User's Guide for the PNLtoGraphviz Package (Version 1.0)

by Imme Ebert-Uphoff (imme@users.sourceforge.net)
January 21, 2006

Note: This may or may not be the most recent version of this document.
The newest version is always available at www.DataOnStage.com.

Abstract

This document describes an interface that writes the graph structure of a network to a file in the *DOT* language used by the *Graphviz* package. The powerful graph layout algorithms of the *Graphviz* package can then be applied for the layout and rendering of the graph. The interface is currently implemented for PNL networks of the type *BayesNet*, *DBN* and *LIMID*.

As a new feature of this release this package uses the LinkConnectionStrength package to also generate graphs of Discrete Bayesian Networks that include connection strengths between node pairs and link strengths of arcs.

Contents

1 Overview

Graphviz is an open source visualization software that provides several powerful graph layout algorithms. The Graphviz layout algorithms take descriptions of graphs in a simple text language (called the *DOT*-language) and create images in most standard output formats (including jpg, gif, imap, mif, mp, pcl, ps, ps2, png, svg, vrml and many others).

The set of PNLtoGraphviz functions described in this document simply generate a representation of the graph in the *DOT*-language and write it to a file which can be read from the Graphviz package. Thus in order to use this interface the Graphviz package must also be installed, but that is very easy to do. The prerequisites can be summarized as follows:

1. Add files PNLtoGraphviz.hpp, PNLtoGraphviz.cpp, LinkConnectionStrength.hpp and LinkConnectionStrength.cpp to your PNL code and link them.
   (Available at www.DataOnStage.com.)
   You may also want to download the files SampleUsePTG.cpp, SampleUseLCD.cpp, models.cpp and models.h that demonstrate the use of the packages.

2. If not already installed, get the Graphviz package.
   (Available at http://www.graphviz.org for most platforms. *Very* easy to install.)

The use of the interface is also in two steps:

1. Within your PNL code call one of the PNLtoGraphviz functions for your network (type BayesNet, DBN or LIMID). This creates a graph file in DOT language.

2. Outside of PNL: Open the DOT-file from Graphviz to generate automatic layout and rendering. A single command is generally sufficient. For example, to generate a postscript image on a Linux system from the file "graph.dot" generated by Step 1 one simply types in the command line:

   ```
   dot -Tps graph.dot -o graph.ps
   ```

   Generating graphs on Windows or other platforms it just as easy (see instructions at http://www.graphviz.org).

2 Organization of this Document

Sections 3 to 5 describe the use of PNLtoGraphviz to generate graphs for PNL networks of type BayesNet, DBN and LIMID. Section 6 describes how to generate graphs for discrete Bayesian Networks that also display link strength and connection strength. Section 7 asks for feedback and suggestions from users.

3 Graph Representation and Examples

This section discusses how networks of different types are drawn in default mode. Sample output is provided here (and in the sample code) using the network models from the file models.cpp in the PNL/high/Demo directory of PNL Release 1.0[1]. The interface syntax is discussed in Section 4 and options for customizing the node shapes are discussed in Section 5.

[1]Note that the AsiaModel in file PNL/high/DEMO/models.cpp of PNL Release 1.0 has a typo. The correct version should read: $P(Bronchitis = True|Smoking = True) = 0.6$, $P(Bronchitis = False|Smoking = True) = 0.4$.

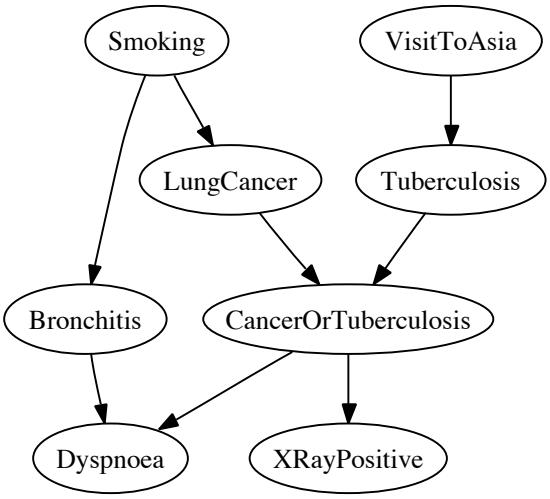

Figure 1: AsiaModel – a BayesNet with only discrete nodes

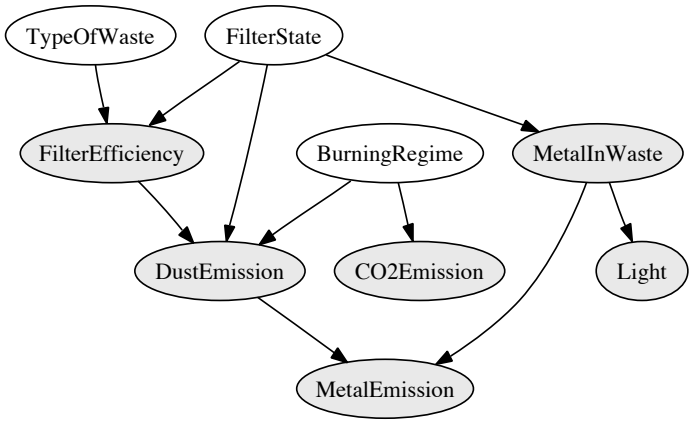

Figure 2: WasteModel – a mixed BayesNet containing both discrete and continuous nodes

3.1 Type BayesNet

The graph representation for a network of type BayesNet is relatively straight forward: in default mode each node is represented as an ellipse containing the name of the node and arrows indicate all parent-child relationships, see for example Figure 1.

Each node can be either discrete or continuous. To distinguish between them the default mode gives a slight shading to continuous nodes. For example in Figure 2 the nodes *TypeOfWaste*, *FilterState* and *BurningRegime* are discrete and all other nodes are continuous.

3.2 Type DBN

For a Dynamic Bayesian Network only slices 0 and 1 are shown in the graph. In principle, one can treat the resulting graph as if it were a BayesNet for the representation. That would result in a layout shown in Figure 3 which is a bit confusing. Thus, nodes of a Dynamic Bayesian Network are instead grouped into two clusters, one containing all nodes of Slice 0, another for Slice 1. The result is shown in Figure 4 which is much easier to read.

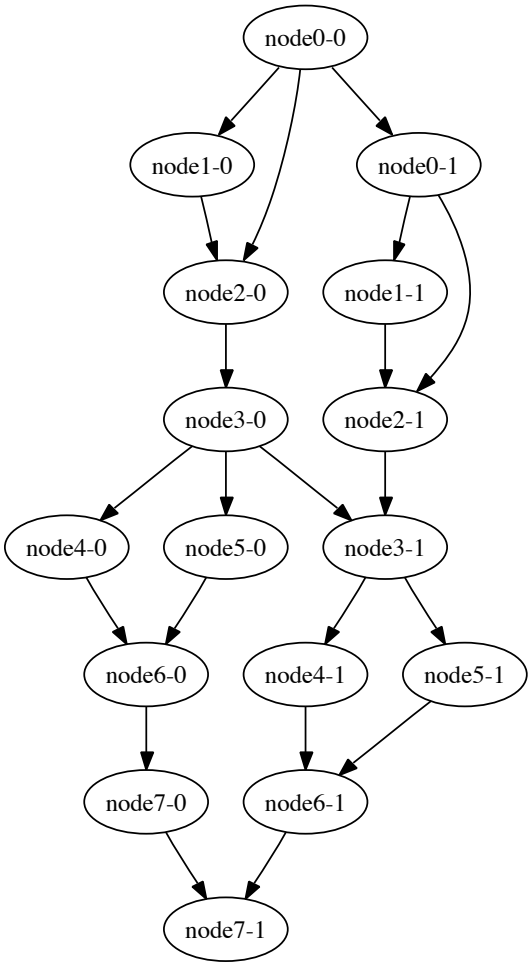

Figure 3: KjaerulfsBNetModel – a DBN *printed as BayesNet*

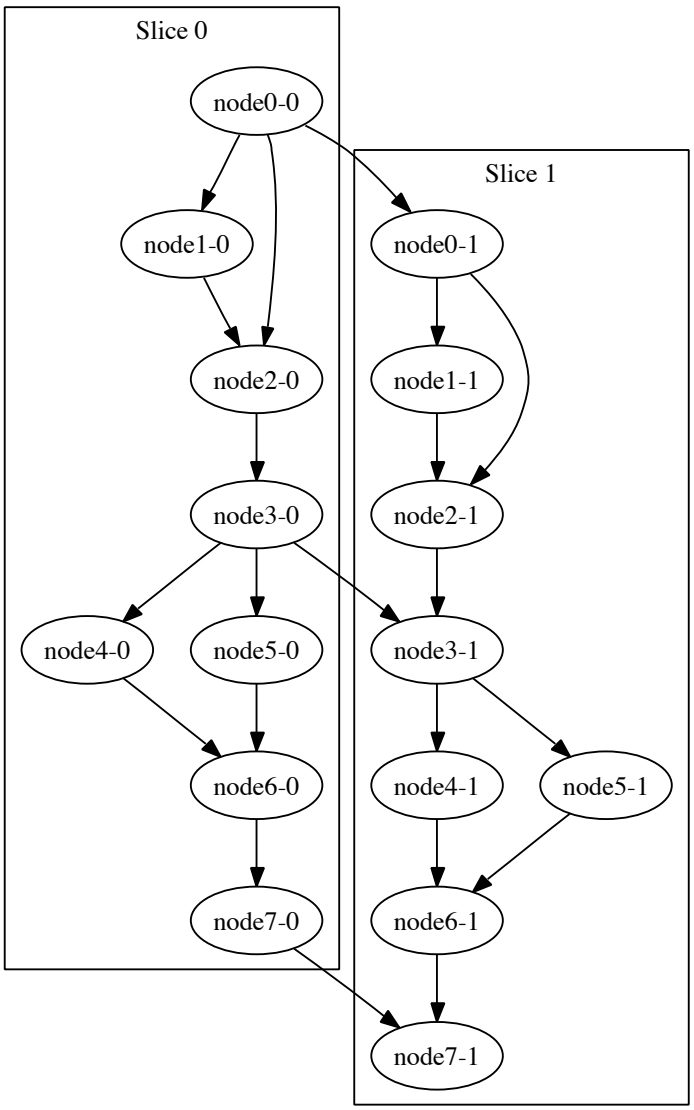

Figure 4: KjaerulfsBNetModel – a DBN *printed as DBN*

Suggestions for future work: There is still room for improvement even in the cluster version. It would be nice to have the two slices aligned horizontally, and to have the nodes of slices 0 and 1 arranged identically. Any ideas on how to achieve this in the *dot*-language would be very much appreciated (alignment is probably doable, but identical arrangement may not be).

3.3 Type LIMID

Influence diagrams can be treated essentially like networks of type BayesNet for the layout. The only question is how to represent the different node types, namely chance, decision and value nodes. The default representation displays chance nodes as ellipses, decision nodes as rectangles and value nodes as diamonds. See Figure 5 for an example.

3.4 Type MRF

This type is not yet implemented. Is there any interest in this? Any feedback would be appreciated (I don't think it would be hard to implement).

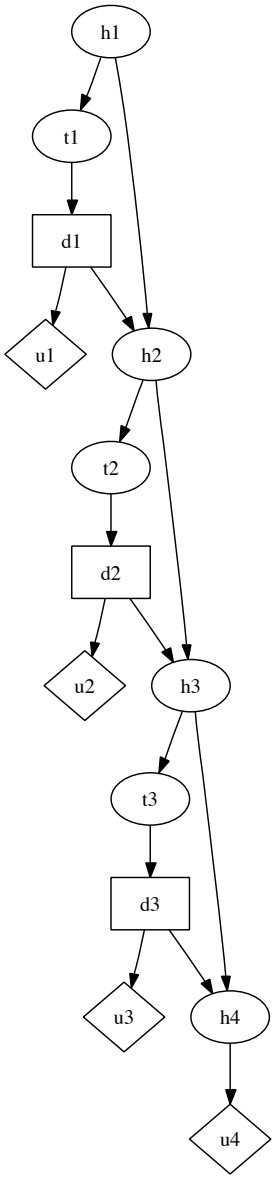

Figure 5: PigsModel – a LIMID (influence diagram)

4 Syntax

A separate function *PNLtoGraphviz* is available for each of the four types of network models, *Bayesnet*, *DBN*, *LIMID* and *MRF*. However, the interface for MRF has not yet been implemented, thus that function for now only prints an error message on the screen.

The syntax for the three functions that are meaningful at this point are:

```
int PNLtoGraphviz ( BayesNet * net, const string & filename,
               map<string,string>
               customized_node_shape=map<string, string>() );

int PNLtoGraphviz ( DBN * net, const string & filename,
               map<string,string>
               customized_node_shape=map<string, string>() );

int PNLtoGraphviz ( LIMID * net, const string & filename,
               map<string,string>
               customized_node_shape=map<string, string>() );
```

The function variables[2] have the following meaning:

- *net* is a pointer to the network whose graph we want to print.
- *filename* is the name of the output file. While any name is allowed, an ending of '.dot' is recommended.
- *customized_node_shape* is an **optional parameter** that can be used to overwrite the default shapes for the various node types. See Section 5 for more details.

Here is a sample call of the function for a BayesNet that uses the default settings for node shapes, thus only has two parameters:

```
BayesNet * Bnet;
BNet = AsiaModelCorrected();  // corrected model from PNL/high/Demo/model.cpp
PNLtoGraphviz( Bnet, "AsiaModel.dot" );
```

This code creates the file *AsiaModel.dot*. Calling the algorithm *dot* of the Graphviz package afterward with input *AsiaModel.dot* produces the layout in Figure 1.

An example of customizing the node shapes is given in the following section.

5 Default and Custom Node Attributes

PNL uses five different node types, namely *discrete*, *continuous*, *chance*, *decision* and *value*. The default node shapes for those five types and their corresponding *DOT* attributes in the default mode of PNLtoGraphviz are listed in Table 1.

By using the optional parameter *custom_node_shape* in function *PNLtoGraphviz*, the node shapes can easily be changed. For a list of possible node shapes and other node attributes, see the Graphviz

[2]Note that *BayesNet*, *DBN* and *LIMID* are of course defined in the "pnlw" namespace and *string* and *map* are defined in the "std" namespace (std::string and std::map).

Node type	Default Shape	Corresponding *DOT* attributes
discrete	Ellipse	[shape=ellipse]
continuous	Ellipse with interior shading	[shape=ellipse,style=filled,fillcolor=gray90]
chance	Ellipse	[shape=ellipse]
decision	Rectangle	[shape=box]
value	Diamond	[shape=diamond]

Table 1: Default Node Shapes

manual entitled "Drawing graphs with *dot*" by Gansner et al. [1]. All node attributes that work with *dot* can be used.

Below is an example of changing the node attributes. The set of node attributes for any node type is enclosed in brackets and separated by commas. Note that only those node types for which the node attributes are to be changed have to be included. For the other node types, the default values are used:

```
using namespace std;

map<string,string> my_node_shape;
my_node_shape["discrete"]="[shape=doublecircle]";
my_node_shape["continuous"]=
   "[shape=doubleoctagon,style=filled,fillcolor=gray80]";

// load a mixed discrete-continuous graph
BayesNet * Bnet = WasteModel();

// graph with default node shapes
cout << "Mixed discrete-continuous graph with default settings.";
cout << endl;
PNLtoGraphviz( Bnet, "WasteModel.dot" );

// SILLY graph with customized node shapes
cout << "Mixed discrete-continuous graph with customized settings.";
cout << endl;
PNLtoGraphviz( Bnet, "WasteModel_silly.dot", my_node_shape );
```

Be careful not to mistype the node type, e.g. to type *continuos* instead of *continuous* as index of *my_node_shape*. This type of typo would not cause an error message, but it would simply not change the desired node shape.

The result of file "WasteModel_silly.dot" (after calling *dot*) is the silly graph in Figure 6, while the result for "WasteModel.dot" is shown in Figure 2.

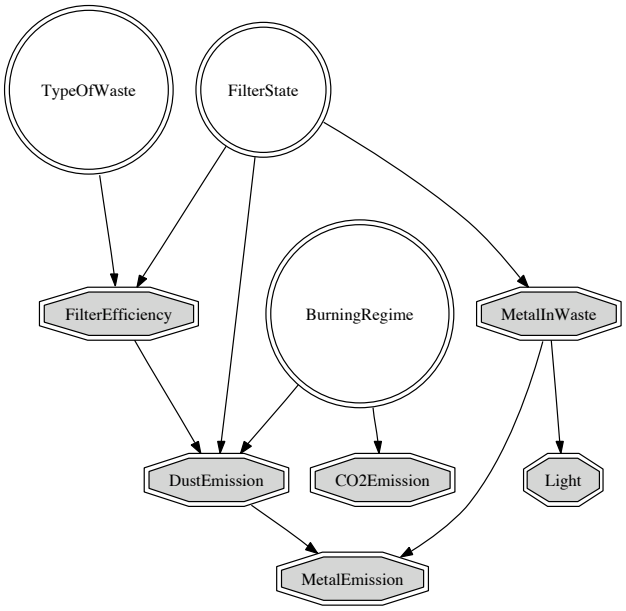

Figure 6: Silly version of WasteModel with custom shapes for discrete and continuous nodes

6 Additional Graph Representations for Discrete Bayesian Networks – Displaying Link Strengths and Connection Strengths

PNLtoGraphviz can work with the LinkConnectionStrength package to generate graphs for discrete Bayesian Networks that include information such as entropy, connection strength between any two nodes and link strength of any arc.

The following measures are implemented in the LinkConnectionStrength package:

- Entropy is used to measure the uncertainty in a single node.
- Mutual information is used to measure connection strength between any two nodes.
- Two measures derived from mutual information are available to measure link strength of any specific link (*True Average Link Strength* and *Blind Average Link Strength*).
- In addition, mutual information *percentage* and link strength *percentage* are provided to measure the *percentage* of the existing uncertainty that has been removed.

Complete definitions of all of the above terms are given in the documentation of the LinkConnectionStrength package [2].

6.1 Available Graphs for Discrete Bayesian Networks

The PNLtoGraphViz package now makes four new graph printing routines available for discrete Bayesian Networks, employing the LinkConnectionStrength package [2] to actually calculate the values for the various measures.

Input variables used in the functions below:

- net: pointer to BayesNet
- filename: name of output file
- target_node_index: index of node relative to which Mutual Information (Percentage) of all other nodes is calculated.
- formula: name of formula to be used for link strength calculation.
  Current options are "TrueAverage" and "BlindAverage".
- want_percentage: True or False; Calculates absolute value of link or connection strength if "True", otherwise percentage value.
- customized_node_shape: optional parameter that allows one to change the look of the nodes in resulting graph (see PNLtoGraphviz documentation for its use).

Output variable:

- Each function *should* return "1" if the file was created successfully and "0" otherwise. However, this features hasn't been tested extensively.

Available Functions:

1. **Entropy Graph for Discrete BN**

```
int PNLtoGraphviz_with_Entropy ( BayesNet * net,
                                 const std::string & filename);
```

Functionality: Creates graph including entropy for each node. The entropy is shown in the graph as a number below the node name.

2. **Graph with Connection Strengths for Discrete BN**

```
int PNLtoGraphviz_with_MI ( BayesNet * net, const std::string & filename,
                            int target_node_index, bool want_percentage);
```

Functionality: Creates graph including mutual information (or mutual information percentage) relative to target_node. The target node is indicated by an octagonal node shape and its entropy included underneath the node name. Connection Strength of all other nodes relative to this one is displayed by (1) number underneath the node name and (2) gray scale of node.

```
int PNLtoGraphviz_with_MI ( BayesNet * net, const std::string & filename,
                            const String target_node_name,
                            bool want_percentage);
```

Functionality: Same as above, but using Node Name instead of Node Index for target_node.

3. **Graph with Link Strengths for Discrete BN**

```
int PNLtoGraphviz_with_LS ( BayesNet * net, const std::string & filename,
                            const std::string & formula,
                            bool want_percentage,
                            std::map<std::string,std::string>
                            customized_node_shape=
                            std::map<std::string, std::string>() );
```

Functionality: Create graph with link strengths (True Average or Blind Average formula, absolute value or percentage). Link Strength of each arc is displayed by (1) number next to the arrow and (2) gray scale of arrow. If an arrow is very weak and would be almost invisible, it is replaced by a dashed arrow.

6.2 Sample Output

This section only provides sample output for a single example. For lots of other examples, including sample code that generates the plots, please see the LinkConnectionStrength User's Guide [2]. In particular, the definition and meaning of link strength and connection strength can be found there.

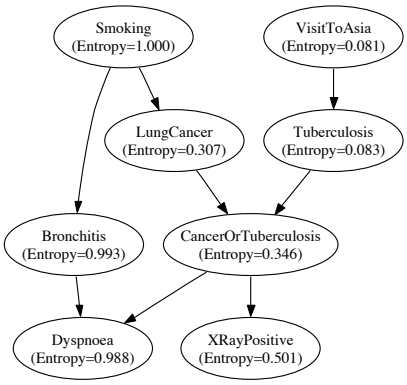

Figure 7: Entropy Graph for Asia Model.

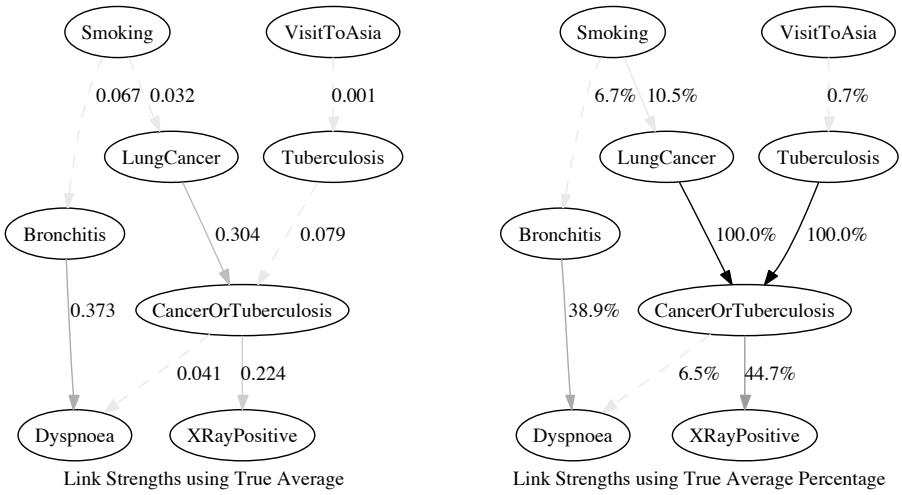

Figure 8: True Average Link Strength (left) and Percentage (right) for Asia Model.

Figures 7 to 9 show the output for various types of graphs for the corrected Asia Model[3]. Figure 7 shows the entropy of each node underneath its name. Figure 8 displays the True Average Link Strength of all arcs. The results are displayed both as numbers next to the arrows and as the gray scale of the arrows. Very light arrows that would be hard to see are shown as dashed arrows instead (in slightly darker gray than they ordinarily would be).

Figure 9 shows some selected mutual information graphs. The target node is shown in a hexagonal shape (with the entropy value underneath the name). All other nodes are shown with regular shape and the number underneath each name indicates the connection strength between that node and the target node. In addition, the darker a node, the stronger is its connection to the target node.

For more information on how to interpret the graphs and the measures, please see the documentation of the LinkConnectionStrength package [2].

[3]Note that the AsiaModel in file PNL/high/DEMO/models.cpp of PNL Release 1.0 has a typo. The correct version should read: $P(Bronchitis = True|Smoking = True) = 0.6$, $P(Bronchitis = False|Smoking = True) = 0.4$.

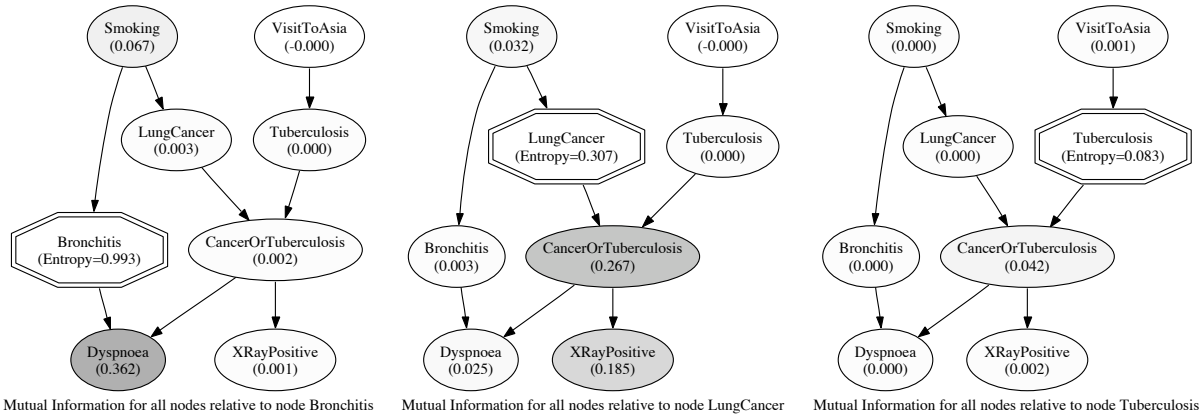

Figure 9: Connection Strength (Mutual Information) relative to node *Bronchitis* (left), *LungCancer* (center) and *Tuberculosis* (right) for Asia Model.

7 Comments or Suggestions?

There are certainly many things that could be improved or added to this interface. I would be very grateful for any feedback, including bug reports, suggestions for improvements or any other comments. Please send your comments to *imme@users.sourceforge.net*.

8 References

[1] Gansner, E., Koutsofios, E. and North, S., "Drawing graphs with *dot*", Feb. 2002. Available at http://graphviz.org/ by clicking on "Documentation" and selecting "User's Guide: dot".

[2] Ebert-Uphoff, I., "User's Guide for the LinkConnectionStrength Package (Version 1.0) – A PNL Package for the Vizualization of Entropy, Link Strength and Connection Strenghts in Discrete Bayesian Networks". Available at www.DataOnStage.com.