
                        "false"
                };
                position: (16690, 12040);
        }

        node Bowel_problem
        {
                name: "bowel-problem";
                type: discrete[2] =
                {
                        "true",
                        "false"
                };
                position: (13765, 10750);
        }
        node Dog_out
        {
                name: "dog-out";
                type: discrete[2] =
                {
                        "true",
                        "false"
                };
                position: (14875, 12070);
        }

        node Hear_bark
        {
                name: "hear-bark";
                type: discrete[2] =
                {
                        "true",
                        "false"
                };
                position: (14815, 13540);
        }

        node Family_out
        {
                name: "family-out";
                type: discrete[2] =
                {
                        "true",
            "false"
      };
      position: (15640, 10720);
}

probability(Light_on | Family_out)
{
      (0): 0.6, 0.4;
      (1): 0.05, 0.95;
}

probability(Bowel_problem)
{
      0.01, 0.99;
}

probability(Dog_out | Family_out, Bowel_problem)
{
      (0, 0): 0.99, 0.01;
      (0, 1): 0.9, 0.1;
      (1, 0): 0.97, 0.03;
      (1, 1): 0.3, 0.7;
}

probability(Hear_bark | Dog_out)
{
      (0): 0.7, 0.3;
      (1): 0.01, 0.99;
}

probability(Family_out)
{
      0.15, 0.85; }
```

Appendix B - User Documentation

Bayesian Network Toolkit

User Guide

Table Of Contents

GETTING STARTED **64**

1. FILE MENU 2
   File Option *64*
   Edit Option *65*
   View Option *65*
   Tools Option *65*
   Help Option *65*
2. NODE LIST 65
3. NETWORK VIEW CANVAS 65
4. STATE INFORMATION TABLE 65
5. KEY 65

LOADING A NETWORK **66**

SAVING A NETWORK **66**

CONSTRUCTING A NETWORK FROM DATA **67**

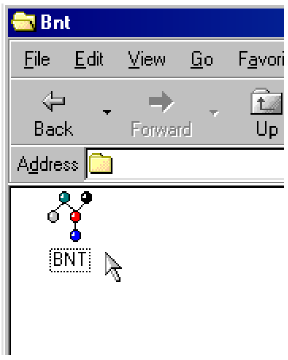
GENERATING DATA **68**

WORKING WITH NETWORKS **69**

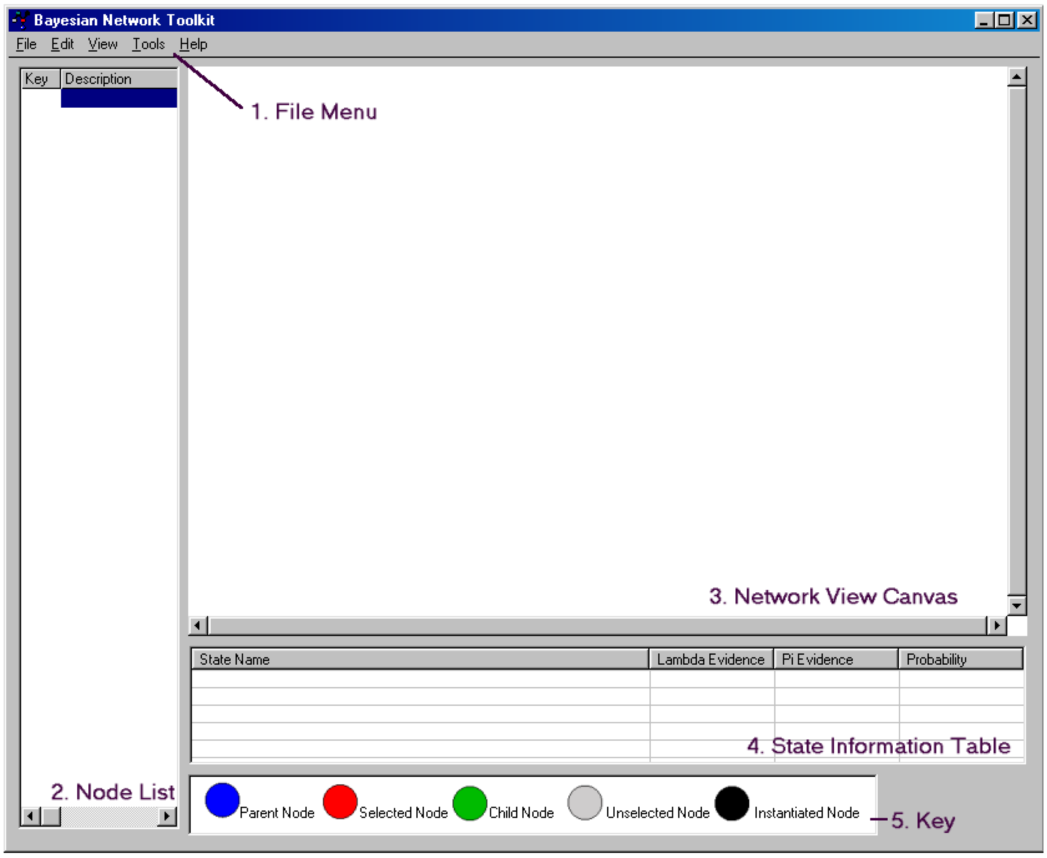
(UN)/INSTANTIATING NODES 69
TRACKING PROBABILITIES 69
EDITING THE PROBABILITY MATRICES 70
HIDING IRRELEVANT NODES 70
RENAMING NODES 71
ALTERING HOW MANY DECIMAL PLACES ARE DISPLAYED 71

TROUBLESHOOTING **72**

Getting Started



Double click on the BNT icon to launch the application. You will be presented with the main toolkit screen (see Screen 1).



Screen 2: BNT Main Screen

The main screen is divided up in to 5 key areas:

1. File Menu

Provides a choice of available actions, comprising:

File Option

- **New:** Unloads any currently loaded network and resets the application back to the load state.
- **Open Network:** Opens a .bif compliant network description file into the application.
- **Load Dataset:** Loads a .dfg compliant dataset and builds a corresponding bayesian network.
- **Save:** Output the currently loaded network as a Version 1 Compliant .bif file.
- **Recent File List:** 4 of the most recently accessed files.
- **Exit:** Quits the application

Edit Option

- **Node:**
  - **Instantiate:** Instantiates the currently selected node into the currently selected state.
  - **Uninstantiate:** Uninstantiates the currently selected node.
  - **Rename:** Prompts for a new name for the currently selected node.
  - **Probability Matrix:** Opens the probability matrix screen for the currently selected node. (See screen 8)
- **Preferences:** Sets to how many decimal places figures should be displayed.

View Option

- **Unselected Nodes:** When selected, the entire network will be displayed in the *Network View Canvas*. When unselected, only the currently active node and its immediate relatives will be displayed.
- **Probability Tables:** Displays the network probability information for all nodes in all states.

Tools Option

- **Data Generator:** Opens the *data generator* to generate .dfg compliant datasets corresponding to the currently loaded network.

Help Option

- **About:** Version and release information.

2. Node List

Displays an ordered list of the nodes in the currently loaded network. The node list comprises 2 columns. The first; a unique key identifying each variable and corresponding to each node displayed in the *Network View Canvas*. The second; the 'name' / description of the node. Clicking on a node in the node list highlights the selected node in the *Network View Canvas* and if selected, hides all other nodes outside the selected nodes' immediate family.

3. Network View Canvas

Displays a graphical representation of the currently loaded network. Relationships between nodes are indicated by a line with an arrow midway denoting direction. Nodes appear as different colours indicating their status (see *Key*). Nodes may be selected by clicking with the mouse. **NOTE** selecting a node in the *Network View Canvas* will **not** hide other nodes outside the selected nodes' immediate family regardless of whether the *view unselected* option is set or not. The structure of the network (position of the variables on screen) may be manipulated by *drag and dropping* nodes.

4. State Information Table

Displays information relating to the currently selected node. A separate entry is displayed in the table for each state of the node with corresponding Lambda and Pi evidence values. The probabilities associated with each state are displayed in the final column. States may be *selected* by clicking on the corresponding table row. When using the *instantiate* option, nodes will be instantiated into their *selected* state.

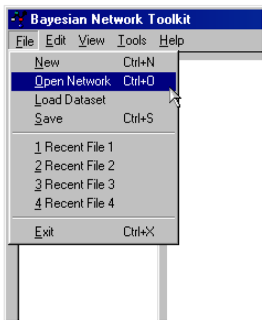
5. Key

A quick reference to the colour coding of nodes displayed in the *Network View Canvas:*

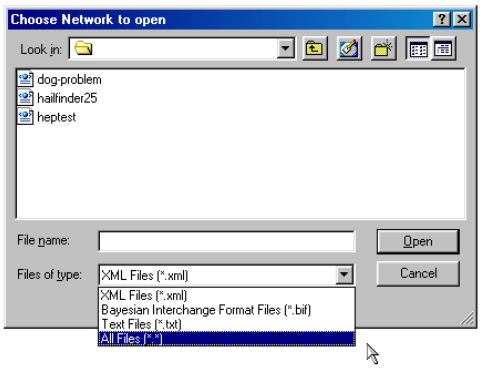
- **Blue:** Indicates the node is a parent of the currently selected node.
- **Red:** Indicates the currently selected node.
- **Green:** Indicates the node is a child of the currently selected node.
- **Grey:** A node outside the immediate family of the currently selected node.
- **Black:** An instantiated node. Note this colour takes precedence over all other node type colours.

.

Loading a Network



1. Invoke the *file open* dialog by selecting <u>*Open Network*</u> from the *File Menu* or with the Ctrl-O shortcut.



dog-problem
hailfinder25
heptest

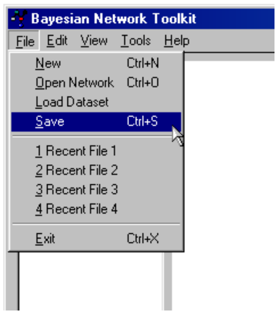
Files of type: XML Files (*.xml)

XML Files (*.xml)
Bayesian Interchange Format Files (*.bif)
Text Files (*.txt)
All Files (*.*)

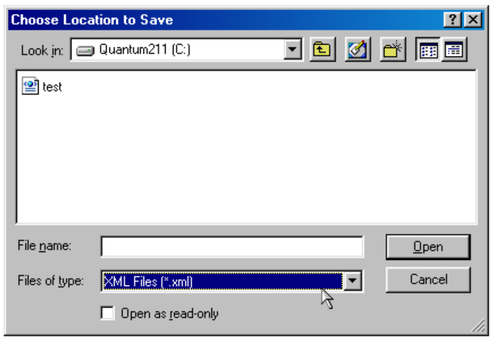
The Bayesian Network Toolkit supports networks saved in the Version 1 (.xml) or Version 3 (.*bif*) of the Bayesian Interchange Format.

2. Having selected a valid file, the network will be displayed in the *main screen*. There may be a slight delay between pressing <u>*Open*</u> and the network being displayed. This is perfectly normal. If the network fails to load, no change will be observed in the *main screen*.

Saving a Network



1. Invoke the *file save* dialog by selecting <u>*Save*</u> from the *File Menu* or with the CTRL-S shortcut.



Look in: Quantum211 (C:)

test

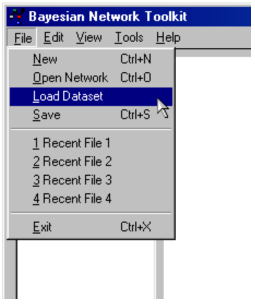
Files of type: XML Files (*.xml)

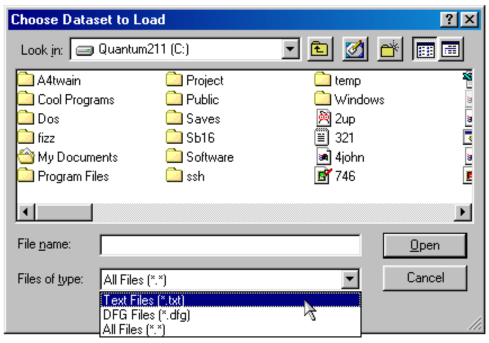
☐ Open as read-only

The Bayesian Network Toolkit will output to Version 1 (.xml) compliant files in the Bayesian Interchange Format. Any loaded network may be saved in this way – whether it was loaded from file or calculated from a dataset.

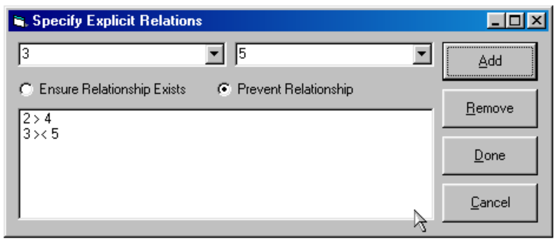
Constructing a Network from Data

1. Invoke the *learn data* dialog by selecting *Load Dataset* from the *File Menu*. The toolkit supports dataset saved in .dfg compliant files.





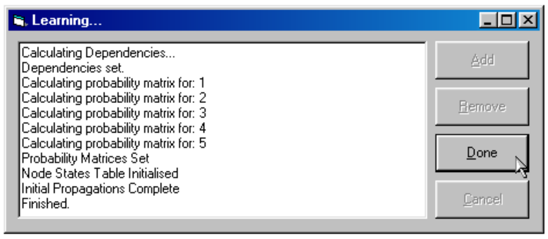
2. Having selected a valid file, the utility will read the basic structural information contained in the dataset. The *Set Explicit Relations* dialog will be displayed.


Screen 2: Set Explicit Relations Dialog

```
2 > 4
3 >< 5
```

3. The *Set Explicit Relations* dialog enables the learning algorithm to be overridden when calculating the relationships between nodes. Explicit parent-child relationships may be prevented or enforced. To set an explicit relationship, choose a parent node from the leftmost drop down list box and a child node from the rightmost drop down. Choose whether the type of rule to be enforced (exists / prevent) and click add. Each new rule will be displayed. Explicit relationships are indicated with the > operator whereby *a* > *b* denotes a has been expressly set as being a **direct father** of node b. Relationships to be prevented are donated by the >< operator, whereby *a* >< *b* indicates that a must **not** be a **direct father** of b. Note b may still be a father of a. That is to say, relationships expressed in the *Set Explicit Relations Dialog* are not symmetric. To prevent node *a* from being a direct relation to node *b*, both *a* >< *b* and *b* >< *a* are required.

4. Once the desired explicit relationships (if any) have been set, choose *Done*. The *Learn progress dialog* will be displayed.


Screen 4 The Learn Progress Dialog

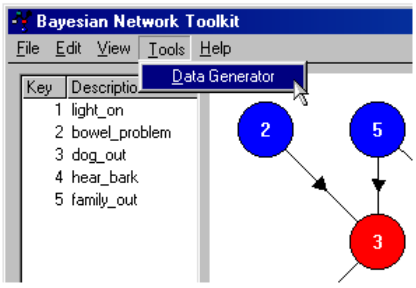
```
Calculating Dependencies...
Dependencies set.
Calculating probability matrix for: 1
Calculating probability matrix for: 2
Calculating probability matrix for: 3
Calculating probability matrix for: 4
Calculating probability matrix for: 5
Probability Matrices Set
Node States Table Initialised
Initial Propagations Complete
Finished.
```

5. The current phase of the learn process will be displayed. They are:

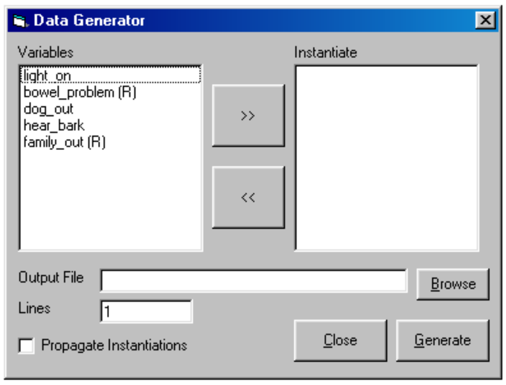
- **Calculating Dependencies:** The algorithm computes the relationships between nodes using the L1 metric algorithm.
- **Calculating Probability Matrices:** A probability matrix for each node in the dataset needs to be calculated.
- **Initialising the Node States Table:** The prior probabilities for the root nodes are set and other node information is stored in the program.
- **Initial Propagation's:** The probabilities for each node are calculated and set based on all nodes being *unobserved*.

6. Once the *Done* button is enabled, the learning process is complete. Click to return to the main screen to view the network.

Generating Data



1. Invoke the *Data Generator* by selecting *Data Generator* from the *Tools menu* option. The data generator *options dialog* will be displayed.



light_on
bowel_problem (R)
dog_out
hear_bark
family_out (R)

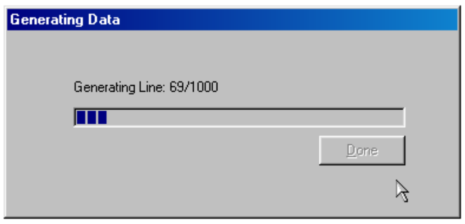
Lines: 1

☐ Propagate Instantiations

Screen 5 The Data Generator Options Dialog

2. Choose which variables should be instantiated by the data generator. Root nodes are donated by the suffix (R). Nodes to be instantiated appear in the right most list. Nodes can be moved to and from the instantiations list with the >> and << buttons respectively. All, no or any subset of nodes may be instantiated. The state of instantiated nodes will be chosen and then instantiated, causing the probabilities of all other nodes in the network to be updated. Conversely, nodes not selected as being instantiated nodes will have their states allocated by the generator, but will not effect the probabilities of any other node in the network. Nodes are instantiated sequentially in the order denoted by the *instantiate list* but with root nodes first. The Propagate Instantiations option is equivalent to selecting all variables as instantiated nodes. Once the instantiations have been selected, specify an output file and enter the desired number of data lines to generate.

3. Select *Generate* to begin. The progress dialog will be displayed.



Generating Line: 69/1000

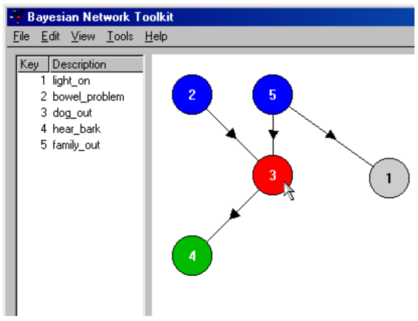
Screen 6 The Progress Dialog

4. The progress bar indicates how far through the generation process is. Progress (time remaining) should be linear. The current data line being generated is also indicated. When the process is complete, the *Done* is enabled. Generation time is dependent on the number of lines to be produced and on the number of instantiated nodes. Performance for the standard *dog-problem* network, instantiating only root nodes is ca. 20 data lines / second.

5. The output file will be written in .dfg format.

Working with Networks

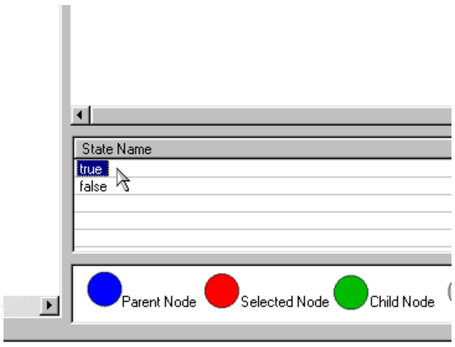
(Un)/Instantiating Nodes



Bayesian Network Toolkit

Key	Description
1	light_on
2	bowel_problem
3	dog_out
4	hear_bark
5	family_out

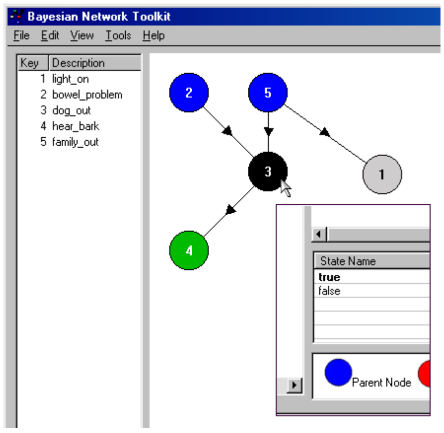
1. Once a network has been loaded. Select the node to be instantiated by clicking directly on the node or selecting the corresponding row in the *node list*.



State Name
true
false

2. Select the state to instantiate the node into by selecting the corresponding row in the *State Information Table*. **Note** if no state is selected, the first state will be instantiated.

3. Choose the *Instantiate* option from the *Edit* menu or use the keyboard shortcut CTRL-I to instantiate the node and calculate the probability propagation's.



Bayesian Network Toolkit

Key	Description
1	light_on
2	bowel_problem
3	dog_out
4	hear_bark
5	family_out

State Name
true
false

4. The node will turn black on the *Network View Canvas* and the instantiated state will appear bold in the *State Information Table* once a node has been instantiated.

5. Nodes can be uninstantiated, simply by selecting the node as with instantiating and selecting the *Uninstantiate* option from the *Edit* menu, or with the keyboard shortcut CTRL-U. Only nodes that have been instantiated may be uninstantiated. If the selected node is not instantiated into any state, the *Uninstantiate* menu option will be disabled.

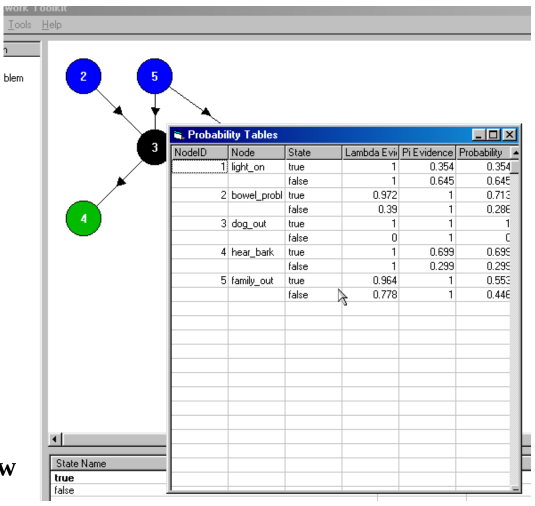
Note it is the convention of this toolkit that the Lambda evidence for instantiated nodes be set as 1 regardless.

Tracking Probabilities

1. Select *Probability Tables* from the *View* menu.

2. A separate window will appear detailing the probabilities and evidence for each state of each node.

3. The Probability Table window will be automatically updates as nodes are instantiated or uninstantiated in the main screen.



Probability Tables

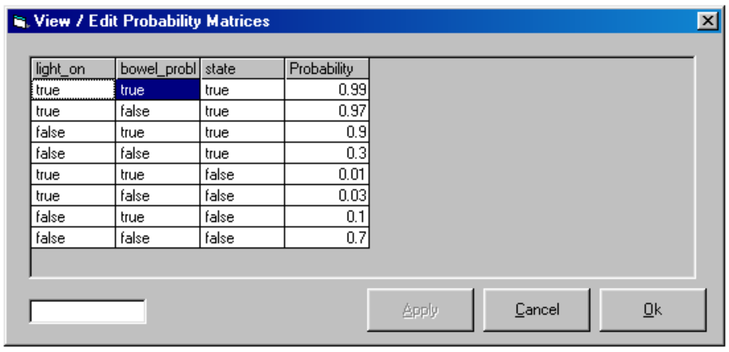
NodeID	Node	State	Lambda Evi	Pi Evidence	Probability
1	light_on	true	1	0.354	0.354
		false	1	0.645	0.645
2	bowel_probl	true	0.972	1	0.713
		false	0.39	1	0.288
3	dog_out	true	1	1	1
		false	0	1	0
4	hear_bark	true	1	0.699	0.699
		false	1	0.299	0.299
5	family_out	true	0.964	1	0.553
		false	0.778	1	0.446

State Name
true
false

Screen 7 The Probability Tables Window

Editing the probability Matrices

1. Select the *Probability Matrix* option from the *Edit / Node* menu. The *View / Edit Probability Matrices* window will be displayed.



light_on	bowel_probl	state	Probability
true	true	true	0.99
true	false	true	0.97
false	true	true	0.9
false	false	true	0.3
true	true	false	0.01
true	false	false	0.03
false	true	false	0.1
false	false	false	0.7

Screen 8 The View / Edit Probability Matrices Window

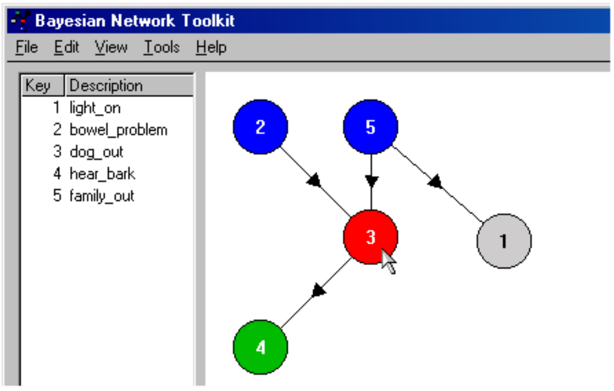
2. The *View / Edit Probability Matrices* window displays the probability matrix for the node currently selected in the *main screen*. **Note** while the *View / Edit Probability Matrices* window is being displayed, it is not possible to select a different node in the *main screen*. The data grid displays the name of each of the node's parents with their corresponding states beneath. The *state* column indicates the state of the currently selected node. The *probability* column displays the entry in the probability matrix corresponding to the state combination indicated in the same row. The first entry in *Screen 7* indicates that if the nodes *Light_on* and *Bowel_problem* are both in state *true* then the probability of the currently selected node being in state *true* is 0.99. etc.

3. To edit a probability, select the corresponding line in the data table. The probability value will appear in the bottom left of the *View / Edit Probability Matrices Window*.

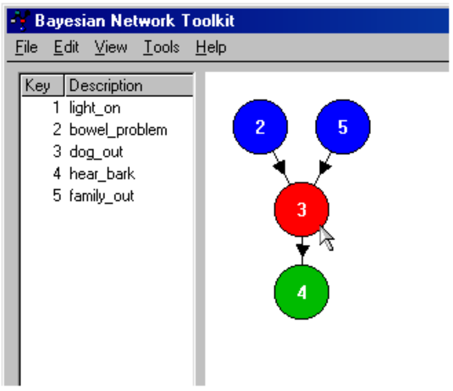
4. Alter the probability as desired and select *Apply* to make the change.

5. To save the changes, click *Ok*. **Note**: The sum of the probabilities for all states of the current node given a particular combination of the parent states **must** equal 0 or 1. This will be checked when the *Ok* button is pressed. Probability matrices not satisfying this property **cannot** be saved.

Hiding Irrelevant Nodes



Key	Description
1	light_on
2	bowel_problem
3	dog_out
4	hear_bark
5	family_out



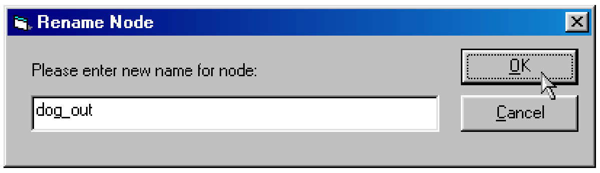
Key	Description
1	light_on
2	bowel_problem
3	dog_out
4	hear_bark
5	family_out

Screen 9 The effect of the Hide Unselected Option

Screen 8 illustrates the effect of using the *Unselected Nodes* option from the *View* menu. Nodes selected from the *node list* will only have their immediate family displayed when the *Unselected Nodes* option is not checked. Clicking on nodes directly when in hide unselected mode will not update the *node view canvas*. All other functionality is unchanged between modes.

Renaming Nodes

1. Select the node to be renamed. Choose the *Rename* option from the *Edit / Node* menu or use the keyboard shortcut CTRL-R. The *Rename Node Dialog* will be displayed.

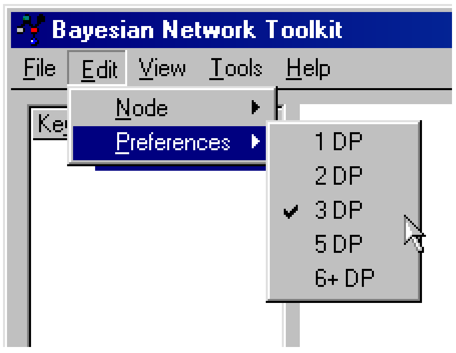


Screen 10 The Rename Node Dialog

2. The old node name will be displayed, simply select and type over with the new name.

3. Changes are saved when the *OK* button is pressed. **Note:** Node names must be unique. A cheque is performed when the OK button is pressed to ensure that no other node already has the new name.

Altering how many decimal places are displayed



Using the *Preferences* option on the *Edit* menu, the number of displayed decimal places may be altered to 1,2,3,5 or 6.

This option will effect figures in the *State Information Table* and *Probability Tables Window*. It will have no effect on the accuracy of the calculated results, since internally the number of decimal places is fixed.

Troubleshooting

1. **Network Fails To load**:
   This is caused by a corrupt network file. Check the file is compliant with Version 1 or 3 of the Bayesian Interchange Format standard.

2. **Application appears not to respond after loading a network:**
   This is quite normal. Some larger networks may take up to 30 seconds to load. The application should resume functioning after this period.

3. **I set an explicit rule to prevent the relationship a->b with the a><b operator, but node b has appeared as a direct ancestor of a.**
   This is correct. The >< operator is not symmetric. To prevent node *b* from being an ancestor of node *a*, the rule *b* >< *a* must also be set.

4. **I've generated a data set, but when I feed it back into the toolkit (by the learn data option) – I don't get the same network as I started with!**
   This is quite normal. The *accuracy* of the generated dataset is dependent on the number of lines of data generated. The more, the more stable the network becomes. It is also mathematically quite correct for the network **not** to be identical to the original causal network.

5. **I've generated a data set, but when I feed it back into the toolkit the node names are lost!**
   Unfortunately, the .dfg file format does not support variable names. Nodes are renamed numerically with root nodes first.

6. **I made a mistake and tried to undo with CTRL-U – but it doesn't seem to work!**
   It shouldn't. The Bayesian Network Toolkit goes slightly against standard and uses the keyboard shortcut CTRL-U to **uninstantiate nodes.**

7. **I altered some entries in the Probability Matrix for a node and clicked apply each time, but the changes don't appear to have taken effect.**
   Changes are not **saved** unless the OK button is clicked. Modifying entries and clicking apply will not be saved if the Cancel or close button is selected.

8. I altered some entries in the Probability Matrix for a node, but now the toolkit appears to have undone all my instantiations.
   This perfectly normal and is a limitation of the program. Nodes can be re-instantiated with the new probability matrix taking effect.

Appendix C - Testing

Testing of the Data Generator

The XML representations of the output generated by the BNT for each of the tests in this section have been removed from this section due to size considerations. Instead a link to each relevant xml file is provided under the section headings.

Test 1: Instantiating Root Nodes

http://www.94.parkview-court.net/project/test1.xml

Test 2: Instantiating All Non-Root Nodes

http://www.94.parkview-court.net/project/test2.xml

Test 3: Instantiating All Nodes

http://www.94.parkview-court.net/project/test3.xml

Test 4: Instantiating 3 Nodes

http://www.94.parkview-court.net/project/test4.xml

Test 5: Instantiating All Non-Root Nodes with 1000 lines of data

http://www.94.parkview-court.net/project/test5.xml

Further testing of the Learning Algorithm

The code for the short utility described in this section in the main body of the report follows:

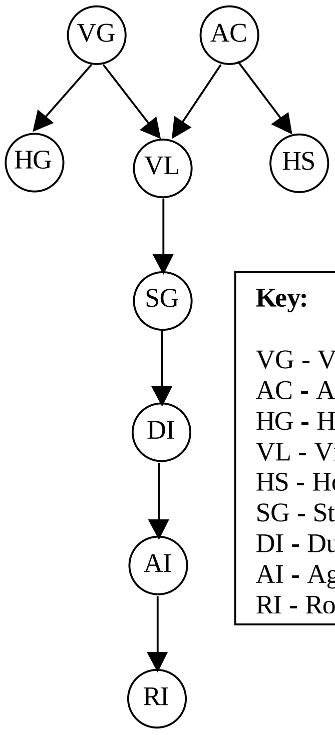
```
lastrnd = RND * 1
test = .3
OPEN "C:\gendata.txt" FOR OUTPUT AS #1
FOR j = 1 TO 5000
      FOR i = 1 TO 18
              IF lastrnd < .3 THEN
                      PRINT #1, 2;
              ELSEIF lastrnd < .6 THEN
                      PRINT #1, 1;
              ELSE
                      PRINT #1, 3;
              END IF
              lastrnd = RND * 1
      NEXT
      PRINT #1,
NEXT
```

Appendix D - Hepatitis C Investigation Results

Bayesian Network Representation of the Dataset

The following diagram represents the network produced by the utility when using the Hepatitis C dataset as learning data. The corresponding XML representation can be downloaded from:

http://www.94.parkview-court.net/project/HepititisC.xml



Key:

VG - Viral Genotype
AC - Alcohol Consumption
HG - Host Genotype
VL - Viral Load
HS - Host Gender
SG - Study Group
DI - Duration of Infection
AI - Age of Infection
RI - Route of Infection