# The Bayesys user manual

Anthony C. Constantinou[1, 2]

Version 1.71[1]

i. Bayesian Artificial Intelligence research lab, Risk and Information Management (RIM) research group, School of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK, E1 4FZ.

E-mail: a.constantinou@qmul.ac.uk

ii. The Alan Turing Institute, UK, British Library, London, UK, NW1 2DB.

www.agenarisk.com

[1] **Citation:** Constantinou, A. (2019). The Bayesys user manual. *Queen Mary University of London*, London, UK. [Online]. Available: http://bayesian-ai.eecs.qmul.ac.uk/bayesys/ or http://www.bayesys.com

# Table of Contents

## Copyright notice

## Acknowledgements

1. This work is supported by the ERSRC Fellowship project EP/S001646/1 "*Bayesian Artificial Intelligence for Decision Making under Uncertainty*" [1], by the project partner Agena Ltd, and by The Alan Turing Institute.

2. Bayesys makes use of the AgenaRisk Bayesian Network software. Bayesys enables users to automatically convert a learned graph into a parameterised Bayesian Network in AgenaRisk.

3. Bayesys makes use of the open-source GraphViz software. The GraphViz Java API, written by Laszlo Szathmary, is incorporated into this project. Link to Laszlo's Github page: https://github.com/jabbalaci/graphviz-java-api

4. I would like to thank the following people for their feedback and recommendations: Kiattikun Chobtham, Yang Liu, Dr Neville Kenneth Kitson and Dr Zhigao Guo.

# Introduction

This manual describes the Bayesys open-source Bayesian network structure learning system. This is a JAVA Netbeans project in active development. This manual refers to the BETA version 1.7. Frequent revisions will continue to be published at www.bayesys.com. (or http://bayesian-ai.eecs.qmul.ac.uk/bayesys/). For an overview of any key revisions made between released versions, refer to the appendix.

## 1. Getting started [with the Netbeans project]

Bayesys is implemented in JAVA Netbeans IDE. What is described in this document applies to the Windows OS. While the project also runs on MAC and Linux OS, the functions that relate to the third-party Graphviz software may have compatibility issues on MAC and Linux and may fail to produce the graphs in PDF; although you will be able to get the graphs in CSV.

To set up the Netbeans project:

a) Download the latest Java Development Kit (JDK), appropriate for the OS you use, from https://www.oracle.com/java/technologies/javase-downloads.html

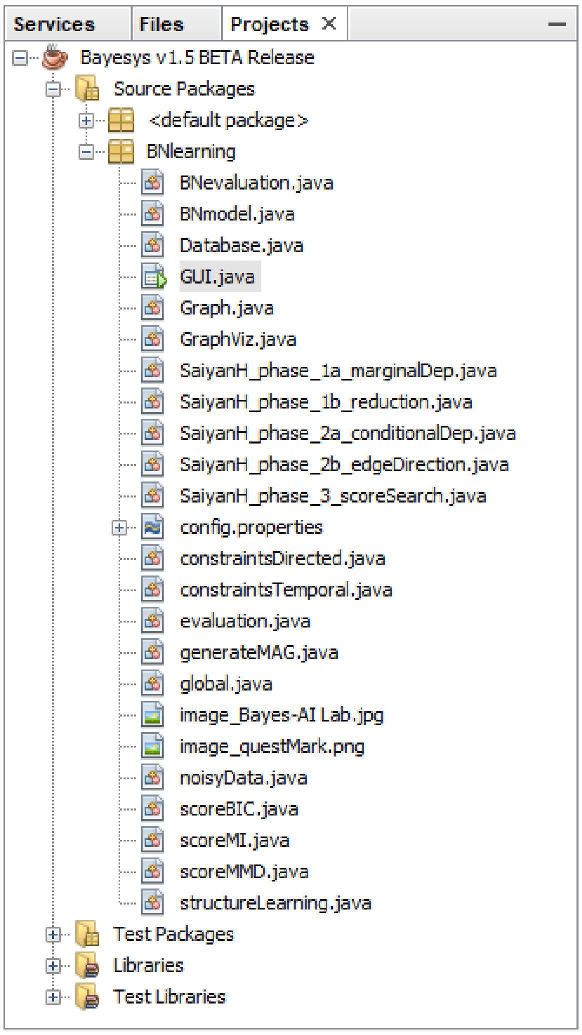
b) Download the latest full version of Netbeans IDE from https://netbeans.org

c) Download the Bayesys Netbeans project from www.bayesys.com. The project is compressed in a ZIP file extension. Extract the contents of the file to your preferred directory.

d) OPTIONAL: You can skip this step if you are not an AgenaRisk user license holder. Open-source AgenaRisk licenses, linked to this implementation, are yet to be issued. Download and install AgenaRisk from www.agenarisk.com.

e) Run Netbeans IDE. Go to *File*, *Open Project*, and browse to the directory you have extracted the Bayesys project. You will see ⊞ Bayesys v1 or a newer version, indicating that the folder is readable as a Netbeans project. Select the readable folder and click on *Open Project* to load Bayesys into Netbeans.

Once the project loads into Netbeans, you will be able to view its contents as shown in Fig 1.1. If not, click on ⊞ to expand the 'BNlearning' directory.



```
Bayesys v1.5 BETA Release
    Source Packages
        <default package>
        BNlearning
            BNevaluation.java
            BNmodel.java
            Database.java
            GUI.java
            Graph.java
            GraphViz.java
            SaiyanH_phase_1a_marginalDep.java
            SaiyanH_phase_1b_reduction.java
            SaiyanH_phase_2a_conditionalDep.java
            SaiyanH_phase_2b_edgeDirection.java
            SaiyanH_phase_3_scoreSearch.java
            config.properties
            constraintsDirected.java
            constraintsTemporal.java
            evaluation.java
            generateMAG.java
            global.java
            image_Bayes-AI Lab.jpg
            image_questMark.png
            noisyData.java
            scoreBIC.java
            scoreMI.java
            scoreMMD.java
            structureLearning.java
    Test Packages
    Libraries
    Test Libraries
```

**Fig 1.1.** The list of classes and other files based on Bayesys BETA v1.5.

To run the project, click the run button. If the button is disabled, make sure that one of the classes under the *BNlearning* directory is selected; e.g., the class GUI.java.

# 2. Learning a Bayesian network

What follows describes the structure learning process based on the SaiyanH hybrid Bayesian Network (BN) structure learning algorithm. Currently, this is the only algorithm available in Bayesys. The full description of SaiyanH can be found in [2]. Note that SaiyanH is based on Saiyan [3], which was an experimental early version no longer available in Bayesys.

To initiate structure learning and generate a Directed Acyclic Graph (DAG), select the process *Structure learning* under tab *Main*. This will also enable tab *Learning* which allows the user to change the parameter inputs of SaiyanH. [NOTE: this feature is currently disabled for further testing; you may run SaiyanH with its default parameter inputs].

## *2.1. Structure learning from data*

Place your data file named *trainingData.csv* placed in folder *Input*.

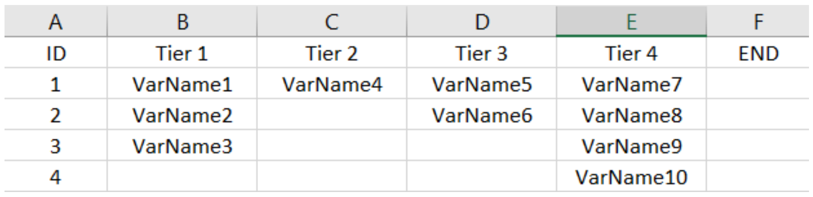
- Each column in *trainingData.csv* should represent a variable; the data file must not include an ID column.
- The dataset should not include missing data values. You can address this issue by replacing all empty cells with a new state called *missing*. This way, the complete dataset can then be used for structure learning under the assumption that missing values are not missing at random.
- The dataset should only include discrete variables. The algorithm assumes a unique discrete state for each unique variable value.

### *2.1.1. Structure learning from both data and knowledge* process

The structure learning process also enables subtab *Knowledge-based constraints* under tab *Learning*. [NOTE: this feature is currently disabled for further testing]. The user can incorporate the following knowledge-based constraints into the structure learning process:

a) **Temporal order** constraints can be specified in the input file *constraintsTemporal.csv* which is placed in folder *Input*. The temporal constraints specify that a variable within a higher tier cannot serve as a parent of a variable within a lower tier, encoded as shown in the example of Fig 2.1. Note that not all tiers need to incorporate the same number of variables, and not all variables need to be assigned to a tier. The system will assume that variables not assigned to a particular tier are under no temporal restrictions.

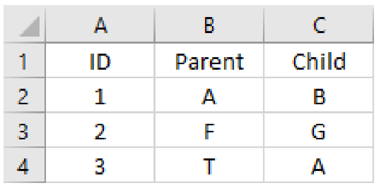
   Note that while temporal constraints are optional, the file *constraintsTemporal.csv* needs to exist in folder *Input* even if no restrictions are specified within the file; i.e., as provided when you first download Bayesys. Furthermore, the subtab *Knowledge-based constraints*, under tab *Learning*, gives the option to the user to specify whether the algorithm should further prohibit edges between variables of the same tier.



| A | B | C | D | E | F |
|---|---|---|---|---|---|
| ID | Tier 1 | Tier 2 | Tier 3 | Tier 4 | END |
| 1 | VarName1 | VarName4 | VarName5 | VarName7 | |
| 2 | VarName2 | | VarName6 | VarName8 | |
| 3 | VarName3 | | | VarName9 | |
| 4 | | | | VarName10 | |

**Fig 2.1.** Hypothetical temporal constraints in input file *constraintsTemporal.csv*. These restrictions specify that a variable within a higher tier cannot serve as a parent of a variable within a lower tier. The column ID (number of max constraints over all variables) and column END which is placed at the end of the final tier, must be present in this input file as shown in the figure.

b) **Directed arc** constraints can be specified in the input file *constraintsDirected.csv* which is placed in folder *Input*. The directed constraints specify which arcs must be preserved within the graph under discovery. They can be encoded as shown in the example of Fig 2.2. While directed restrictions are also optional, the input file *constraintsDirected.csv* needs to be present in folder *Input* even if no restrictions are specified; i.e., as provided when you first download Bayesys.



| | A | B | C |
|---|---|---|---|
| 1 | ID | Parent | Child |
| 2 | 1 | A | B |
| 3 | 2 | F | G |
| 4 | 3 | T | A |

**Fig 2.2.** Hypothetical directed constraints in input file *constraintsDirected.csv*. These restrictions specify that there must be a directed arc from each parent to each child. The column ID must be present in this input file as shown in the figure.

## *2.2. Outputs generated*

The structure learning process generates a number of outputs, some of which are optional. These are:

a) **Arcs**: the file *DAGlearned.csv* in folder *Output*. This file simply captures the arcs discovered.

b) **Graphs**: Three PDF files that capture the learning progress of SaiyanH by means of graphical structure, from phase 1 to phase 3. The phase 3 graph represents the final learned graph; i.e., the graph that associates with *DAGlearned.csv*.

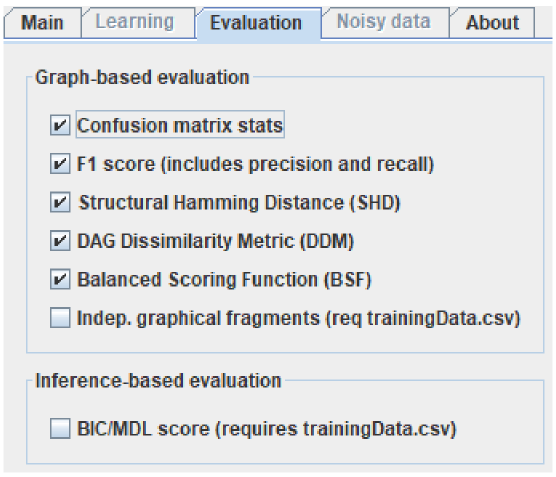
This output is optional. To generate the graphs, you must select *Save graphs* under *Structure learning* in tab *Main*. The PDF files are generated in directory *Output/SaiyanH*.

c) **Associational scores**: Four CSV files that capture the marginal associational scores generated during phase 1 of SaiyanH (one file), and the conditional associational scores generated during phase 2 of SaiyanH (three files). The three files that correspond to phase 2 represent scores classified as conditional dependency, conditional independency, and conditional insignificance.

This output is optional. To generate the CSV files, select *Save associational scores* under *Structure learning* in tab *Main*. The CSV files are generated in directory *Output/ SaiyanH*.

# 3. Evaluate a learned graph

Select *Evaluate graph* in tab *Main* to evaluate a learned graph with respect to the ground truth graph. The evaluation process is separated into a graph-based and an inference-based evaluation, as shown in Fig 3.1. Moreover, if you select *Generate DAGlearned.PDF* under *Evaluate graph* in tab *Main*, this will generate the corresponding graph of *DAGlearned.csv* in a PDF file.



Graph-based evaluation

- [✓] Confusion matrix stats
- [✓] F1 score (includes precision and recall)
- [✓] Structural Hamming Distance (SHD)
- [✓] DAG Dissimilarity Metric (DDM)
- [✓] Balanced Scoring Function (BSF)
- [ ] Indep. graphical fragments (req trainingData.csv)

Inference-based evaluation

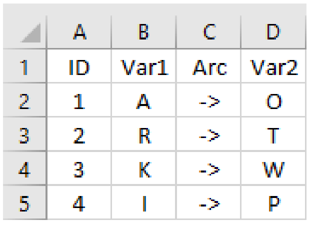
- [ ] BIC/MDL score (requires trainingData.csv)

**Fig 3.1.** The graph-based and inference-based evaluations of a learned graph based on Bayesys v1.63.

The first five graph-based evaluators (those ticked in Fig 3.1) require two input files:

a) the *DAGtrue.csv* in folder *Input*, and
b) the *DAGlearned.csv* in folder *Output*,

encoded as shown in Fig 3.2. If you are running *Evaluate graph* together with *Structure learning*, then the *DAGlearned.csv* will be generated automatically in folder *Output* at the end of the structure learning process, and before the system runs the evaluation process. If you are running *Evaluate graph* without *Structure learning* (e.g., evaluating a previously learned graph or a learned graph produced by some other algorithm), then you need to place both *DAGtrue.csv* and *DAGlearned.csv* files in the *Input* and *Output* folders respectively.



|   | A | B | C | D |
|---|---|---|---|---|
| 1 | ID | Var1 | Arc | Var2 |
| 2 | 1 | A | -> | O |
| 3 | 2 | R | -> | T |
| 4 | 3 | K | -> | W |
| 5 | 4 | I | -> | P |

**Fig 3.2.** An example of the *DAGtrue/DAGlearned.csv* input/output file.

These five graph-based evaluators produce scores that approximate the relevance of the learned graph with respect to the ground truth graph. The implementation assumes that DAGtrue is a DAG or a Mixed Ancestral Graph (MAG). While SaiyanH always produces a DAG in *DAGlearned.csv*, the system allows for other types of edges to be included in *DAGlearned.csv* as those shown in Table 3.1. This is useful if you are looking to evaluate a graph generated by some other algorithm, with the scoring criteria listed in Table 3.1. The material that follows is comes from [4].

**Table 3.1.** The penalty weights used by the scoring metrics implemented in Bayesys [4].

| Rule | True graph | Learned graph | Penalty | Reasoning |
|---|---|---|---|---|
| 1 | $A \rightarrow B$ | $A \rightarrow B$, A o→ B | 0 | Complete match |
| 2 | $A \rightarrow B$ | $A \leftrightarrow B$, $A - B$, $A \leftarrow B$, A←o B | 0.5 | Partial match |
| 3 | $A \rightarrow B$ | $A \perp B$ | 1 | No match |
| 4 | $A \leftrightarrow B$ | Any edge/arc | 0 | Latent confounder |
| 5 | $A \perp B$ | $A \perp B$ | 0 | Complete match |
| 6 | $A \perp B$ | Any edge/arc | 1 | Incorrect dependency discovered |

The five scoring metrics are:

a) ***Confusion matrix stats***: produces the following scores:

- *True Positives* ($TP$): the number of correct edges discovered in the learned graph.
- *Half True Positives* ($TP \times 0.5$): the number of partially correct edges discovered in the learned graph; e.g., ←, −, or ↔ instead of →.
- *False Positives* ($FP$): the number of incorrect arcs or edges discovered in the learned graph.
- *True Negatives* ($TN$): the number of correct direct independencies discovered in the learned graph.
- *False negatives* ($FN$): the number of incorrect direct independencies discovered in the learned graph.

b) ***Precision***: defined as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$, and represents the rate of correct direct dependencies from those discovered.

c) ***Recall***: defined as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$, and represents the rate of direct dependencies discovered from those in the true graph.

d) ***F1 Score:*** which represents the harmonic mean of Recall and Precision, defined as:

$$\mathrm{F1} = 2\frac{rp}{r+p}$$

where $r$ is Recall and $p$ is Precision. The F1 score ranges from 0 to 1, where 1 represents the highest score (with perfect Precision and Recall) and 0 the lowest.

e) ***Structural Hamming Distance (SHD)***: defined as the minimum number of edge insertions, deletions, and arc reversals that are needed in order to transform the learned graph into the true graph. To be consistent with Table 3.1, the number of insertions and deletions generate a penalty of 1, whereas arc reversals generate a penalty of 0.5.

f) ***DAG Dissimilarity Metric (DDM)***: a score that ranges from $-\infty$ to 1, where a score of 1 indicates perfect agreement between the two graphs. The score moves to $-\infty$ the stronger the dissimilarity is between the two graphs. Specifically, if we compare how dissimilar the graph $A$ is with respect to graph $B$, then

$$\mathrm{DDM} = \frac{\mathrm{TP} + \frac{\mathrm{r}}{2} - FN - FP}{\mathrm{t}}$$

where $r$ is the number of arcs from $B$ reoriented in $A$, and $t$ is total number of arcs in $B$.

i. ***Balanced Scoring Function (BSF)***: This is a balanced score that takes into consideration all the confusion matrix parameters (i.e., TP, TN, FP, and FN) to balance the score between direct independencies and direct dependencies [4]. Specifically,

$$\mathrm{BSF} = 0.5\left(\frac{\mathrm{TP}}{a} + \frac{\mathrm{TN}}{i} - \frac{\mathrm{FP}}{i} - \frac{\mathrm{FN}}{a}\right)$$

where $a$ is the numbers of edges and $i$ is the number of direct independences in the true graph, and

$$i = \frac{n(n-1)}{2} - a$$

where $n$ is the number of variables in the data. The BSF score ranges from -1 to 1, where -1 corresponds to the worst possible graph, 1 to the graph that matches the true graph, whereas a score of 0 corresponds to an empty or a fully connected graph.

Further to the above five scoring metrics, the feature ***Indep. graphical fragments*** returns the number of independent graphical fragments (also known as disjoint subgraphs) found in *DAGlearned.csv* with respect to all of the variables in *trainingData.csv*. Note that this process requires that *trainingData.csv* is in folder *Input*, in addition to the *DAGlearned.csv* and *DAGtrue.csv* files discussed above.

Finally, selecting *BIC score* will produce the inference-based score Bayesian Information Criterion (BIC). Note that this process also requires the *trainingData.csv* file in folder *Input.* The BIC score in Bayesys represents one[2] of the standard forms of BIC, and is computed as follows:

$$BIC = LL(G|D) - \left(\frac{log_2 N}{2}\right)F$$

for graph $G$ given data $D$, where $LL$ is the log-likelihood, $N$ is the sample size of $D$, and $F$ is the number of free parameters (also known as independent parameters) in $G$. Assuming $V$ represents the set of the variables $v_i$ in graph $G$, and $|V|$ is the size of set $V$, the number of free parameters $F$ is:

$$F = \sum_{i}^{|V|}(r_i - 1)\prod_{j}^{|\pi_{v_i}|} q_j$$

where $r_i$ is the number of states of $v_i$, $\pi_{v_i}$ is the parent set of $v_i$, $|\pi_{v_i}|$ is the size of set $\pi_{v_i}$, and $q_j$ is the number of states of $v_j$ in parent set $\pi_{v_i}$.

[2] As of Bayesys v1.7, the UI gives the option to the user to change the log scale in the BIC equation.

## 4. Convert DAG into a BN model in AgenaRisk

Select *Generate BN model* to convert a learned DAG into a BN model in the AgenaRisk BN modelling software (based on AgenaRisk's SDK 6120). This process requires the input files *trainingData.csv* in folder *Input* and *DAGlearned.csv* in folder *Output*. The output of this process is *bnLearned.cmp* in folder *Output,* and which can be loaded in AgenaRisk.

Note that running the BN model in AgenaRisk requires a license. Special licenses will be issued in due course.

## 5. Worked 3-step example: *Structure Learning, Evaluation, and BN model.*

The example illustrated in this section is based on the ASIA data example which you can find in folder *ASIA example* placed in the main directory of the Netbeans project. These files are taken from the Bayesys data repository [5], which includes more case studies. To reproduce the example, rename the appropriate ASIA files into *DAGtrue.csv* and *trainingData.csv* and place them in folder *Input*.

### *5.1. Structure learning*

Run Bayesys and select *Structure learning*, along with *Save graphs* and *Save associational* scores. Hit run in tab *Main*. The following files are almost instantly produced in directory *Output/SaiyanH*:

a) three PDFs that correspond to the graphs shown in Fig 5.1 (i.e., the three phases in SaiyanH),
b) four CSV files that represent the marginal and conditional dependence scores from phases 1 and 2, as shown in Fig 5.2.

Fig 5.3 presents relevant information generated in the terminal window of Netbeans, during this process.

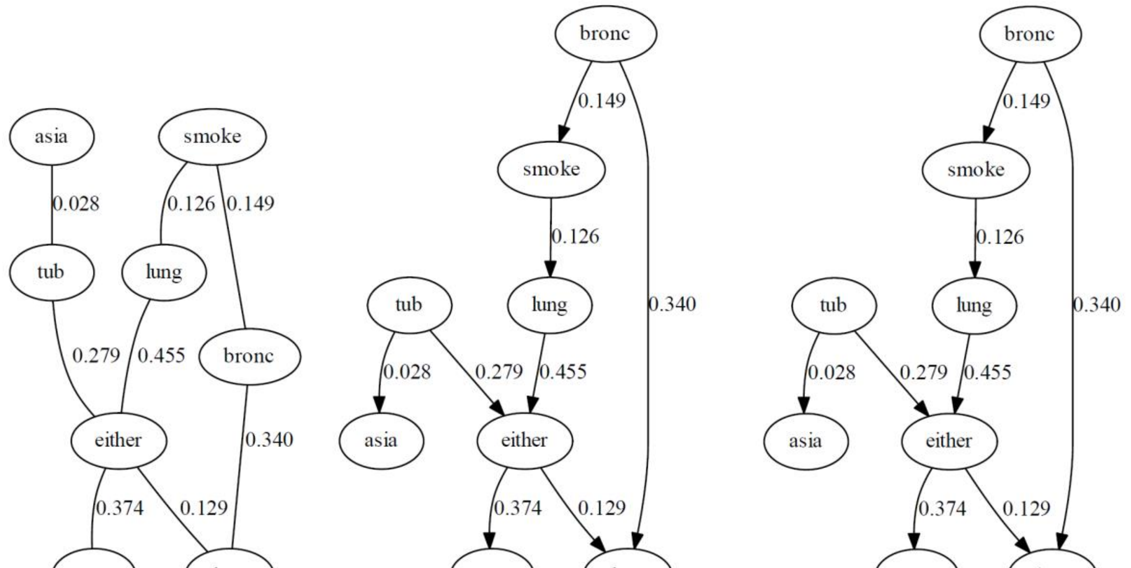
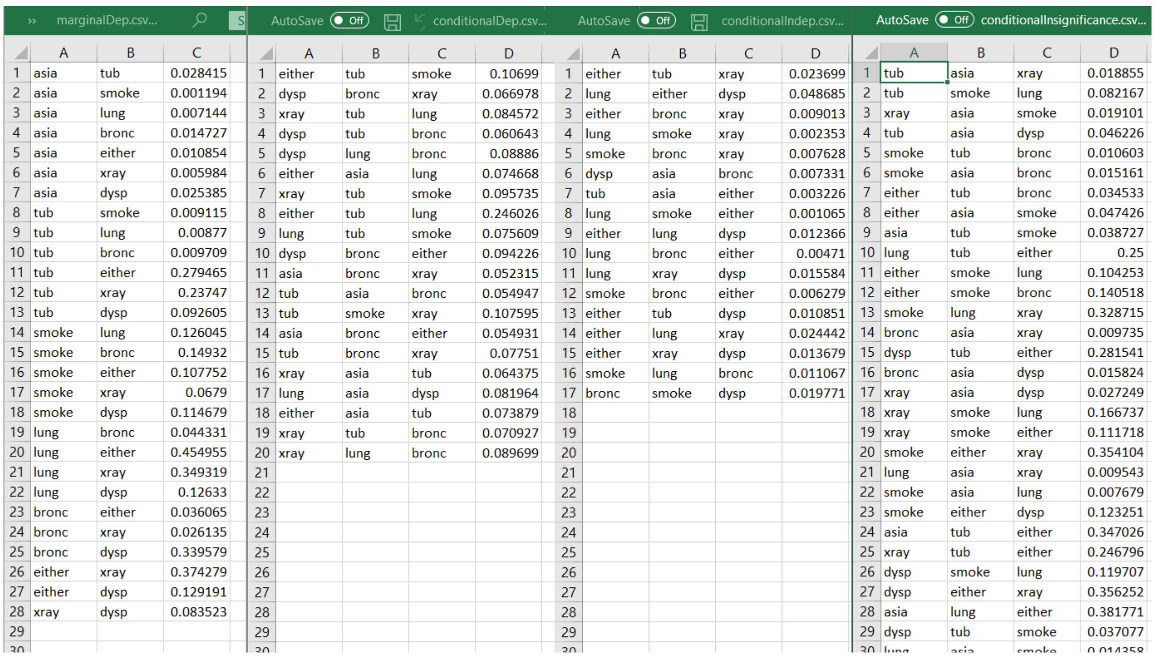


**Fig 5.1.** The three PDF graphs generated after selecting *Save graphs*. They correspond to the three phases of SaiyanH. The outputs are based on the ASIA 10k sample-size example. In this example, outputs of phases 2 and 3 are identical because score-based learning did not discover a graph with a higher score.



marginalDep.csv...

| | A | B | C |
|---|---|---|---|
| 1 | asia | tub | 0.028415 |
| 2 | asia | smoke | 0.001194 |
| 3 | asia | lung | 0.007144 |
| 4 | asia | bronc | 0.014727 |
| 5 | asia | either | 0.010854 |
| 6 | asia | xray | 0.005984 |
| 7 | asia | dysp | 0.025385 |
| 8 | tub | smoke | 0.009115 |
| 9 | tub | lung | 0.00877 |
| 10 | tub | bronc | 0.009709 |
| 11 | tub | either | 0.279465 |
| 12 | tub | xray | 0.23747 |
| 13 | tub | dysp | 0.092605 |
| 14 | smoke | lung | 0.126045 |
| 15 | smoke | bronc | 0.14932 |
| 16 | smoke | either | 0.107752 |
| 17 | smoke | xray | 0.0679 |
| 18 | smoke | dysp | 0.114679 |
| 19 | lung | bronc | 0.044331 |
| 20 | lung | either | 0.454955 |
| 21 | lung | xray | 0.349319 |
| 22 | lung | dysp | 0.12633 |
| 23 | bronc | either | 0.036065 |
| 24 | bronc | xray | 0.026135 |
| 25 | bronc | dysp | 0.339579 |
| 26 | either | xray | 0.374279 |
| 27 | either | dysp | 0.129191 |
| 28 | xray | dysp | 0.083523 |
| 29 | | | |

AutoSave Off conditionalDep.csv...

| | A | B | C | D |
|---|---|---|---|---|
| 1 | either | tub | smoke | 0.10699 |
| 2 | dysp | bronc | xray | 0.066978 |
| 3 | xray | tub | lung | 0.084572 |
| 4 | dysp | tub | bronc | 0.060643 |
| 5 | dysp | lung | bronc | 0.08886 |
| 6 | either | asia | lung | 0.074668 |
| 7 | xray | tub | smoke | 0.095735 |
| 8 | either | tub | lung | 0.246026 |
| 9 | lung | tub | smoke | 0.075609 |
| 10 | dysp | bronc | either | 0.094226 |
| 11 | asia | bronc | xray | 0.052315 |
| 12 | tub | asia | bronc | 0.054947 |
| 13 | tub | smoke | xray | 0.107595 |
| 14 | asia | bronc | either | 0.054931 |
| 15 | tub | bronc | xray | 0.07751 |
| 16 | xray | asia | tub | 0.064375 |
| 17 | lung | asia | dysp | 0.081964 |
| 18 | either | asia | tub | 0.073879 |
| 19 | xray | tub | bronc | 0.070927 |
| 20 | xray | lung | bronc | 0.089699 |

AutoSave Off conditionalIndep.csv...

| | A | B | C | D |
|---|---|---|---|---|
| 1 | either | tub | xray | 0.023699 |
| 2 | lung | either | dysp | 0.048685 |
| 3 | either | bronc | xray | 0.009013 |
| 4 | lung | smoke | xray | 0.002353 |
| 5 | smoke | bronc | xray | 0.007628 |
| 6 | dysp | asia | bronc | 0.007331 |
| 7 | tub | asia | either | 0.003226 |
| 8 | lung | smoke | either | 0.001065 |
| 9 | either | lung | dysp | 0.012366 |
| 10 | lung | bronc | either | 0.00471 |
| 11 | lung | xray | dysp | 0.015584 |
| 12 | smoke | bronc | either | 0.006279 |
| 13 | either | tub | dysp | 0.010851 |
| 14 | either | lung | xray | 0.024442 |
| 15 | either | xray | dysp | 0.013679 |
| 16 | smoke | lung | bronc | 0.011067 |
| 17 | bronc | smoke | dysp | 0.019771 |

AutoSave Off conditionalInsignificance.csv...

| | A | B | C | D |
|---|---|---|---|---|
| 1 | tub | asia | xray | 0.018855 |
| 2 | tub | smoke | lung | 0.082167 |
| 3 | xray | asia | smoke | 0.019101 |
| 4 | tub | asia | dysp | 0.046226 |
| 5 | smoke | tub | bronc | 0.010603 |
| 6 | smoke | asia | bronc | 0.015161 |
| 7 | either | tub | bronc | 0.034533 |
| 8 | either | asia | smoke | 0.047426 |
| 9 | asia | tub | smoke | 0.038727 |
| 10 | lung | tub | either | 0.25 |
| 11 | either | smoke | lung | 0.104253 |
| 12 | either | smoke | bronc | 0.140518 |
| 13 | smoke | lung | xray | 0.328715 |
| 14 | bronc | asia | xray | 0.009735 |
| 15 | dysp | tub | either | 0.281541 |
| 16 | bronc | asia | dysp | 0.015824 |
| 17 | xray | asia | dysp | 0.027249 |
| 18 | xray | smoke | lung | 0.166737 |
| 19 | xray | smoke | either | 0.111718 |
| 20 | smoke | either | xray | 0.354104 |
| 21 | lung | asia | xray | 0.009543 |
| 22 | smoke | asia | lung | 0.007679 |
| 23 | smoke | either | dysp | 0.123251 |
| 24 | asia | tub | either | 0.347026 |
| 25 | xray | tub | either | 0.246796 |
| 26 | dysp | smoke | lung | 0.119707 |
| 27 | dysp | either | xray | 0.356252 |
| 28 | asia | lung | either | 0.381771 |
| 29 | dysp | tub | smoke | 0.037077 |

**Fig 5.2.** The four CSV files generated after selecting *Generate DAGlearned.PDF*. They capture the marginal (left sheet) and conditional dependence associational scores. Note that, in the three conditional score files, column *A* represents the conditional variable. The outputs are based on the ASIA 10k sample-size example.

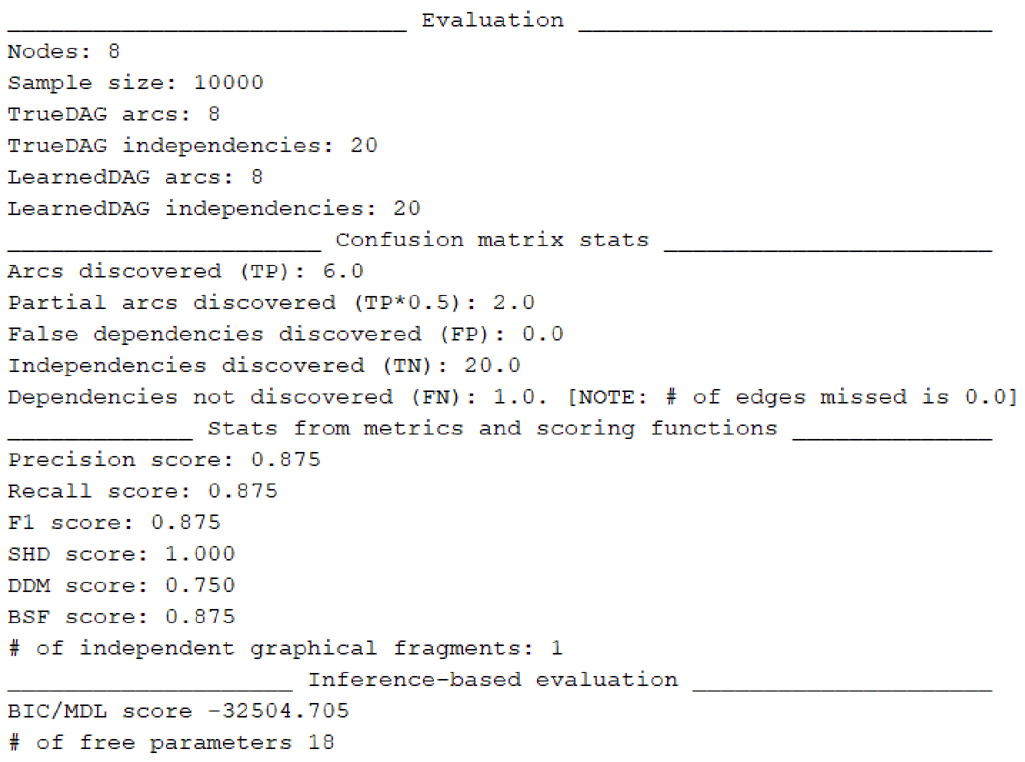
```
________________________ Training data info ________________________
Variables: 8
Sample size: 10000
___________________ Knowledge-based constraints ____________________
Temporal constraints specified: 0
Directed constraints specified: 0
_______________________ Structure learning _________________________
Running SaiyanH with settings:
    a) Associational score: MeanMax [Absolute]
    b) Conditional independence pruning: true
    c) Faithfulness condition pruning: true
    d) TABU search max escape attempts: V(V-1)
Entering Phase 1 [EMST graph]...
Phase 1 completed.
Entering Phase 2 [constraint-based learning]...
Phase 2 completed.
Entering Phase 3 [score-based learning]...
Phase 3 completed.
Arcs randomised during phase 2 constraint-based learning: 0
Structure learning elapsed time: 0 seconds total (Phase 1 = 0 secs, Phase 2 = 0 secs).
```

**Fig 5.3.** The output information generated in the terminal window of Netbeans after performing structure learning on the ASIA 10k sample-size example.

## 5.2. *Evaluate graph*

Following the process described in subsection 5.1, select *Evaluate graph* in tab *Main*. In tab *Evaluation*, select all of the scoring metrics. Hit run in tab *Main*. The process will instantly produce the information shown in Fig 5.4 in the terminal window of Netbeans. Note that the *# of independent graphical fragments* will always be 1 for graphs generated by SaiyanH, since it learns graphs that enable full propagation of evidence by design [2].

The *Evaluate graph* process can be performed with or without *Structure learning*. If you are performing this process without first running the example described in subsection 5.1, you will need to place the *trainingData.csv*, *DAGtrue.csv*, and *DAGlearned.csv* in the appropriate directories as discussed in Section 3.



```
____________________________ Evaluation _____________________________
Nodes: 8
Sample size: 10000
TrueDAG arcs: 8
TrueDAG independencies: 20
LearnedDAG arcs: 8
LearnedDAG independencies: 20
______________________ Confusion matrix stats _______________________
Arcs discovered (TP): 6.0
Partial arcs discovered (TP*0.5): 2.0
False dependencies discovered (FP): 0.0
Independencies discovered (TN): 20.0
Dependencies not discovered (FN): 1.0. [NOTE: # of edges missed is 0.0]
_____________ Stats from metrics and scoring functions ______________
Precision score: 0.875
Recall score: 0.875
F1 score: 0.875
SHD score: 1.000
DDM score: 0.750
BSF score: 0.875
# of independent graphical fragments: 1
____________________ Inference-based evaluation _____________________
BIC/MDL score -32504.705
# of free parameters 18
```

**Fig 5.4.** The output information generated in the terminal window of Netbeans after running the *Evaluate graph* process, and with all of the scoring metrics selected in tab *Evaluation*. The output is based on the ASIA 10k sample-size example.

## 5.3. *Generate BN model*

Select *Generate BN model* in tab *Main* to generate the AgenaRisk BN model file *bnLearned.cmp* in folder *Output*. Note that this process can be performed with or without *Structure learning*, as long as *trainingData.csv* and *DAGlearned.csv* files exist in folders *Input* and *Output* respectively. The learned BN model can be loaded in AgenaRisk, as shown in Fig 5.5. Fig 5.6 presents the relevant output information generated in the terminal window of Netbeans during this process.

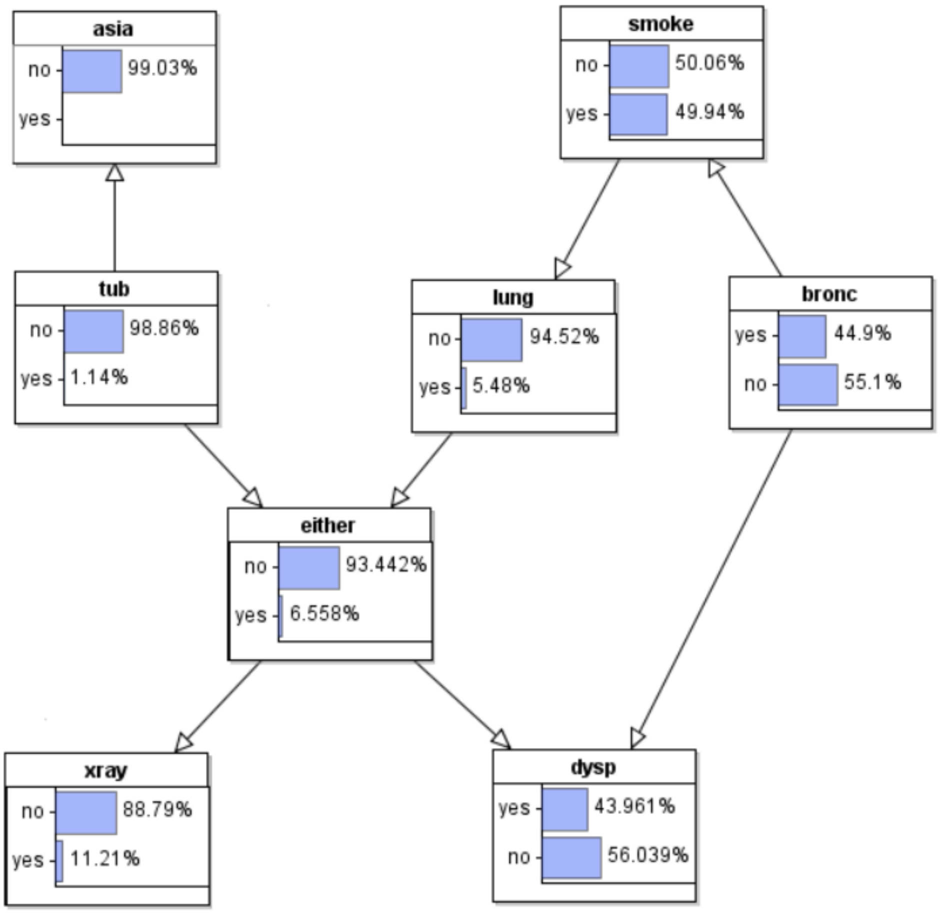
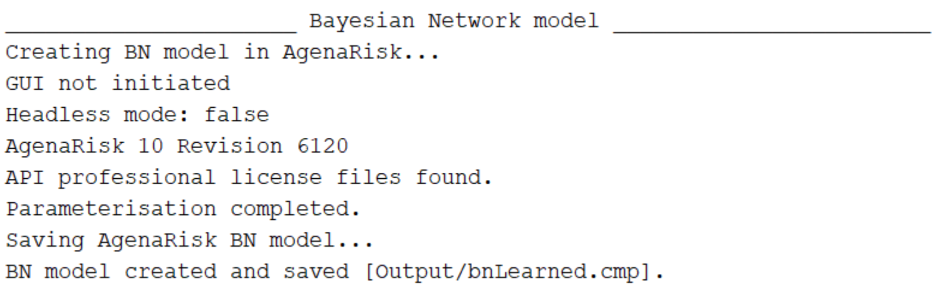


**Fig 5.5.** The BN model after loading *bnLearned.cmp* file in AgenaRisk, based on the ASIA 10k sample-size example.



```
_____________________ Bayesian Network model _______________________
Creating BN model in AgenaRisk...
GUI not initiated
Headless mode: false
AgenaRisk 10 Revision 6120
API professional license files found.
Parameterisation completed.
Saving AgenaRisk BN model...
BN model created and saved [Output/bnLearned.cmp].
```

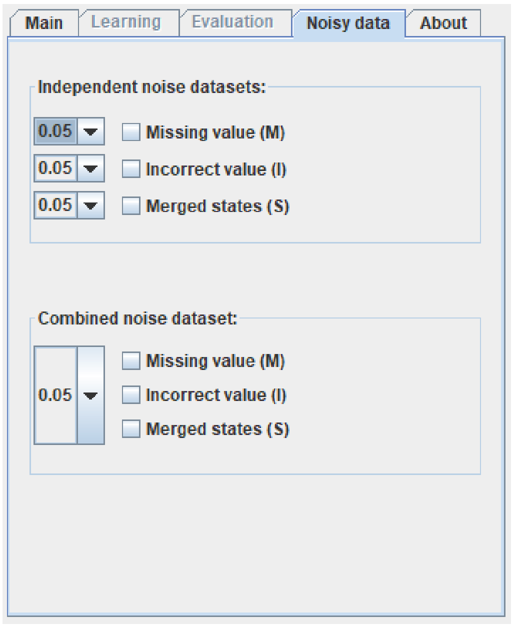
**Fig 5.6.** The output information generated in the terminal window of Netbeans after running the *Generate BN model* process in Bayesys.

# 6. Add noise to data

The process *Add noise to data* was implemented in Bayesys as part of the empirical evaluation assessments in [6]. This process is independent of other processes. Selecting *Add noise to data* will enable tab *Noisy data* as shown in Fig 6.1. This process requires *trainindData.csv* as input in folder *Input*, and produces the output files in directory *Output/Noisy data*.

There are three types of noise, and these can be simulated independently or in combinations of two. For example, selecting two types of noise under *Independent noise dataset* will produce two new datasets, and each dataset will correspond to each type of noise selected, whereas selecting two types of noise under *Combined noise dataset* will produce one dataset that incorporates both types of noise. To perform all three types of noise on the same data, this can be done by running the process twice. For example, you could add noise *S* and *I* on the first run, and rerun the process to add noise *M* on the output of the first run.

The filename of the output data file depends on the noisy case/s selected from those shown in Fig 6.1 and the numerical parameter which represents the rate of noise; e.g., parameter 0.05 implies each data point will have 5% chance of being manipulated. For example, selecting case *M* with parameter 0.05 will produce the filename *trainingData_M5*, whereas selecting the combined cases *M* and *I* will produce the filename *trainingData_MI5*. Fig 6.2 presents the output of the terminal window in Netbeans after selecting cases *M* and *I* under *Combined noise dataset*, with parameter set to 0.05, on the ASIA *trainingData_ASIA_10k.csv* sample dataset.



Independent noise datasets:

0.05 Missing value (M)
0.05 Incorrect value (I)
0.05 Merged states (S)

Combined noise dataset:

0.05 Missing value (M)
Incorrect value (I)
Merged states (S)

**Fig 6.1.** Features of tab *Noisy data*, which is enabled when selecting the process *Add noise to data*.

```
______________________ Noisy data generator info ____________
Entering combined noise randomisation [incorrect values].
Total data points randomised: 4073 out of 80000 [5.091%].
Entering combined noise randomisation [missing values].
Total data points randomised: 4142 out of 80000 [5.178%].
Generated trainingData_MI5.csv
```

**Fig 6.2.** The output in the terminal window after applying cases *M* and *I*, with parameters 0.05 for each case, to the ASIA *trainingData_ASIA_10k.csv* sample dataset.

# 7. Generate MAG

A MAG (i.e., Mixed Ancestral Graph) is used in BN problems that incorporate latent variables. The process *Generate MAG* was implemented in Bayesys as part of the empirical evaluation assessments in [6]. This process is independent of other processes.

*Generate MAG* requires three input files. These are a) *DAGtrue.csv*, b) *trainindData.csv*, and c) *trainingDataMAG.csv*, all in folder *Input*. The file *trainingDataMAG.csv* is simply a copy of *trainingData.csv* that incorporates all variables minus those which you would like to make latent. For example, if *trainingData.csv* includes variables $\{A, B, C, D, E\}$, and *trainingDataMAG.csv* includes variables $\{A, C, D, E\}$, then the system will produce the MAG of *DAGtrue.csv* in which variable $B$ is latent.

Running *Generate MAG* produces two output files in directory *Output/MAG*. These are *MAGtrue.csv* and *MAGtrue.pdf*. Both outputs capture the same information. Specifically, *MAGtrue.csv* captures the edges of MAG, revised from *DAGtrue.csv*, whereas *MAGtrue.pdf* represents the graph of *MAGtrue.csv*. Figures 7.1 and 7.2 present an example taken from [5] [6], where the ground truth DAG of the classic ALARM network is converted into the corresponding MAG that incorporates the two specified latent variables.

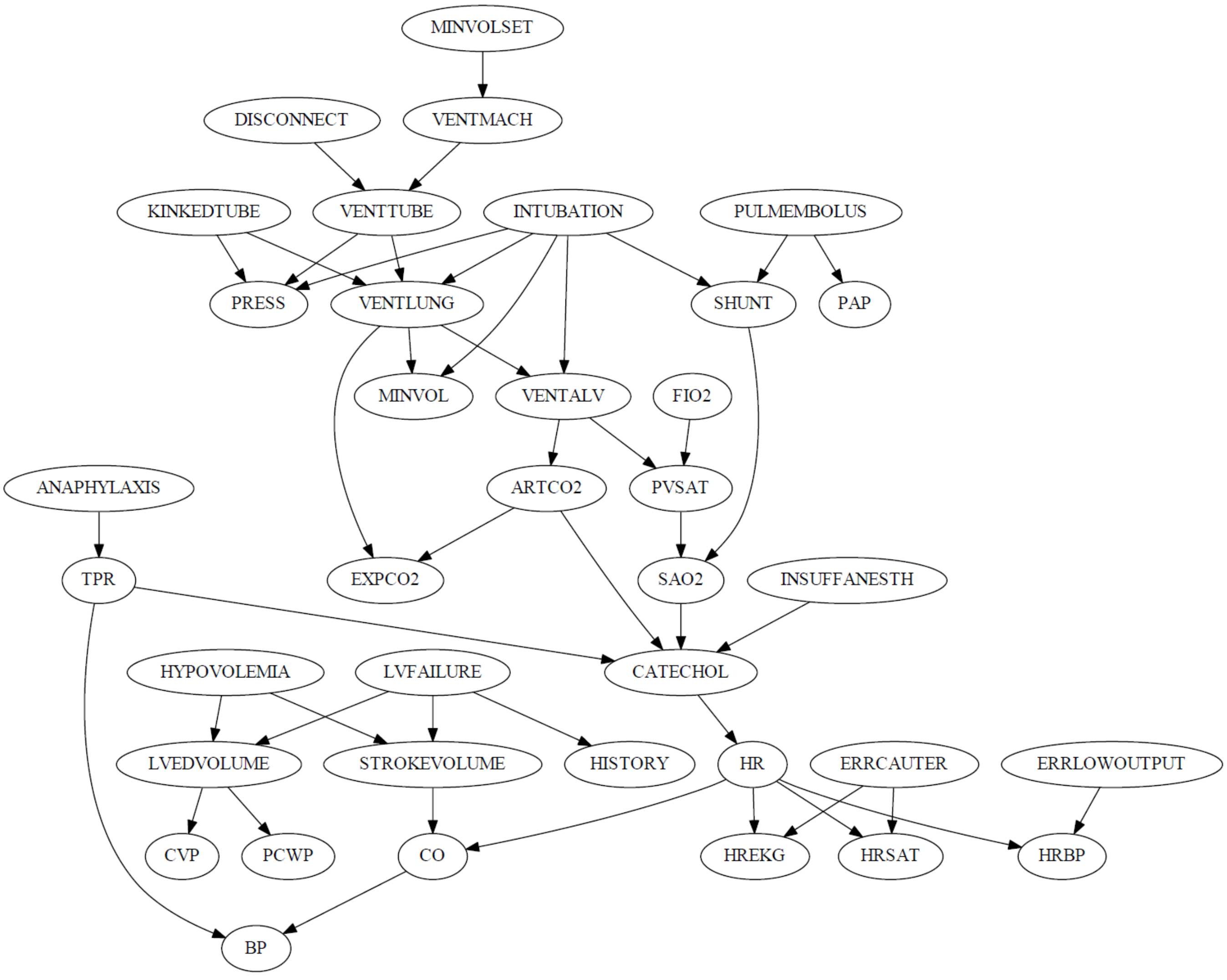


**Fig 7.1.** The ground truth DAG of the ALARM network case study as used in [5] [6]. Number of variables: 37. Number of arcs: 46.

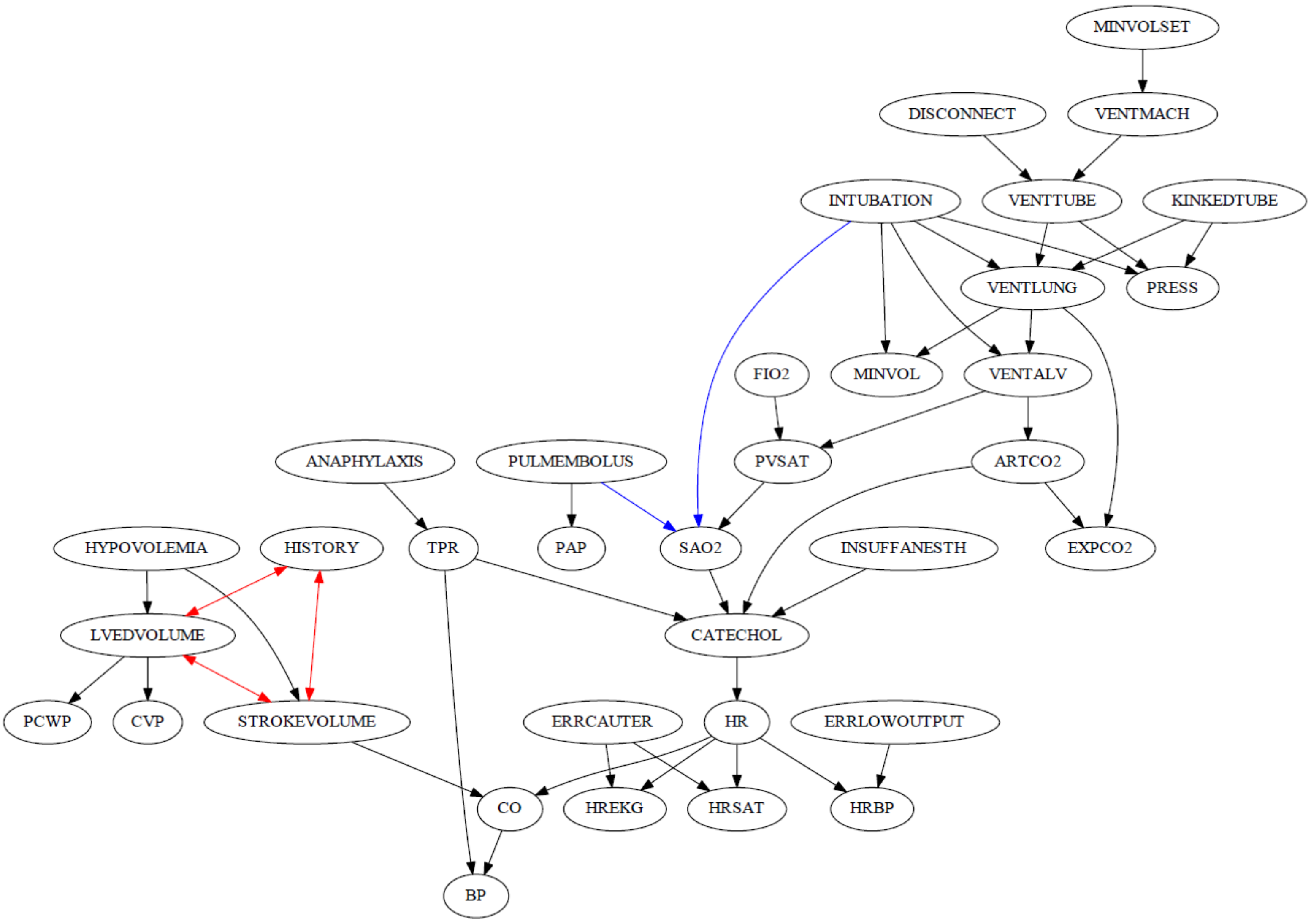


**Fig 7.2.** One of the ground truth MAGs of the ALARM network case study used in [5] [6]. Number of variables: 35. Number of edges: 46. Missing variables: LVFAILURE, SHUNT. Blue and red edges represent arcs and bi-directed edges in MAG, respectively, that are not present in the ground truth DAG.

# 8. Troubleshooting and things to know

## *8.1.* ***Input data files***

a) Variable and state names should not include a comma. This is because the data is read as Comma Separated Values (CSV) and the system automatically considers a comma as a data value separator.

b) Variable names should not start with a numeric value.

c) The naming of the input files should match those specified in this document; e.g., *trainingData.csv*.

d) Any variable names specified in temporal or directed knowledge-based constraints, such as in *constraintsTemporal.csv*, should match (case sensitive) the variable names in *trainingData.csv*.

## *8.2.* ***While running the system***

e) Before you run Bayesys, make sure that you close ALL of the input and output files. If an output file remains open while running Bayesys, the system may still complete a process without an error, but also without replacing any files that were running in the background.

## *8.3.* ***Netbeans terminal output errors***

f) ERROR: "*I/O error while writing the dot source to temp file!*". This error associates with the PDF graphs generated when calling the GraphViz software. This error may occur when a PDF file being generated by Bayesys is already in use by the OS. However, you may sometimes get this error for no apparent reason (problems with cache). When this happens, check that the output PDF files have been correctly revised by Bayesys (e.g., by checking time modified in file). Otherwise, rerun the process and the error should not appear during the second run.

## *8.4.* ***Computational time***

g) Large graphs with hundreds of variables increase computational time faster than linear. The computational time of SaiyanH is also heavily dependent on the sample size of the input data. Keep in mind that generating graphs in PDF that contain hundreds of variables may significantly contribute to the overall computational time.

# Appendix: Revision notes

## v1.5 Main revision notes:

1. The way *Precision* score is computed has been revised (this also influences the *F1* score).

2. The *Evaluate graph* process now gives the option to the user to generate *DAGlearned.pdf* which represents the graph of *DAGlearned.csv*.

3. A new main process has been added, called *Add noise to data* (refer to Section 6).

4. A new main process has been added, called *Generate MAG* (refer to Section 7).

5. SaiyanH will infrequently fail to orient all edges during phase 2. When this happens, the orientation of the edges will be randomised. The terminal window now outputs this information.

## v1.6 Main revision notes:

1. *DAGlearned.pdf* generator, under process *Evaluate graph*, is now optional and now also supports edges o→ and o−o.

2. The number of independent graphical fragments is now separated as an independent evaluation process, under tab *Evaluation*, that requires *trainingData.csv* to run.

## v1.7 Main revision notes:

1. Improvements have been made to the code that reads data from CSV files. The following data corruption issues have now been resolved:

   a. Errors caused by datasets with excess commas, often caused due to heavy file editing.
   b. Errors caused by datasets that incorporate hidden values. These values were neither visible in CSV form, nor in TXT form, nor readable in String code JAVA format. This type of data corruption is often caused by the formatting style of the source from which data are copied into a file.

2. The system will now end the structure leaning process (and inform the user in the terminal window) if the input dataset is found to incorporate missing data values (i.e., empty data cells).

3. The system will now end the structure leaning process (and inform the user in the terminal window) if the name of a variable starts with a numeric character.

4. The system will now warn the user when the name of a variable in *DAGlearned.csv* does not match any of the node names in *DAGtrue.csv*. The warning will be generated in the terminal window in red font colour.

5. The system will now warn the user when a variable has more than 20 states. The warning will be generated in the terminal window in red font colour.

6. The BIC score reported in the terminal window, under *Evaluation*, is now based on the standard version of BIC and also indicates the log scale used in the BIC equation. The UI now enables the user to change the log scale used in the BIC equation.

7. The *Evaluation* now returns F1=0 when F1=NaN; i.e., in the case of $Recall = Precision = 0$.

8. The runtime information for SaiyanH now includes the runtime spent in each of the three learning phases.

9. The terminal output information on the number of nodes in DAGtrue has been corrected.

## References

[1] Constantinou, A. C. (2018). Bayesian Artificial Intelligence for Decision Making under Uncertainty. *Engineering and Physical Sciences Research Council (EPSRC)*, EP/S001646/1.

[2] Constantinou, A. C. (2020). Learning Bayesian networks that enable full propagation of evidence. arXiv:2004.04571 [cs.LG].

[3] Constantinou, A. C. (2020). Learning Bayesian networks with the Saiyan algorithm. *ACM Transactions on Knowledge Discovery from Data*.

[4] Constantinou, A. (2019). Evaluating structure learning algorithms with a balanced scoring function. arXiv 1905.12666 [cs.LG].

[5] Constantinou, A. C., Liu, Y., Chobtham, K., Guo, Z., and Kitson, N. K. (2020). The Bayesys data and Bayesian network repository. *Queen Mary University of London*, London, UK. [Online]. Available: http://bayesian-ai.eecs.qmul.ac.uk/bayesys/ and http://www.bayesys.com

[6] Constantinou, A. C., Liu, Y., Chobtham, K., Guo, Z., and Kitson, N. K. (2020). Large-scale empirical validation of Bayesian Network structure learning algorithms with noisy data. arXiv:2005.09020 [cs.LG]