

# The Bayesys user manual

Anthony C. Constantinou<sup>a, b</sup>

Version 3.6<sup>1</sup>  
(last revision: Jun 2024)

- a) [Bayesian AI](#) research lab,  
Machine Intelligence and Decision Systems ([MInDS](#)) 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](mailto:a.constantinou@qmul.ac.uk)

- b) The Alan Turing Institute, UK, British Library, London, UK, NW1 2DB.



**MInDS**

Machine Intelligence and Decision Systems  
Research Group



Bayesian Artificial Intelligence  
Research Lab



Queen Mary  
University of London

**The  
Alan Turing  
Institute**

**EPSRC**

Engineering and Physical Sciences  
Research Council



**BAYESFUSION,LLC**  
Data Analytics, Modeling, Decision Support

[www.bayesfusion.com](http://www.bayesfusion.com)

**agena.ai**

[www.agena.ai](http://www.agena.ai)

---

<sup>1</sup> **Citation:** Constantinou, A. (2019). The Bayesys user manual. Bayesian AI research lab, MInDS research group, Queen Mary University of London, London, UK. [Online]. Available: <http://bayesian-ai.eecs.qmul.ac.uk/bayesys/>

# Table of Contents

Copyright notice.....	3
Acknowledgements .....	4
List of Abbreviations .....	5
Introduction .....	6
<b>1. Getting started with the Java NetBeans project.....</b>	<b>7</b>
<b>2. Quick overview .....</b>	<b>10</b>
<b>3. Structure learning .....</b>	<b>11</b>
3.1. Algorithms .....	11
3.2. Structure learning with categorical data .....	14
<b>4. Structure learning with knowledge-based constraints .....</b>	<b>16</b>
<b>5. Evaluating graphical structures .....</b>	<b>22</b>
<b>6. Learning Bayesian network models .....</b>	<b>26</b>
6.1. Converting a learned graph into a Bayesian network model in GeNIe .....	26
6.2. Converting a learned graph into a Bayesian network model in AgenaRisk .....	26
<b>7. Worked example: Structure learning, evaluation, and parameterisation of a BN model... 27</b>	<b>27</b>
7.1. Structure learning .....	27
7.2. Evaluate graph .....	29
7.3. Generate BN model in GeNIe .....	32
7.3.1. Generate BDN model in GeNIe .....	33
7.4. Generate BN model in AgenaRisk.....	35
<b>8. Worked examples: Structure learning with knowledge-based constraints .....</b>	<b>37</b>
<b>9. Worked example: Performing multiple structure learning experiments .....</b>	<b>41</b>
<b>10. Worked example: Model-averaging .....</b>	<b>45</b>
<b>11. Generate synthetic data .....</b>	<b>48</b>
11.1. Generate clean data .....	48
11.2. Generate noisy data.....	48
<b>12. Generate MAG .....</b>	<b>50</b>
<b>13. Troubleshooting and things to know .....</b>	<b>52</b>
13.1. Input data files.....	52
13.2. While running the system.....	52
13.3. NetBeans terminal output errors .....	52
13.4. Warnings .....	52
13.5. Computational time .....	53
13.6. BN model generator (GeNIe) .....	53
<b>Appendix A: Revision notes .....</b>	<b>54</b>
<b>Appendix B: Stats for nerds .....</b>	<b>57</b>
<b>References.....</b>	<b>58</b>

## Copyright notice



Copyright © Bayesys.com. Bayesys is a free and an open-source software package. This manual is distributed under the terms of a CC BY-SA license: [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

THE BAYESYS SYSTEM IS DISTRIBUTED AND LICENSED FREE OF CHARGE IN THE HOPE IT WILL BE USEFUL. BECAUSE OF THIS, THERE IS NO WARRANTY OF ANY KIND FOR THE ACCURACY OR USEFULNESS OF THIS INFORMATION EXCEPT AS REQUIRED BY APPLICABLE LAW OR EXPRESSLY AGREED IN WRITING.

## Acknowledgements

- a) This project was 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.
- b) Supported by Agena Ltd (UK) who develop AgenaRisk, which is a Bayesian network software for risk analysis, AI and decision-making applications [2]. Bayesys enables users to convert learned graphs into parameterised Bayesian network models with appropriate file extensions that can be loaded into AgenaRisk [*Note: The AgenaRisk functions are no longer compatible with the new version of AgenaRisk which requires active license and internet connection*].
- c) Supported by BayesFusion LLC (USA) who develop GeNIe, which is a tool for AI and ML with Bayesian networks. Bayesys enables users to convert learned graphs into parameterised Bayesian network models, including Bayesian decision networks/Influence diagrams, with appropriate file extensions that can be loaded into GeNIe.
- d) Bayesys makes use of the open-source Graphviz visualisation system [3] that turns a textual representation of a graph into a visual drawing. The Graphviz Java API, written by Laszlo Szathmary, is incorporated into this Java project. Link to Laszlo’s Github page: <https://github.com/jabbalaci/graphviz-java-api>
- e) The following people have assisted by providing feedback, recommendations and improvements:

Dr Kiattikun Chobtham,  
Dr Yang Liu,  
Dr Neville Kenneth Kitson,  
Dr Zhigao Guo,  
Dr Marek J. Druzdzal,  
Mr Xiangyuan Tao.

## List of Abbreviations

<b>BDN</b>	Bayesian Decision Network.
<b>BIC</b>	Bayesian Information Criterion.
<b>BN</b>	Bayesian Network.
<b>BNSL</b>	Bayesian Network Structure Learning.
<b>BSF</b>	Balanced Scoring Function.
<b>CPT</b>	Conditional Probability Table.
<b>CPDAG</b>	Completed Partially Directed Acyclic Graph.
<b>CSV</b>	Comma Separated Values.
<b>DAG</b>	Directed Acyclic Graph.
<b>DDM</b>	DAG Dissimilarity Metric.
<b>FN</b>	False Negative.
<b>FP</b>	False Positive.
<b>GES</b>	Greedy Equivalence Search
<b>GUI</b>	Graphical User Interface.
<b>HC</b>	Hill-Climbing.
<b>IDE</b>	Interactive Development Environment.
<b>JDK</b>	Java Development Kit.
<b>LL</b>	Log Likelihood.
<b>MAG</b>	Maximal Ancestral Graph.
<b>MAHC</b>	Model-Averaging Hill-Climbing
<b>PAG</b>	Partial Ancestral Graph.
<b>SHD</b>	Structural Hamming Distance.
<b>TN</b>	True Negative.
<b>TP</b>	True Positive.
<b>UI</b>	User Interface.

## 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 Bayesys **version 3.6**. Frequent revisions will continue to be published online at [www.bayesys.com](http://www.bayesys.com) (or <http://bayesian-ai.eecs.qmul.ac.uk/bayesys/>). For an overview of any key revisions between released versions, refer to Appendix A.

# 1. Getting started with the Java NetBeans project

Bayesys is a Java application created using the NetBeans Interactive Development Environment (IDE), so it is necessary to have Java and NetBeans installed on your computer to run Bayesys.

What is described in this document applies to the Windows OS. While the project also runs on MAC and Linux OS, you may experience compatibility issues on MAC/Linux when using the functions that relate to the third-party Graphviz software that turns a textual representation of a graph into a visual drawing; i.e., the process may corrupt the PDF files that should contain drawings of the graphs. Please note that MAC/Linux users can still perform all other processes, including obtaining the learned graphs in a CSV format.

The latest release of Bayesys has been validated for versions JDK 17 and Apache NetBeans 21. To set up the Bayesys NetBeans project:

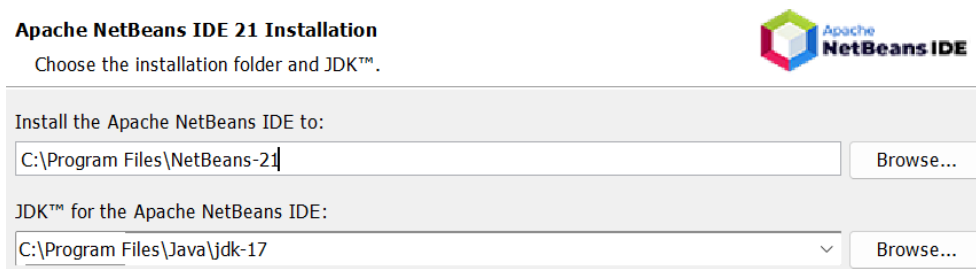
- a) Download and install the Java SE Development Kit version 17 (JDK – not the JRE), appropriate for your OS, from <https://www.oracle.com/java/technologies/downloads/> . Preference should be given to the 64-bit version.

Be careful not to download or use a JDK version older than 17, since older versions may require subscription fees – but JDK 17 or newer versions do not. If you have JDK 16 installed on your machine, you should uninstall it and install JDK 17.

- b) Download the latest NetBeans IDE directly from <https://netbeans.apache.org/download/index.html>. If you are given the option during the installation process, preference should be given to the 64-bit version.

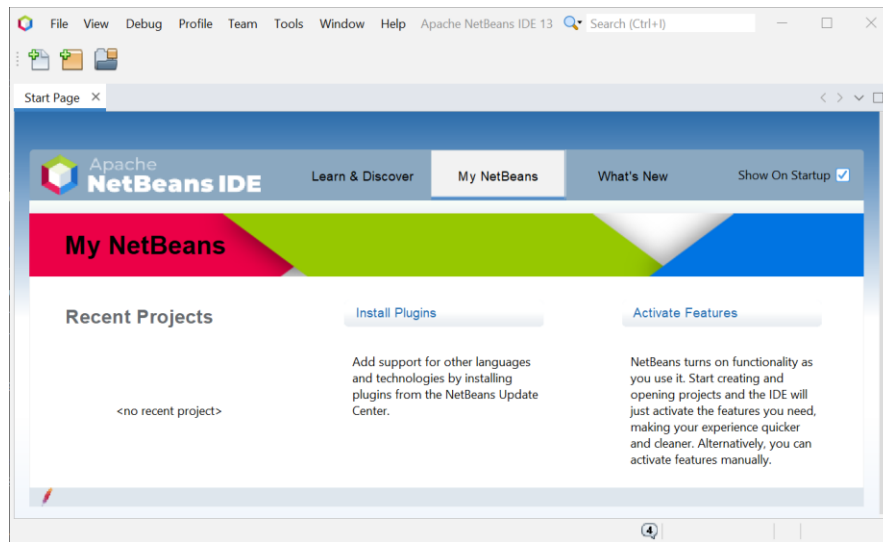
Once downloaded, install NetBeans IDE on your computer by clicking the downloaded installation file (e.g., v13 called Apache-NetBeans-13-bin-windows-x64.exe on Windows). You can generally just choose the default options to install NetBeans.

During the installation process, you will get the option to specify the directory of the JDK. Ensure that JDK v17 is selected, as shown in **Figure 1.1**. The option to specify the directory for JDK during the Apache Netbeans 21 installation process. below.




**Figure 1.1.** The option to specify the directory for JDK during the Apache Netbeans 21 installation process.

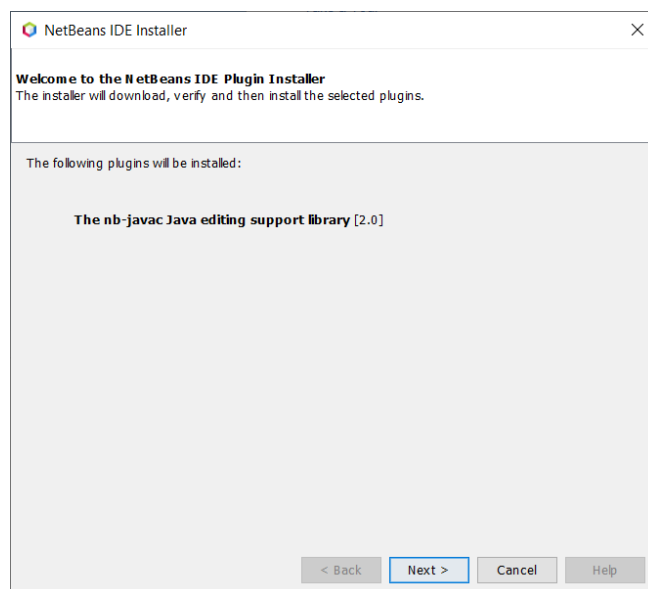
Once the NetBeans installation has completed, running NetBeans should open a window like:



**Figure 1.2.** The startup screen of NetBeans IDE 13.



- c) Download the Bayesys Java NetBeans project from [www.bayesys.com](http://www.bayesys.com). The project is compressed in a ZIP file extension. Extract the contents of the file to your preferred directory.
- d) In the Java NetBeans IDE, go to *File, Open Project*, and browse to the directory you have extracted the Bayesys project. You should see  **Bayesys v1**, or a more recent version, indicating that the folder is readable as a Java NetBeans project. Select the readable folder and click *Open Project* to load Bayesys in NetBeans.

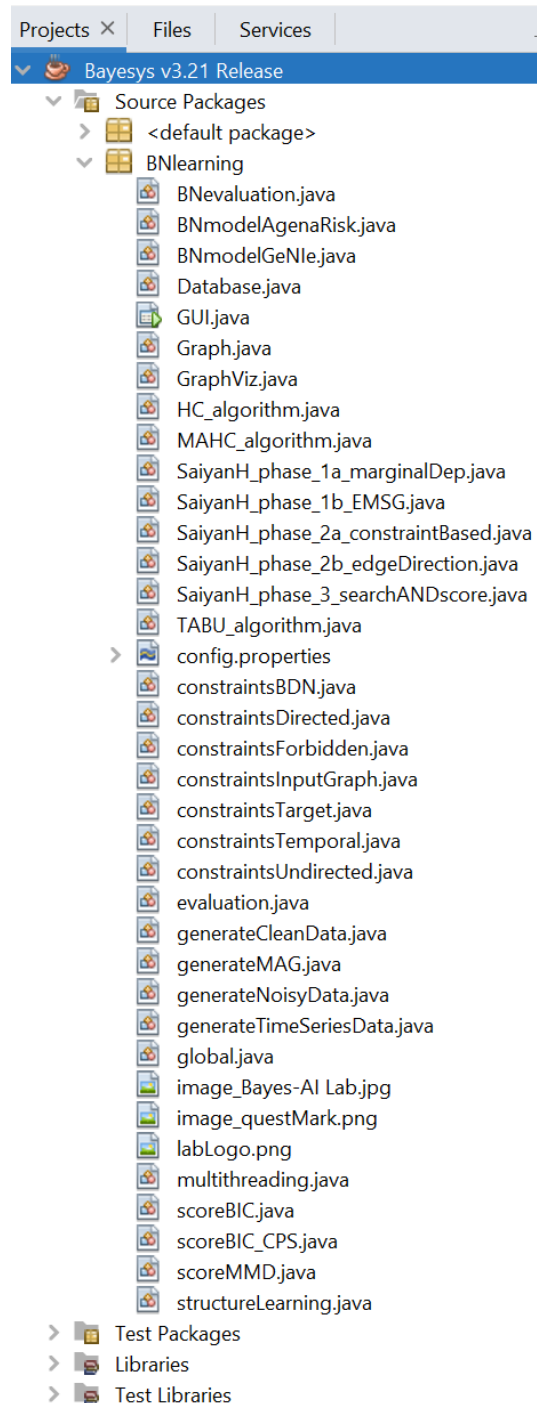
When loading the Bayesys project into NetBeans for the first time, you may be prompted to install a plugin, as shown in **Figure 1.3** below. You should complete the plugin installation.



**Figure 1.3.** Likely to be prompted to install a plugin once you load Bayesys v3+ into NetBeans.

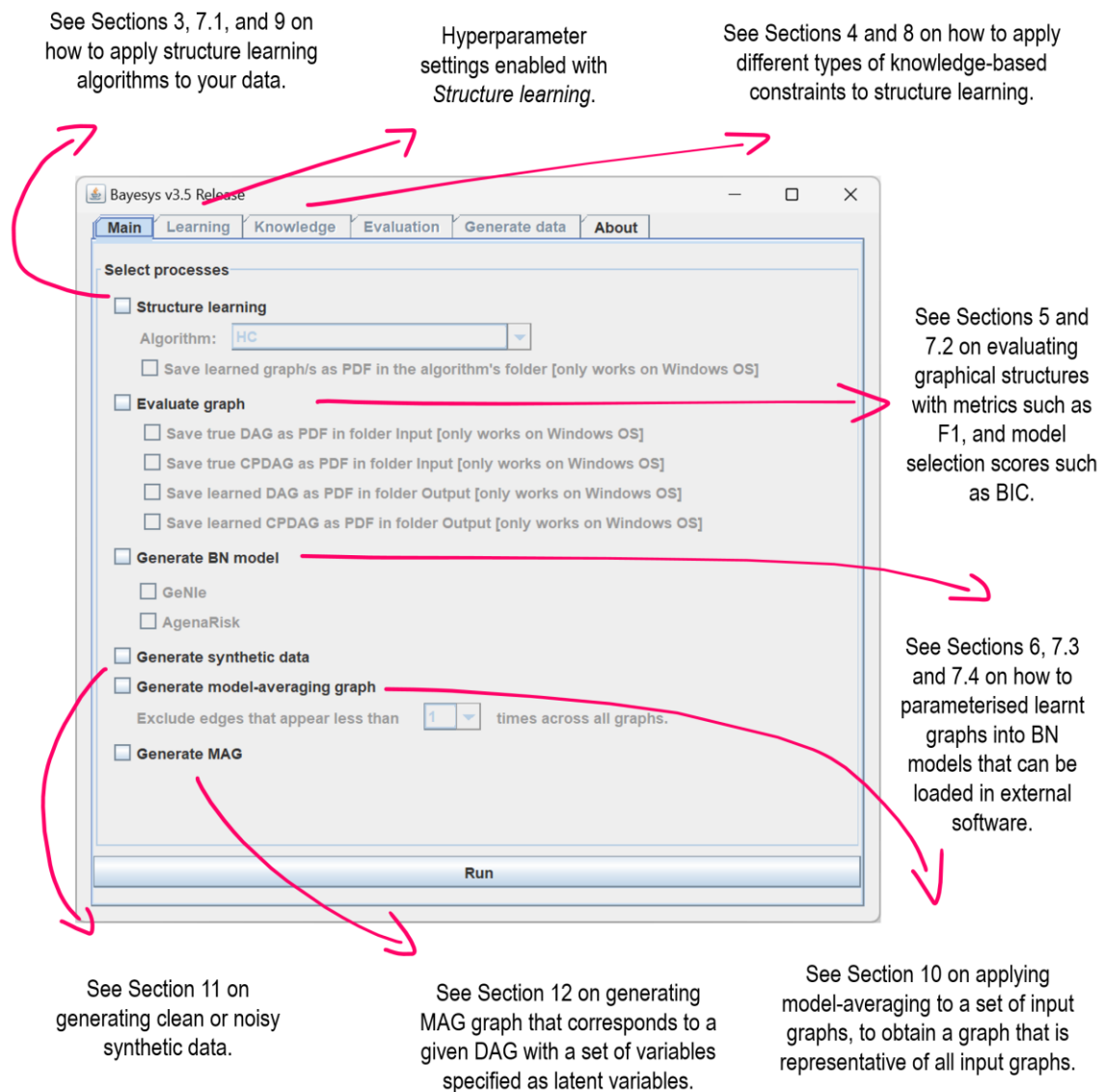


Once the project has been loaded into NetBeans, you should be able to view its contents as shown in **Figure 1.4**. If not, click on  to expand the ‘BNlearning’ directory. To run the project, click the run  button. If the run button is disabled, you can enable it by first clicking on the Java class *GUI.java* which you can find under the *BNlearning* directory shown in **Figure 1.4**.



**Figure 1.4.** The list of Java classes and other files associated with Bayesys v3.21.

## 2. Quick overview



**Figure 2.1.** Overview of the main learning methods available in Bayesys v3.55 implementation, with the corresponding Sections within this manual that cover them in detail.

### 3. Structure learning

#### 3.1. Algorithms

Five BN structure learning algorithms are currently available in Bayesys. These are:

- a) **SaiyanH** a hybrid structure learning algorithm described in [4]<sup>2</sup>. It starts with a dependency function that produces an undirected graph that can be viewed as an extended version of the maximum spanning graph, ensuring that each variable associates with at least one edge. It then obtains a DAG using a combination of constraint-based, score-based and interventional impact rules. The DAG is then provided as input to Tabu search, with the restriction not to delete edges that lead to disjoint subgraphs or disjoint nodes.

The SaiyanH version available in Bayesys v3.2+ includes revisions that have improved its accuracy slightly and its computational runtime considerably, in relation to the results presented in [4] and [5], and to some extent [6]. Refer to Appendix A for detailed revision notes.

- b) **MAHC** (Model-Averaging Hill-Climbing) is a score-based algorithm described in [7]. It combines two novel strategies with hill-climbing search. The algorithm starts by pruning the search space of graphs, where the pruning strategy can be viewed as an aggressive version of the pruning strategies that are typically applied to combinatorial optimisation structure learning problems. It then performs model averaging in the hill-climbing search process and moves to the neighbouring graph that maximises the objective function, on average, for that neighbour and over all its valid neighbours.
- c) **HC** represents the classic score-based hill-climbing search. The HC algorithm used in Bayesys is described in [6]. It starts from an empty graph and performs hill-climbing search by exploring arc additions, reversals and deletions, moving to the DAG that maximises the BIC score. The hyperparameter settings enable users to apply the pruning strategies of MAHC to HC.
- d) **TABU** represents the classic score-based Tabu search. The TABU algorithm implemented in Bayesys is described in [6]. When it reaches a local maximum, it performs  $|V|$  escapes (where  $V$  is the set of variables in the data) that minimally decrease the BIC score, and repeats hill-climbing on each iteration of  $|V|$  in an attempt to escape a local maximum. The hyperparameter settings enable users to apply the pruning strategies of MAHC to TABU.
- e) **GES** (Greedy Equivalence Search) is a greedy algorithm that explores the space of Markov equivalence classes [8]. It starts with an ‘insert’ phase that explores edge additions that increase the objective score the most, and then moves to the ‘delete’ phase in an analogous fashion until a final maximum scoring graph is produced.

Because Bayesys is designed to explore the DAG-space, rather than the CPDAG-space, the GES algorithm in Bayesys is applied to the DAG space. That is, at each iteration of arc addition, GES obtains the CPDAG of the current best DAG and applies greedy search to all possible orientations of undirected edges, one at a time, excluding orientations that lead to DAGs outside the equivalence class. This process

---

<sup>2</sup> SaiyanH is a revised version of Saiyan [16]. Only SaiyanH is available in Bayesys.

approximates the equivalence search-space in exchange for lower computational complexity; as opposed to applying greedy search to all possible Markov equivalence DAGs at each iteration.

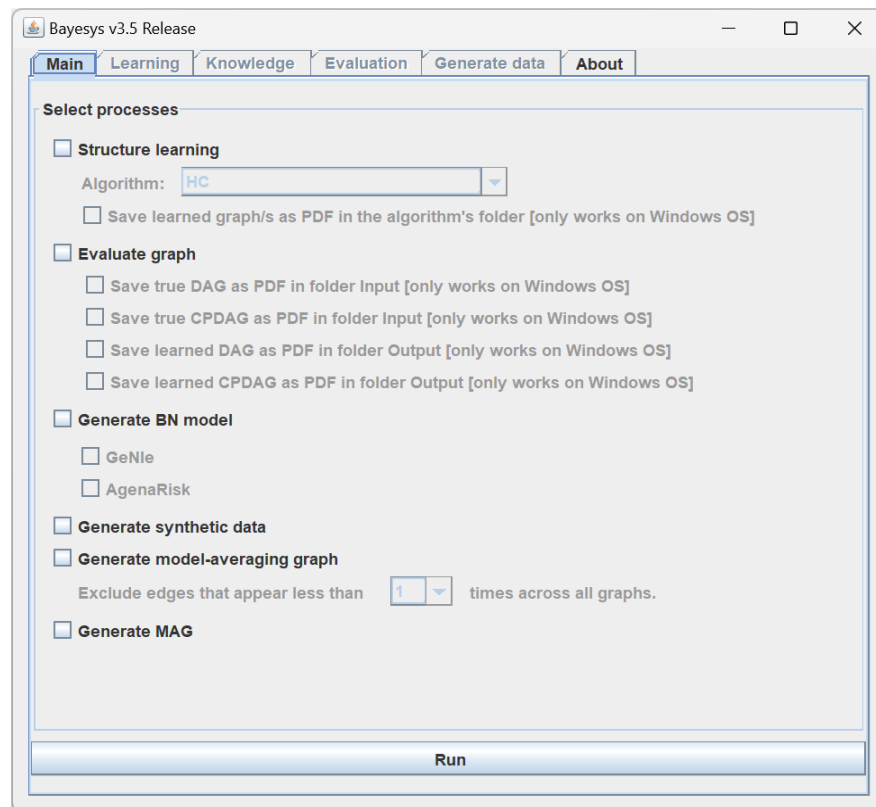
**Table 3.1** summarises the accuracy and speed of each of the six algorithms described above, based on synthetic data generated from six real-world networks that are available in the Bayesys repository [9]. Descriptions about the evaluation metrics can be found in Section 5.

**Table 3.1.** CPDAG scores, runtime, and BIC scores of the five algorithms available in Bayesys. The networks used to generate clean synthetic data are available in [7]. Runtime (secs) is based on a single-thread processing with CPU AMD R9 5950X on Windows 11. The last row presents the average scores (note SHD and BIC scaling might be biased) and runtime, where green and red colouring represents superior and inferior relative performance respectively. Note that HC and TABU are highly sensitive to the order of the variables as they appear in the data [10], whereas SaiyanH and MAHC are sensitive to a lower extent. These results assume default hyperparameter settings and are based on Bayesys v3.6 with JDK v1.7.

Network	Nodes	Edges	Sample size	HC					TABU					GES					SaiyanH					MAHC				
				BSF	SHD	F1	Runtime (secs)	BIC	BSF	SHD	F1	Runtime (secs)	BIC	BSF	SHD	F1	Runtime (secs)	BIC	BSF	SHD	F1	Runtime (secs)	BIC	BSF	SHD	F1	Runtime (secs)	BIC
Asia	8	8	10 <sup>2</sup>	0.575	4	0.667	0	-377.6	0.575	4	0.667	0	-377.6	0.638	3.5	0.733	0	-378.3	0.525	5	0.625	0	-380.0	0.575	4	0.667	0	-377.6
Asia	8	8	10 <sup>3</sup>	0.700	3	0.800	0	-3259.2	0.700	3	0.800	0	-3259.2	0.700	3	0.800	0	-3259.2	0.712	3.5	0.765	0	-3269.0	0.700	3	0.800	0	-3259.2
Asia	8	8	10 <sup>4</sup>	1.000	0	1.000	0	-32264.6	1.000	0	1.000	0	-32264.6	1.000	0	1.000	0	-32264.6	1.000	0	1.000	0	-32264.6	0.875	1	0.875	0	-32286.3
Asia	8	8	10 <sup>5</sup>	1.000	0	1.000	0	-322411.7	1.000	0	1.000	0	-322411.7	1.000	0	1.000	0	-322411.7	0.813	1.5	0.867	0	-326561.6	0.875	1	0.875	0	-322697.5
Sports	9	15	10 <sup>2</sup>	0.067	14	0.118	0	-1990.3	0.067	14	0.118	0	-1990.3	0.067	14	0.118	0	-1990.3	0.419	9	0.519	0	-5200.8	0.067	14	0.118	0	-1990.3
Sports	9	15	10 <sup>3</sup>	0.600	6	0.750	0	-16698.0	0.600	6	0.750	0	-16698.0	0.600	6	0.750	0	-16698.0	0.419	9	0.609	0	-16857.6	0.333	10	0.500	0	-17115.2
Sports	9	15	10 <sup>4</sup>	0.600	6	0.750	0	-158258.6	0.519	7.5	0.680	0	-158146.8	0.600	6	0.750	0	-158258.6	0.438	9	0.615	0	-159037.4	0.600	6	0.750	0	-158258.6
Sports	9	15	10 <sup>5</sup>	1.000	0	1.000	0	-1550580.0	1.000	0	1.000	0	-1550580.0	1.000	0	1.000	0	-1550580.0	1.000	0	1.000	2	-1550580.0	0.457	9	0.643	2	-1566413.7
Property	27	31	10 <sup>2</sup>	0.226	28.5	0.306	0	-5085.1	0.226	28.5	0.306	0	-5085.1	0.226	28.5	0.306	0	-5085.1	0.401	28.5	0.443	0	-22988.8	0.213	28	0.298	0	-5091.8
Property	27	31	10 <sup>3</sup>	0.536	18	0.596	0	-39710.3	0.536	18	0.596	0	-39710.3	0.497	21	0.542	0	-39884.8	0.507	22.5	0.569	0	-54854.5	0.474	19	0.577	0	-40432.2
Property	27	31	10 <sup>4</sup>	0.620	19	0.615	0	-348584.3	0.607	18.5	0.609	0	-348432.2	0.553	22	0.545	0	-348517.3	0.752	9.5	0.783	5	-358660.7	0.720	10.5	0.776	0	-355305.3
Property	27	31	10 <sup>5</sup>	0.665	18.5	0.632	5	-3379042.8	0.701	16.5	0.672	7	-3378929.0	0.630	20.5	0.594	5	-3376385.5	0.703	11	0.733	57	-3516923.5	0.845	7.5	0.869	6	-3450773.4
Diarrhoea	28	68	10 <sup>2</sup>	0.120	64.5	0.216	0	-2982.4	0.120	64.5	0.216	0	-2982.4	0.120	64.5	0.216	0	-2982.4	0.134	74.5	0.243	0	-4889.2	0.119	63	0.212	0	-2990.0
Diarrhoea	28	68	10 <sup>3</sup>	0.324	46	0.454	0	-28087.0	0.338	45	0.469	0	-28086.0	0.309	47	0.433	0	-28088.3	0.350	45	0.485	0	-28156.2	0.294	48	0.426	0	-28172.8
Diarrhoea	28	68	10 <sup>4</sup>	0.560	31.5	0.675	0	-272995.8	0.574	30.5	0.687	0	-272969.8	0.545	32.5	0.658	0	-273148.4	0.596	27.5	0.717	4	-273171.6	0.548	31.5	0.676	1	-273089.4
Diarrhoea	28	68	10 <sup>5</sup>	0.799	16	0.833	7	-2705054.1	0.799	16	0.833	10	-2704993.8	0.818	15.5	0.856	8	-2705622.5	0.647	24	0.759	48	-2710582.6	0.813	13.5	0.860	55	-2704523.8
Alarm	37	46	10 <sup>2</sup>	0.264	38.5	0.338	0	-2219.1	0.264	38.5	0.338	0	-2219.1	0.264	38.5	0.338	0	-2219.1	0.468	43	0.479	0	-3722.5	0.223	38.5	0.309	0	-2303.0
Alarm	37	46	10 <sup>3</sup>	0.652	22.5	0.670	0	-17109.2	0.652	22.5	0.670	0	-17092.9	0.652	22.5	0.670	1	-17109.2	0.675	27	0.667	0	-17517.6	0.557	25	0.612	0	-17659.1
Alarm	37	46	10 <sup>4</sup>	0.725	21	0.701	0	-154627.1	0.725	21	0.701	0	-154627.1	0.725	21	0.701	2	-154627.1	0.876	8.5	0.880	8	-153073.5	0.711	17	0.725	2	-154381.9
Alarm	37	46	10 <sup>5</sup>	0.826	14.5	0.794	7	-1510655.1	0.827	13.5	0.802	10	-1510476.1	0.826	14.5	0.794	9	-1510780.7	0.706	20	0.688	80	-1528637.1	0.753	16	0.737	37	-1525408.3
ForMed	88	138	10 <sup>2</sup>	0.319	132.5	0.392	1	-7139.6	0.326	131.5	0.399	5	-7136.3	0.316	132	0.390	77	-7148.7	0.260	194.5	0.271	6	-93505.0	0.247	114.5	0.350	2	-7263.8
ForMed	88	138	10 <sup>3</sup>	0.644	66.5	0.705	2	-63027.5	0.648	65	0.711	5	-63017.7	0.621	72.5	0.681	147	-63263.6	0.512	113.5	0.520	17	-131780.9	0.506	71	0.619	11	-64038.9
ForMed	88	138	10 <sup>4</sup>	0.810	43.5	0.809	7	-608020.9	0.818	42.5	0.814	12	-607941.1	0.791	52	0.775	264	-608656.9	0.658	53	0.740	120	-614544.9	0.731	39	0.805	50	-610503.2
ForMed	88	138	10 <sup>5</sup>	0.638	125	0.542	92	-6020284.1	0.642	123.5	0.546	140	-6020277.9	0.619	132.5	0.521	495	-6022242.0	0.698	45.5	0.772	1252	-6098168.4	0.875	23	0.900	280	-6022568.9
Pathfinder	109	195	10 <sup>2</sup>	0.106	223.5	0.165	1	-6512.7	0.109	223	0.168	5	-6501.1	0.106	223.5	0.165	37	-6512.7	0.156	265	0.201	5	-30999.2	0.149	195	0.231	2	-6677.4
Pathfinder	109	195	10 <sup>3</sup>	0.189	231.5	0.252	3	-51632.6	0.189	231.5	0.252	6	-51632.6	0.189	230.5	0.252	146	-51632.8	0.216	275.5	0.246	26	-140104.7	0.146	224.5	0.207	11	-53873.3
Pathfinder	109	195	10 <sup>4</sup>	0.288	231.5	0.333	11	-413262.2	0.288	231.5	0.334	23	-411614.6	0.288	231.5	0.333	324	-413232.1	0.261	246.5	0.298	208	-502086.1	0.233	235.5	0.284	112	-438628.6
Pathfinder	109	195	10 <sup>5</sup>	0.481	161	0.530	115	-3632770.3	0.481	162	0.529	135	-3630497.9	0.481	161	0.530	381	-3633999.5	0.323	202.5	0.384	2304	-4079289.7	0.452	159	0.519	3836	-3707912.5
COVID-19	17	37	10 <sup>2</sup>	0.152	32	0.245	0	-2718.1	0.142	33	0.240	0	-2715.2	0.152	32	0.245	0	-2718.1	0.169	34.5	0.298	0	-6194.6	0.135	32	0.217	0	-2778.0
COVID-19	17	37	10 <sup>3</sup>	0.416	23.5	0.541	0	-21364.1	0.416	23.5	0.541	0	-21364.1	0.416	23.5	0.541	0	-21364.1	0.409	22.5	0.517	0	-21785.4	0.395	23	0.536	0	-22038.0
COVID-19	17	37	10 <sup>4</sup>	0.554	19	0.620	0	-179373.9	0.554	19	0.620	0	-179373.9	0.544	20	0.611	0	-180904.0	0.679	15	0.761	1	-181953.0	0.639	14	0.716	1	-185765.6
COVID-19	17	37	10 <sup>5</sup>	0.538	24	0.585	2	-1684320.9	0.551	23.5	0.590	3	-1681354.4	0.514	25.5	0.573	2	-1686436.1	0.814	10	0.842	15	-1685243.7	0.727	14.5	0.747	36	-1696371.8
DIABETES	22	37	10 <sup>2</sup>	0.005	39	0.048	0	-1904.2	0.005	39	0.048	0	-1904.2	0.005	39	0.048	0	-1904.2	-0.009	54	0.174	0	-4075.9	0.012	38	0.049	0	-1904.3
DIABETES	22	37	10 <sup>3</sup>	0.236	29	0.333	0	-18090.1	0.236	29	0.333	0	-18090.1	0.236	29	0.333	0	-18090.1	0.227	38	0.343	0	-19114.8	0.222	29.5	0.321	0	-18104.4
DIABETES	22	37	10 <sup>4</sup>	0.486	19	0.581	0	-178821.8	0.486	19	0.581	0	-178821.8	0.411	22.5	0.492	0	-179069.0	0.476	23	0.551	1	-178874.6	0.486	19	0.581	0	-178824.5
DIABETES	22	37	10 <sup>5</sup>	0.790	8.5	0.819	3	-1775337.5	0.817	7.5	0.847	4	-1775160.3	0.721	12.5	0.743	2	-1775983.2	0.599	17	0.676	18	-1778641.8	0.811	7	0.857	4	-1775820.3
Average score/runtime				0.514	49.46	0.567	7.1	-700460	0.515	49.21	0.569	10.1	-700243	0.505	50.56	0.557	52.8	-700651	0.516	55.22	0.584	116	-731491	0.487	44.6	0.562	123.6	-707100

### 3.2. Structure learning with categorical data

Once you load Bayesys in NetBeans IDE, as illustrated in Section 1, click the *Run* button. The window shown in **Figure 3.1** should appear.



**Figure 3.1.** Methods available under tab Main in Bayesys v3.5.

Select the method *Structure learning* under tab *Main*. This will activate tab *Learning* which provides access to some algorithm-specific parameter inputs, some of which can be modified. Before you run *Structure learning*, ensure you have placed the input data set named *trainingData.csv* in folder *Input*. For an example, see subsection 7.1. Note that:

- a) Each column in *trainingData.csv* represents a variable (do *not* include an ID column), and each row a data sample.
- b) All variables are assumed to be categorical. The algorithm assumes a unique category for each unique variable value, which makes it unsuitable for structure learning with continuous data.
- c) There should not be any empty cells in *trainingData.csv*. If your data set contains missing values, you could address this using a data imputation algorithm, such as the MBMF (Markov-Blanket MissForest) which is suitable for both random and systematic missingness [11]; direct link to GitHub package [here](#). Alternatively, you could replace all empty cells with a new state called “missing”, or any other state name to differentiate it from observed values. This way the algorithm performs structure learning under the assumption that the missing values are *not* missing at random.

### 3.3. Structure learning outputs

The structure learning process generates three types of output. These are:

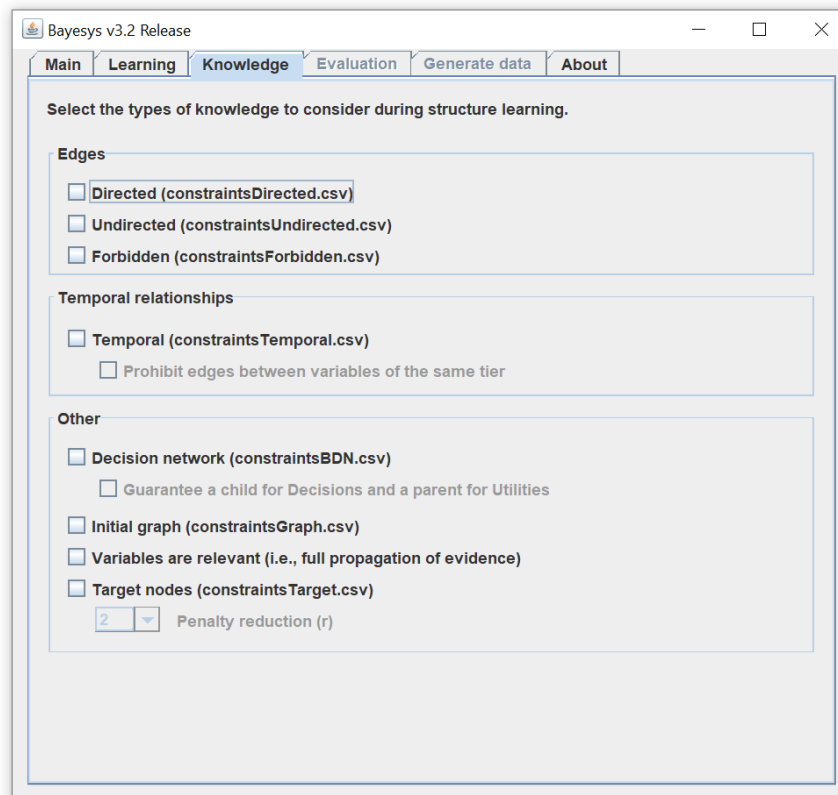
- a) **Edges:** the file *DAGlearned.csv* in folder *Output*. This file contains all the arcs discovered between variables.
- b) **Graphs (optional):** a drawing of the true and learned DAGs and CPDAGs in PDF format. The true graphs are generated in folder *Input* and the learned graphs in folder *Output*. These features are optional and can be selecting under *Evaluate graph* in tab *Main*.
- c) **Algorithm-specific outputs (optional):** Some algorithms can produce additional output information. For example, selecting *SaiyanH* under tab *Main*, followed by *Save learned graph/s as PDF in the algorithm's folder*, produces three PDF files in directory *Output/SaiyanH*. Each of those PDF files corresponds to the graph produced at the end of each of the three learning phases in SaiyanH.

Additional output information for SaiyanH can be found under tab *Learning* → *SaiyanH*. Selecting *Save associational scores* generates four CSV files, in directory *Output/SaiyanH*, that capture the marginal associational scores generated during the first structure learning (one file), and the conditional associational scores generated during the second phase (three files).

For troubleshooting and things to know, refer to Section 13.

## 4. Structure learning with knowledge-based constraints

The *Structure learning* process also activates tab *Knowledge* which can be used to incorporate various soft and hard knowledge-based constraints. **Figure 4.1** presents the methods available under tab *Knowledge*. **Table 4.1** summarises each of these constraints, and further details are available in [6].



**Figure 4.1.** The knowledge approaches available under tab Knowledge in Bayesys v3.2.

In brief, the knowledge approaches are:

**Directed:** represents directed edges that should be preserved during structure learning. This knowledge approach requires the file *constraintsDirected.csv* in folder *Input*. An example of this file is shown in **Figure 4.2**, where the encoding specifies the arcs that must be preserved in the search space of graphs. In this example, the search space of graphs will be restricted to those containing  $A \rightarrow B$ ,  $F \rightarrow G$  and  $T \rightarrow A$ .

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

**Figure 4.2.** An example of Directed constraints encoded in input file *constraintsDirected.csv*.



**Undirected:** represents edges that should be preserved during structure learning. This knowledge approach requires the file *constraintsUndirected.csv* in folder *Input*. An example of this input file is shown in **Figure 4.3**, where the encoding specifies the edges that must be preserved in the search space of graphs. In this example, the search space of graphs will be restricted to those containing  $A \rightarrow B$  or  $A \leftarrow B$ ,  $F \rightarrow G$  or  $F \leftarrow G$ , and  $T \rightarrow A$  or  $T \leftarrow A$ .

	A	B	C
1	ID	Var1	Var2
2	1	A	B
3	2	F	G
4	3	T	A

**Figure 4.3.** An example of Undirected constraints encoded in input file *constraintsUndirected.csv*.

**Forbidden:** represents edges that should not be explored during structure learning. This knowledge approach requires the file *constraintsForbidden.csv* in folder *Input*. An example of this input file is shown in **Figure 4.4**, where the encoding specifies the edges that should not be investigated in the search space of graphs. In this example, the search space of graphs will be restricted to those satisfying  $A \perp B$ ,  $F \perp G$  and  $T \perp A$ .

	A	B	C
1	ID	Var1	Var2
2	1	A	B
3	2	F	G
4	3	T	A

**Figure 4.4.** An example of Forbidden constraints encoded in input file *constraintsForbidden.csv*.

**Temporal:** represents temporal restrictions during structure learning. This knowledge approach requires the file *constraintsTemporal.csv* in folder *Input*. An example of this input file is shown in **Figure 4.5**, where the encoding specifies that a variable within a higher tier cannot serve as a parent or an ancestor of a variable within a lower tier.

Note that not all the variables need to be assigned to a tier. Bayesys assumes that a variable not assigned to a particular tier is under no temporal restrictions. Lastly, temporal restrictions can be extended to prohibit edges between variables of the same tier, by selecting the sub-process *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	

**Figure 4.5.** Hypothetical Temporal constraints encoded in input file *constraintsTemporal.csv*. The column ID (represents the max number of constraints in a tier) and column END must be present in this input file as shown in this figure.

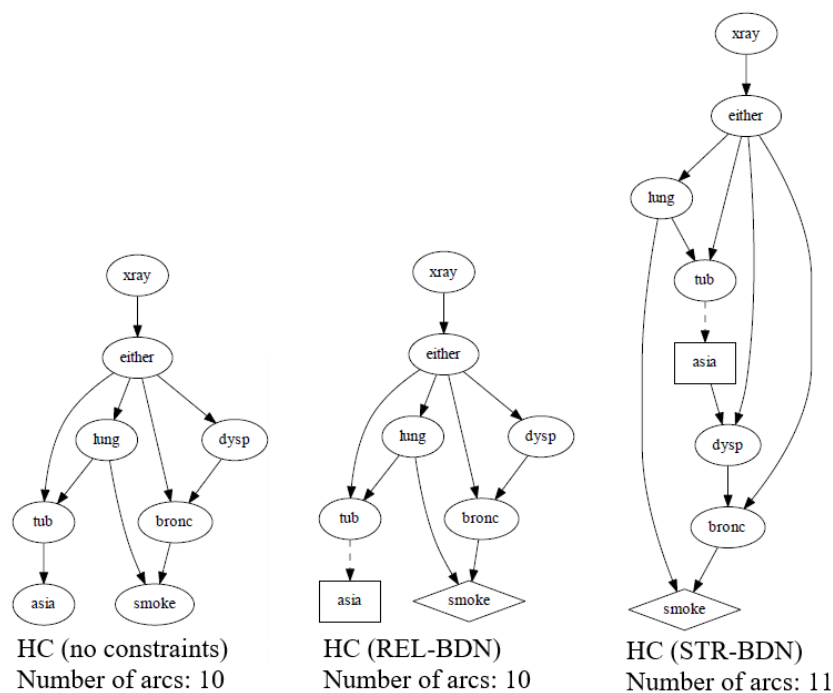
**Decision network:** performs the necessary visual modifications needed to convert the generated PDF graph from a BN graph into a Bayesian Decision Network (BDN; also known as an Influence Diagram) graph, where *Decision* nodes are represented by rectangles, *Utility* nodes by diamonds, and *Informational arcs* entering *Decisions* by dashed arcs. This knowledge approach requires the file *constraintsBDN.csv* in folder *Input*. An example of this input file is shown in **Figure 4.6**, where the encoding specifies which nodes represent ‘decisions’ and which nodes represent ‘utility’. The data column *State to maximise* is useful only when the learned graph is converted into a BDN model in GeNIe (refer to subsections 6.1 and 7.3).

	A	B	C	D	E
1	ID	Node	Node type	State to maximise	
2	1	VarName2	Decision	n/a	
3	4	VarName6	Decision	n/a	
4	3	VarName3	Utility	yes	

**Figure 4.6.** An example of BDN constraints encoded in input file *constraintsBDN.csv*.

Further, selecting the sub-process *Guarantee a child for Decisions and a parent for Utilities* restricts structure learning such that ‘decision’ nodes will have at least one child node and ‘utility’ nodes will have at least one parent node. This restriction is imposed at the end of each structure learning algorithm, where further arcs are added to the learned graph, such that minimally decrease the BIC score, until the BDN conditions are met.

**Figure 4.7** illustrates a case where the graph on the left represents the standard BN graphical output generated by HC-DAG for the ASIA network. The graph in the middle represents the output of approach *Decision network* with *asia* specified as a ‘decision’ node and *smoke* specified as a ‘utility’ node in *constraintsBDN.csv*. Lastly, the graph on the right represents the modified graph produced by the sub-process *Guarantee a child for Decisions and a parent for Utilities*. The ‘decision’ and ‘utility’ and selected purely for illustration purposes.



**Figure 4.7.** How the learned graph on the left (in this example generated by HC) can be influenced by the approach *Decision networks* and its sub-process *Guarantee a child for Decisions and a parent for Utilities*.

**Initial graph:** represents a soft constraint that guides structure learning from a given best-guess initial graph, rather than from an empty graph. This knowledge approach requires the file *constraintsGraph.csv* in folder *Input*. The structure of this input file is identical to the example shown in **Figure 4.2** on *Directed* edges, but in this case the encoding specifies which arcs must be present in the initial, rather than the learned, graph.

**Variables are relevant:** imposes the restriction on the learned graph not to contain disjoint subgraphs or nodes. This constraint is inherited from SaiyanH which incorporates this restriction by design. Therefore, selecting *Variables are relevant* while performing structure learning with SaiyanH should make no difference to the learned output; but it is useful for all the other available algorithms.

**Target nodes:** encourages structure learning to produce a higher number of potential causes for targeted variables of interest. It can be useful when working with high dimensional data of limited sample size, which tend to lead to sparse networks that do not adequately capture all the potential causes of effects of interest.

Specifically, this soft constraint encourages structure learning to produce larger parent-sets for targeted nodes by diminishing the dimensionality penalty of the BIC score. This is achieved by decreasing the rate of increase of the number of free parameters given as an input to the BIC score, for a targeted variable. The modified  $BIC_r$  for graph  $G$  and data  $D$  is:

$$BIC_{TAR-VAR} = LL(G|D) - \left(\frac{\log_2 N}{2}\right)p$$

where  $N$  is the sample size of  $D$ , and  $p$  is the adjusted number of free parameters in  $G$ , given

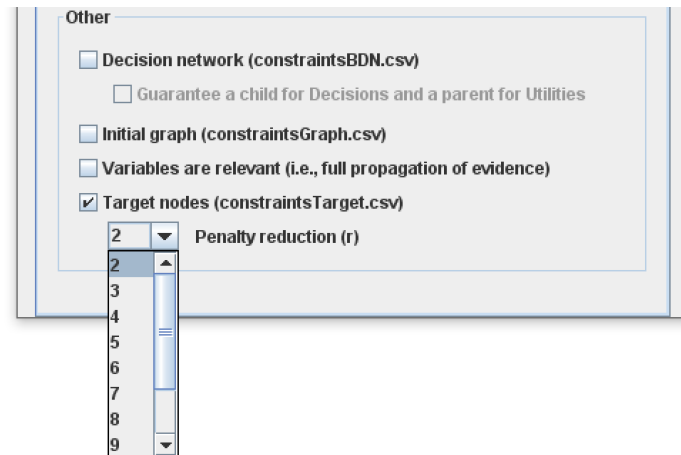
$$p = \sum_i^{|V|} \left( (s_i - 1) \prod_j^{|\pi_{v_i}|} q_j \right) / r_i$$

where  $V$  is the set of variables in graph  $G$ ,  $|V|$  is the size of set  $V$ ,  $s_i$  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  $\pi_{v_i}$ ,  $q_j$  is the number of states of  $v_j$  in parent set  $\pi_{v_i}$ , and  $r_i$  is the new parameter used to diminish the free parameters penalty for targeted variables; i.e.,  $r = 1$  for non-targeted variables and  $r > 1$ , as determined by the user, for targeted variables.

This knowledge approach requires the file *constraintsTarget.csv* in folder *Input*. An example of this input file is shown in **Figure 4.8**, where the encoding specifies which nodes should be targeted. **Figure 4.9** presents how to set the penalty reduction  $r_i$  through the UI, which can be found as a parameter input under the *Target nodes* constraint.

	A	B
1	ID	Target node
2	1	VarName1
3	2	VarName1
4	3	VarName1

**Figure 4.8.** An example of Target nodes constraint encoded in input file *constraintsTarget.csv*.



**Figure 4.9.** How to specify the dimensionality penalty reduction parameter input  $r$ , for the Target nodes constraint.

Many of the knowledge approaches can be combined. For example, the *Directed* constraint can be combined with the *Forbidden* and *Temporal* constraints, but not with the *Undirected* constraint. **Figure 4.10** presents an example of the output information generated in the terminal window of NetBeans regarding the number of restrictions imposed into the structure learning process, by each constraint approach.

```

_____ Knowledge-based constraints _____
Directed dependence constraints specified: 0
Temporal variables specified: 0 over 0 tiers.
Prohibit edges between variables of the same tier: false
Undirected dependence constraints specified: 0
Forbidden edges specified: 0
Input graph relationships specified: 0
Target variables specified: 0.
Decision nodes specified: 0.
Utility nodes specified: 0.

```

**Figure 4.10.** An example of the output information generated in the terminal window of NetBeans regarding the number of restrictions imposed into the structure learning process by each knowledge approach. The example assumes no constraints and is based on Bayesys v2.2.

For troubleshooting and things to know, refer to Section 13.

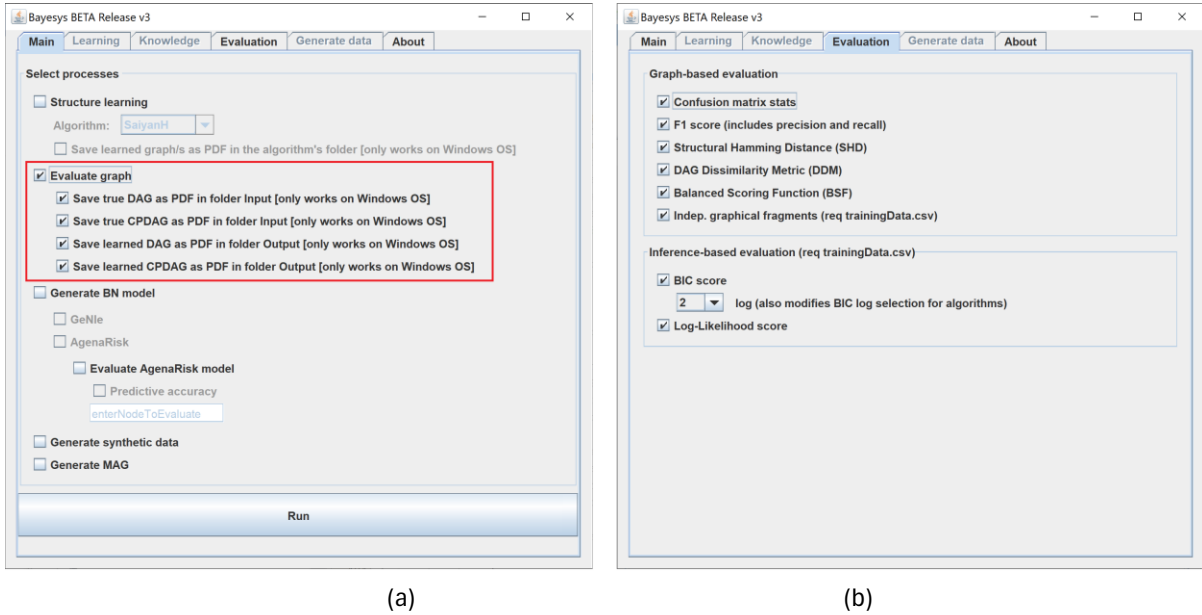
**Table 4.1.** The soft and hard knowledge-based constraints as described in [6].

ID	Knowledge approach	Knowledge input example	Knowledge	Constrains or guides
DIR-EDG	Directed edge	$A \rightarrow B$	Causal relationship or direct dependency.	Constrains the search space of graphs to those containing $A \rightarrow B$ .
UND-EDG	Undirected edge	$A - B$	Causal relationship or direct dependency without knowledge of the direction of the relationship.	Constrains the search space of graphs to those containing $A \rightarrow B$ or $A \leftarrow B$ .
FOR-EDG	Forbidden edge	$A \perp B$	No causal relationship or direct dependency.	Constrains the search space of graphs to those not containing $A \rightarrow B$ and $A \leftarrow B$ .
REL-TEM	Relaxed incomplete temporal order	Tier 1: $\{A\}$ Tier 2: $\{B, C\}$	Temporal information such that $B$ and $C$ occur after observing $A$ and hence, $B$ and $C$ cannot be parents nor ancestors of $A$ .	Constrains the search space of graphs to those not containing $A \leftarrow B$ , $A \leftarrow C$ , or $B$ and/or $C$ as ancestors of $A$ .
STR-TEM	Strict incomplete temporal order	Tier 1: $\{A\}$ Tier 2: $\{B, C\}$	In addition to constraint $REL - TEM$ , it prohibits edges between nodes of the same tier.	As in REL-TEM, plus constrains the search space of graphs to those not containing $B \rightarrow C$ or $B \leftarrow C$ .
INI-GRA	Initial graph	DAG	An initial best guess graph	Can be viewed as a soft constraint that guides structure learning by changing the starting point in the search space of graphs from an empty graph to an initial best-guess graph.
VAR-REL	Variables are relevant	n/a	All variables in the input data are relevant.	The learnt graph must not contain disjoint subgraphs or unconnected nodes.
TAR-VAR	Target variable/s	A node $A$ or a set of nodes $\{A, B\}$	A target variable, or a set of variables, for which we would favour higher dimensionality in the form of an increased parent-set for those variables.	Can be viewed as a soft constraint that guides the search space of graphs to those in which targeted variable $A$ , or variable set $\{A, B\}$ , is assigned a relaxed dimensionality penalty, as determined by the user, through the objective function.
REL-BDN	Relaxed BDNs	A set of Decision nodes $\{A, B\}$ and Utility nodes $\{C, D\}$	Some of the input variables represent decisions or utilities.	The learnt graph is a BDN graph containing Decisions $\{A, B\}$ and Utilities $\{C, D\}$ represented by rectangle and diamond nodes respectively, where arcs entering Decisions are Informational represented by a dashed arc.
STR-BDN	Strict BDNs	A set of Decision nodes $\{A, B\}$ and Utility nodes $\{C, D\}$		As in REL-BDN, plus constrains the learnt graphs to those where Decision nodes $\{A, B\}$ and Utility nodes $\{C, D\}$ have at least one child and parent node respectively.

## 5. Evaluating graphical structures

The process *Evaluate graph* can be used to evaluate the accuracy of the learned graph with reference to the true graph. As shown in **Figure 5.1a**, this process also gives us the option to generate PDFs of the true DAG and true CPDAG in folder *Input*, as well as the learned DAG and learned CPDAG in folder *Output*. Moreover, selecting *Evaluate graph* in tab *Main* also activates the tab *Evaluation* shown in **Figure 5.1b**, which gives the option to the user to indicate which evaluation scores to generate (all scores are selected by default). Lastly, *Evaluate graph* can be performed with or without *Structure learning*. For an example, see subsection 7.2.

Note that when *Saved learned DAG as PDF*, under *Evaluate graph*, is combined with the *Decision network* knowledge approach under tab *Knowledge* (refer to Section 4), the PDF graph will represent a Bayesian Decision Network (BDN) that includes ‘decision’ and ‘utility’ nodes, and will be named *BDNlearned.PDF* instead of *DAGlearned.pdf*. This process takes into consideration both the *DAGlearned.csv* and *constraintsBDN.csv* files (refer to **Figure 4.6**).



**Figure 5.1.** The Evaluate graph process, along with the available graph-based and inference-based metrics under tab Evaluation. The figure is based on Bayesys v3.

The first five graph-based metrics under tab *Evaluation* require the following two input files:

- the *DAGtrue.csv* in folder *Input*, and
- the *DAGlearned.csv* in folder *Output*,

encoded as shown in **Figure 5.2**. If we run *Evaluate graph* together with *Structure learning*, the *DAGlearned.csv* file will be generated in folder *Output* at the end of the structure learning process, before the system runs the evaluation method. If we run *Evaluate graph* without *Structure learning*, we would need to manually place the *DAGlearned.csv* file in folder *Output*, in addition to the *DAGtrue.csv* file in folder *Input*, before running the evaluation.

	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

**Figure 5.2.** An example of the DAGtrue/DAGlearned.csv input/output file.

The five scoring metrics are:

**Confusion matrix stats:** this metric produces the following scores:

- *True Positives (TP)*: the number of correct edges present in the learned graph.
- *Half True Positives ( $TP \times 0.5$ )*: the number of partially correct edges (e.g.,  $\leftarrow$ ,  $-$ , or  $\leftrightarrow$  instead of  $\rightarrow$ ) present in the learned graph.
- *False Positives (FP)*: the number of incorrect edges present in the learned graph.
- *True Negatives (TN)*: the number of correct edges absent in the learned graph.
- *False negatives (FN)*: the number of incorrect edges absent in the learned graph (i.e., edges not discovered).

**Precision:** defined as  $TP/(TP + FP)$ , represents the rate of correct edges over all edges discovered.

**Recall:** defined as  $TP/(TP + FN)$ , represents the rate of edges discovered over all edges in the true graph.

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

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

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.

**Structural Hamming Distance (SHD):** defined as the minimum number of edge insertions, deletions, and arc reversals needed to transform the learned graph into the true graph. In Bayesys the number of insertions and deletions generate a penalty of 1, while arc reversals generate a penalty of 0.5, as we later define in **Table 5.1** and **Table 5.2**. This adjustment is made to acknowledge the fact that an arc reversal represents correct dependency but with an incorrect direction.

**DAG Dissimilarity Metric (DDM):** a score that ranges from  $-\infty$  to 1, where the 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:

$$\text{DDM} = \frac{TP + \frac{c}{2} - FN - FP}{t}$$

where  $c$  is the number of edges in  $B$  that are reoriented in  $A$ , and  $t$  is total number of edges in  $B$ .

**Balanced Scoring Function (BSF):** This is a metric that takes into consideration all the four confusion matrix parameters (i.e., TP, TN, FP, and FN) to balance the score between direct independencies and direct dependencies [12]. Specifically:

$$\text{BSF} = \frac{1}{2} \left( \frac{TP}{a} + \frac{TN}{i} - \frac{FP}{i} - \frac{FN}{a} \right)$$

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

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

where  $|V|$  is the number of variables in the data; i.e., the size of set  $V$  containing the data variables.

The BSF score ranges from -1 to 1, where a score of -1 corresponds to the worst possible graph (i.e., the reverse of the true graph), a score of 1 corresponds to a perfect match between the learned and the true graphs, and a score of 0 corresponds to a learned graph that is equivalent to the score generated by an empty or a fully connected graph.

**Table 5.1.** The DAG penalty weights used by the scoring metrics implemented in Bayesys.

Rule	True graph	Learned graph	Penalty	Reasoning
1	$A \rightarrow B$	$A \rightarrow B, A \circ \rightarrow B$	0	Complete match
2	$A \rightarrow B$	$A \leftrightarrow B, A - B, A \leftarrow B, A \leftarrow \circ B$	0.5	Partial match
3	any edge	no edge	1	No match
4	$A \leftrightarrow B$	$A \leftrightarrow B$	0	Complete match
5	$A \leftrightarrow B$	$A - B, A \leftarrow B, A \rightarrow B, A \leftarrow \circ B, A \circ \rightarrow B$	0.5	Partial match
6	no edge	no edge	0	Complete match
7	no edge	Any edge/arc	1	No match

**Table 5.2.** The CPDAG penalty weights used by the scoring metrics implemented in Bayesys [7].

Rule	True graph	Learned graph	Penalty	Reasoning
1	$A \rightarrow B$	$A \rightarrow B, A \circ \rightarrow B$	0	Complete match
2	$A \rightarrow B$	$A \leftrightarrow B, A - B, A \leftarrow B, A \leftarrow \circ B$	0.5	Partial match
3	any edge	no edge	1	No match
4	$A \leftrightarrow B$	$A \leftrightarrow B$	0	Complete match
5	$A \leftrightarrow B$	$A - B, A \leftarrow B, A \rightarrow B, A \leftarrow \circ B, A \circ \rightarrow B$	0.5	Partial match
6	$A - B$	$A - B$	0	Complete match
7	$A - B$	$A \leftrightarrow B, A \leftarrow B, A \rightarrow B, A \leftarrow \circ B, A \circ \rightarrow B$	0.5	Partial match
8	no edge	no edge	0	Complete match
9	no edge	Any edge/arc	1	No match



While all the algorithms implemented in Bayesys produce a DAG, the five above graphical-based metrics are used to produce both DAG and CPDAG scores as described in **Table 5.1** and **Table 5.2** respectively. Note that the scoring functions also consider the case of a Mixed Ancestral Graph (MAG); i.e., the file *DAGlearned.csv* can contain undirected and bidirected edges. This is useful when trying to evaluate a graph produced by an algorithm available in some other structure learning software.

In addition to the five graphical-based metrics, selecting *Indep. graphical fragments* will return the number of independent graphical fragments (i.e., disjoint subgraphs/nodes) found in the learned graph. Note this method requires the file *trainingData.csv* to be present in folder *Input*, in addition to the *DAGlearned.csv* and *DAGtrue.csv* files.

Lastly, selecting the inference-based metric *BIC score* will output the Bayesian Information Criterion (BIC) score of the learned graph. This method requires the file *trainingData.csv* in folder *Input* and the file *DAGlearned.csv* in folder *Output*. Assuming<sup>3</sup>  $\log_2$ , the BIC score in Bayesys is computed as follows:

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

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

$$p = \sum_i^{|V|} (s_i - 1) \prod_j^{|\pi_{v_i}|} q_j$$

where  $s_i$  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  $\pi_{v_i}$ . Note that selecting the metric *Log-Likelihood score* produces  $LL(G|D)$ , which is the first part of the BIC equation and represents a score that tells us how well the learned distributions fit the empirical distributions; whereas the second part of the BIC equation represents the dimensionality penalty of the model.

The *BIC score* can also be computed for MAGs that include bi-directed edges. When a bi-directed edge is found in *DAGlearned.csv*, Bayesys will estimate the BIC MAG score by enumerating all possible valid DAGs that could have been produced via all directed combinations of bi-directed edges, and return the average BIC score over those valid DAGs.

For troubleshooting and things to know, refer to Section 13.

---

<sup>3</sup> The UI provides a selection of different log scales for computing the BIC score.

## 6. Learning Bayesian network models

### 6.1. *Converting a learned graph into a Bayesian network model in GeNIe*

This step requires that you download and install the GeNIe software, which is “a tool for artificial intelligence modelling and machine learning with Bayesian networks and other types of graphical probabilistic models”. GeNIe can be downloaded from <https://www.bayesfusion.com/downloads/>. Note that GeNIe is free for academic users or for academic teaching and research.

A GeNIe BN model can be generated by selecting *Generate BN model* and the subprocess *GeNIe*, both of which can be found under tab *Main*. This method requires the input files *trainingData.csv* in folder *Input* and *DAGlearned.csv* in folder *Output*, and can be performed with or without *Structure learning*. The output of this process will be a file called *GeNIe\_BN.xdsl* in folder *Output*, which can be loaded into the GeNIe software. Note that if knowledge approach *Decision network* (under tap *Knowledge*) is selected in conjunction with *Structure learning*, then the generated file will be a BDN, instead of a BN, named *GeNIe\_BDN.xdsl*. For some examples, see subsection 7.3.

### 6.2. *Converting a learned graph into a Bayesian network model in AgenaRisk*

*Note that, as of Bayesys v3.01, the AgenaRisk functions are not working due compatibility issues with the new version of AgenaRisk Developer license which requires active license and internet connection.*

This step requires that you download and install the AgenaRisk software from [www.agenarisk.com](http://www.agenarisk.com). If you are an AgenaRisk user license holder, you should download the relevant version appropriate for your license. Alternatively, you could try the 14-day trial Desktop version which can be downloaded from <https://www.agenarisk.com/agenarisk-free-trial>

An AgenaRisk BN model can be generated by selecting *Generate BN model* and the subprocess *AgenaRisk*, both of which can be found under tab *Main*. This method also requires the input files *trainingData.csv* in folder *Input* and *DAGlearned.csv* in folder *Output*, and can be performed with or without *Structure learning*. The output of this process will be a file called *AgenaRisk\_BN.cmp* in folder *Output*, and which can be loaded into the AgenaRisk software. Note that the AgenaRisk BN model generator is based on AgenaRisk’s SDK 6120. For an example, see subsection 7.4.


## 7. Worked example: Structure learning, evaluation, and parameterisation of a BN model.

The example illustrated in this section is based on the HC algorithm and the ASIA network with data sample size 10k. The data file can be found in folder *Sample input files/Structure learning*, in the main directory of the NetBeans project. These sample files are taken from the Bayesys data repository [9].

### 7.1. Structure learning

Copy the two ASIA files *DAGtrue\_ASIA.csv* and *trainingData\_ASIA\_10k.csv* (from the *Sample input files* folder) in folder *Input*, and rename them into *DAGtrue.csv* and *trainingData.csv* respectively. Keep in mind that the file *DAGtrue.csv* is needed only for the *Evaluation* step described in subsection 7.2.

Run Bayesys and under tab *Main* select *Structure learning*, set the algorithm to HC (default selection). If you are on Windows OS, also tick *Save learned graph/s as PDF in the algorithm's folder*; leave this unticked if on Mac OS, as this function would return an error due to compatibility issues (see (iii) below for an alternative solution). Hit *Run*. You should see the output depicted in **Figure 7.1** in the terminal window of NetBeans.

Note that the '**BUILD**' time shown in the output corresponds to the time the application was building and running (i.e., from the time we hit the run button  until the application is closed), and includes the time taken by the user to select the different features to be executed. As shown in **Figure 7.1**, the actual *Structure learning elapsed time* (i.e., the time needed by the algorithm to learn a graph) in this example was 0 seconds.

```
run:
_____ Training data info _____
Variables: 8
Sample size: 10000
_____ Knowledge-based constraints _____
Directed edge constraints specified: 0
Temporal variables specified: 0 over 0 tiers.
Prohibit edges between variables of the same tier: false
Undirected edge constraints specified: 0
Forbidden edges specified: 0
Input graph relationships specified: 0
Target variables specified: 0.
Decision nodes specified: 0.
Utility nodes specified: 0.
_____ Structure learning _____
Running HC with settings:
  a) Initial graph: Empty
  b) BIC log: 2
Entering score-based learning HC search...
HC search completed.
Structure learning elapsed time: 0 seconds.
BUILD SUCCESSFUL (total time: 1 minute 50 seconds)
```

**Figure 7.1.** The output information generated in the terminal window of NetBeans IDE after performing structure learning with HC on the ASIA network, using the sample data set with 10k sample size. The output information is based on Bayesys v3.3.

The previous process should have also generated the following files:

- i. *DAGlearned.csv* in folder *Output*, as shown in **Figure 7.2**.

	A	B	C	D
1	ID	Variable 1	Dependency	Variable 2
2		1 asia	->	tub
3		2 tub	->	either
4		3 smoke	->	lung
5		4 smoke	->	bronc
6		5 lung	->	either
7		6 bronc	->	dysp
8		7 either	->	xray
9		8 either	->	dysp

**Figure 7.2.** The edges recorded in the DAGlearned.csv file based on Bayesys v3.3.

- ii. *CPDAGlearned.csv* in folder *Output*, as shown in **Figure 7.3**.

	A	B	C	D
1	ID	Variable 1	Dependen	Variable 2
2		1 asia	-	tub
3		2 tub	->	either
4		3 smoke	-	lung
5		4 smoke	-	bronc
6		5 lung	->	either
7		6 bronc	->	dysp
8		7 either	->	xray
9		8 either	->	dysp

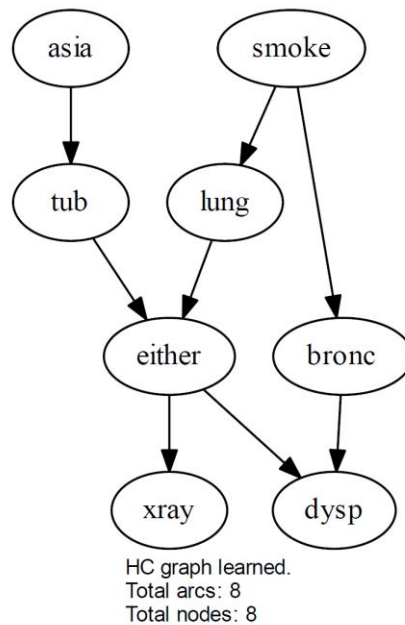
**Figure 7.3.** The edges recorded in the CPDAGlearned.csv file based on Bayesys v3.3.

- iii. *HC.pdf* in directory *Output/HC*, shown in **Figure 7.4**. If you are working on MAC/Linux OS, the DAGlearned.pdf file is likely to be corrupted, or fail to generate and return an error, due to compatibility issues. You can use an online Graphviz editor, such as the one available here: <https://edotor.net/>. This editor turns a textual representation of a graph into a visual drawing. Using the code shown below, which corresponds to the relationships recorded in DAGlearned.csv, should produce the same graph as in **Figure 7.4**.

```

digraph {
    asia -> tub
    tub -> either
    smoke -> lung
    smoke -> bronc
    lung -> either
    bronc -> dysp
    either -> xray
    either -> dysp
}

```



**Figure 7.4.** The DAG generated in file HC.pdf based on Bayesys v3.3.

## 7.2. Evaluate graph

The *Evaluate graph* method can be performed together with *Structure learning*, or on its own. Since we have already performed the structure learning process at the previous step, we will now only run *Evaluate graph*. Recall that the file *DAGtrue.csv*, which is needed for this step, was added in folder *Input* during the previous step described in subsection 7.1.

Run Bayesys and select *Evaluate graph* under tab *Main*, and then select all four *Save...* options under *Evaluate graph*. Move to tab *Evaluation*, and notice that all evaluation metrics are selected by default. Go back to *Main* tab and hit *Run*. **Figure 7.5** presents the information you should see in the terminal window of NetBeans.

Lastly, this process should have also generated four PDF files corresponding to each of the four options ticked under *Evaluate graph*. As before, the PDF files might be corrupted, or fail to generate and return an error, if you are running this on a MAC. The four files are:

- iv. *DAGtrue.pdf* in folder *Input*, as shown in **Figure 7.6a**.
- v. *CPDAGtrue.pdf* in folder *Input*, as shown in **Figure 7.6b**.
- vi. *DAGlearned.pdf* in folder *Output*, as shown in **Figure 7.7a**.
- vii. *CPDAGlearned.pdf* in folder *Output*, as shown in **Figure 7.7b**.

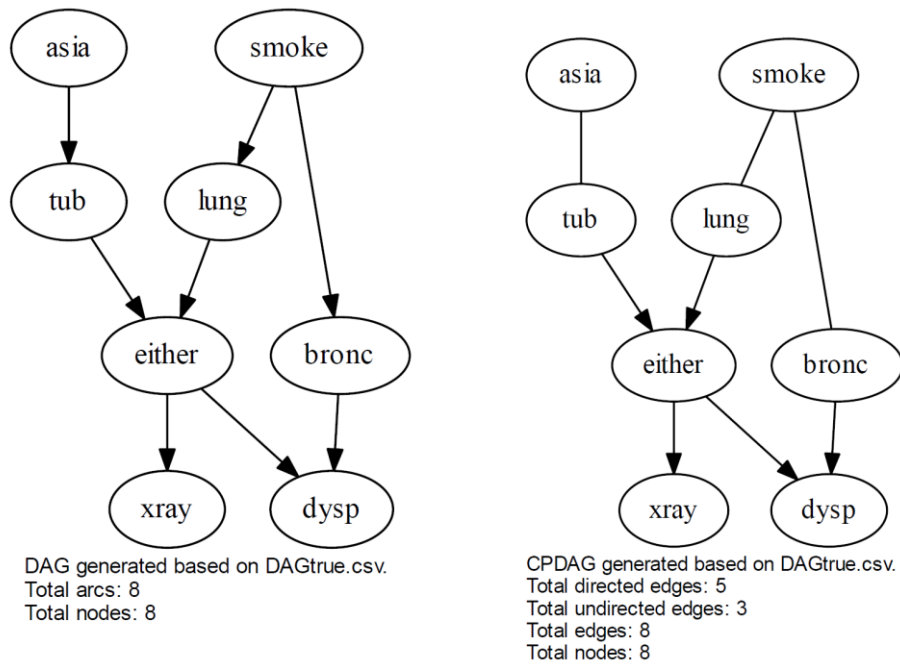
Notice that, in this case, the learned graphs are identical to the true graphs. This can occasionally happen (i.e., an algorithm would perform that well) when the network is simple and the input data are both sufficient and clean.

```

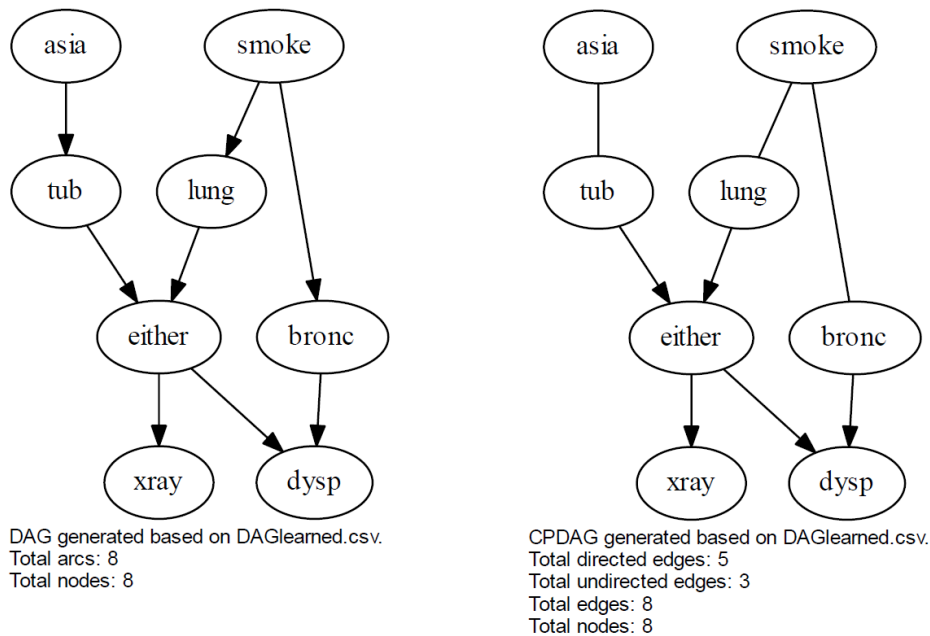
run:
_____ Evaluation _____
Nodes in DAGtrue: 8
Sample size in trainingData: 10000
Arcs in DAGtrue: 8
Direct independencies in DAGtrue: 20
Arcs in DAGlearned: 8
Direct independencies in DAGlearned: 20
Acyclic graph: true
_____ DAG Confusion Matrix _____
Arcs discovered (TP) [DAG]: 8.0
Partial arcs discovered (partial-TP) [DAG]: 0.0
False arcs discovered (FP) [DAG]: 0.0
Direct independencies discovered (TN) [DAG]: 20.0
Arcs not discovered (FN) [DAG]: 0.0. [NOTE: # of edges missed is 0.0]
_____ CPDAG Confusion Matrix _____
Arcs discovered (TP) [CPDAG]: 8.0
Partial arcs discovered (partial-TP) [CPDAG]: 0.0
False arcs discovered (FP) [CPDAG]: 0.0
Direct independencies discovered (TN) [CPDAG]: 20.0
Arcs not discovered (FN) [CPDAG]: 0.0. [NOTE: # of edges missed is 0.0]
_____ DAG metrics _____
Precision score [DAG]: 1.000
Recall score [DAG]: 1.000
F1 score [DAG]: 1.000
SHD score [DAG]: 0.000
DDM score [DAG]: 1.000
BSF score [DAG]: 1.000
# of independent graphical fragments: 1 (includes 0 single-state variables)
_____ CPDAG metrics _____
Precision score [CPDAG]: 1.000
Recall score [CPDAG]: 1.000
F1 score [CPDAG]: 1.000
SHD score [CPDAG]: 0.000
DDM score [CPDAG]: 1.000
BSF score [CPDAG]: 1.000
# of independent graphical fragments: 1 (includes 0 single-state variables)
_____ Inference-based evaluation _____
LL for graph [log2]: -32145.051
BIC score [log2] -32264.641
# of free parameters 18
BUILD SUCCESSFUL (total time: 51 seconds)

```

**Figure 7.5.** The output information generated in the terminal window of NetBeans after running the Evaluate graph method. The output is based on the ASIA 10k sample size example using Bayesys v3.3.



**Figure 7.6.** The true DAG (left) and true CPDAG (right) graphs generated in folder Input. The outputs are based on Bayesys v3.3.



**Figure 7.7.** The learned DAG (left) and learned CPDAG (right) graphs generated by the HC algorithm in folder Output, when applied to the ASIA networks with 10k sample size. The outputs are based on Bayesys v3.3.

### 7.3. Generate BN model in GeNIe

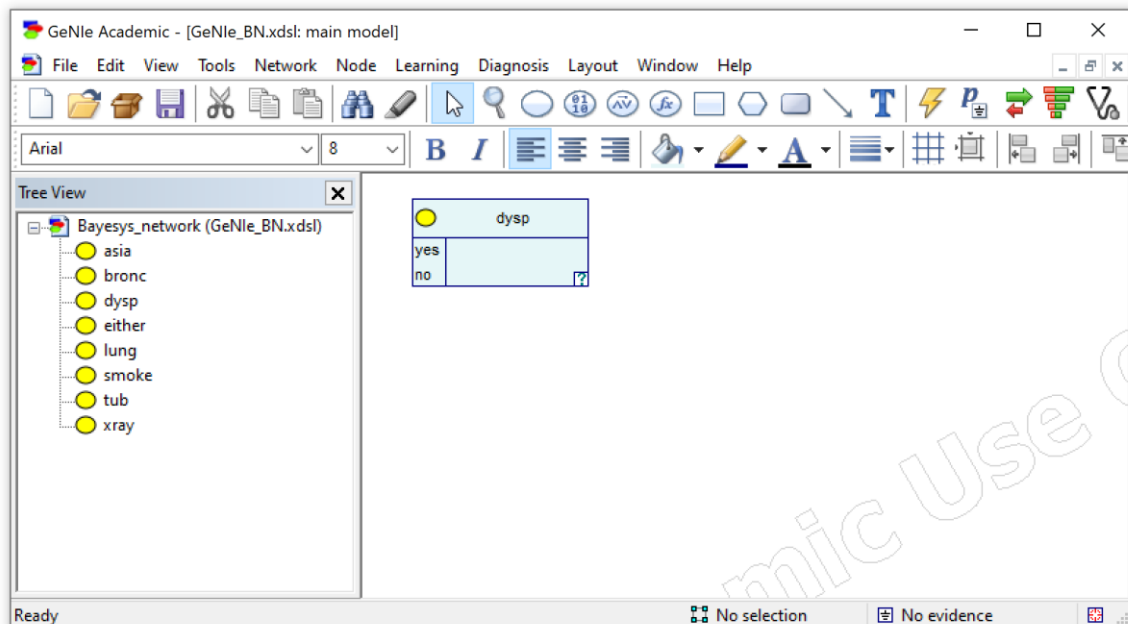
Rerun Bayesys and this time select *Generate BN model* under tab *Main*, then select *GeNIe* (default selection). As with *Evaluation*, this method can be performed with or without *Structure learning*, as long as the *trainingData.csv* and *DAGlearned.csv* files are present in folders *Input* and *Output* respectively. Click *Run*. Bayesys will then generate the BN model file *GeNIe\_BN.xdsl* in folder *Output*. **Figure 7.8** presents the relevant output information generated in the terminal window of NetBeans IDE.

```
run:
_____ Training data info _____
Variables: 8
Sample size: 10000
_____ Bayesian Network model _____
Creating BN model in GeNIe...
GeNIe XML model saved [Output/GeNIe_BN.xdsl].
BUILD SUCCESSFUL (total time: 3 seconds)
```

**Figure 7.8.** The output information generated in the terminal window of NetBeans IDE after running the Generate BN model method, with GeNIe set as the preferred BN file extension. This output is based on Bayesys v2.3.

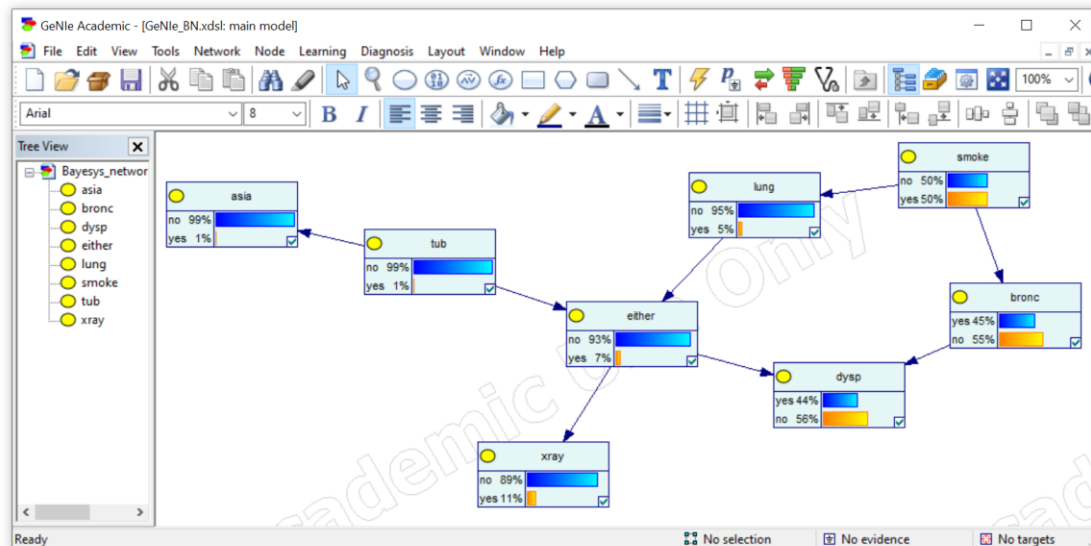
Once you load the learned BN file *GeNIe\_BN.xdsl* in GeNIe, you will be presented with what is shown in **Figure 7.9**, where the nodes are placed on top of each. To rearrange the nodes, click *Layout*, then *Graph Layout*, then *Parent Ordering* and click OK. Then, click *Layout*, then *Graph Layout*, then *Spring Embedder* and click OK. To show the probability distributions corresponding to each node, click *Update* (the yellow thunder icon on the top bar of UI). The result of these steps is shown in **Figure 7.10**. For further details, refer to the GeNIe modeller user manual [13].

For troubleshooting and things to know in relation to the GeNIe BN model generator implemented in Bayesys, refer to subsection 13.6.



**Figure 7.9.** The BN model we see when we first load *GeNIe\_BN.xdsl* in GeNIe. The nodes need to be arranged.





**Figure 7.10.** The BN model of **Figure 7.9** after arranging the nodes and making visible the probability distributions.

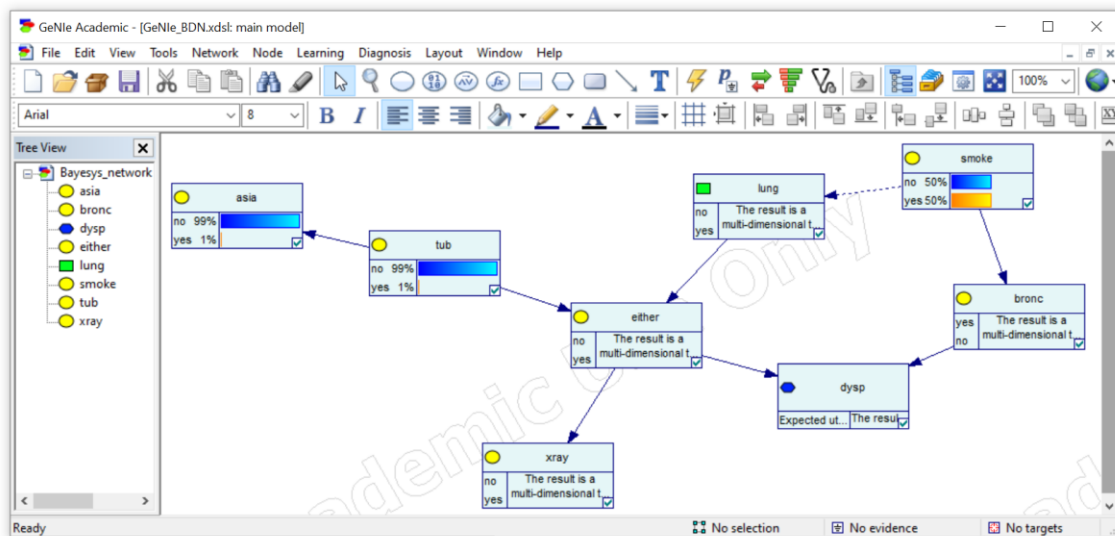
### 7.3.1. Generate BDN model in GeNIe

To generate a BDN, instead of a BN, repeat the steps described in subsection 7.3, but this time together with *Structure learning*, and select the knowledge approach *Decision network* under tab *Knowledge*. Before you do that, place the file *constraintsBDN.csv*, depicted in **Figure 7.11** and available in directory *Sample input files/Knowledge/BDNs*, in folder *Input*. The specified decision and utility nodes are selected purely for illustration purposes. Running Bayesys should now generate the file *GeNIe\_BDN.xdsl* (note the difference from BN to BDN in the filename).

	A	B	C	D	E
1	ID	Node	Node type	State to maximise	
2		1 lung	Decision	n/a	
3		2 dysp	Utility	no	

**Figure 7.11.** The *constraintsBDN.csv* file used as input for method *Decision network*.












**Figure 7.12** presents the BDN after loading *GeNIe\_BDN.xdsl* in GeNIe and rearranging the nodes. Observe that, compared to the graph depicted in **Figure 7.10**, a) the node *lung* has been converted into a decision node, b) the conditional arc from *smoke* entering *lung* has changed into an informational arc, and c) the node *dysp* has been converted into a utility node.



**Figure 7.12.** The BDN after loading *GeNIe\_BDN.xdsl* in GeNIe, converted from the BN model in **Figure 7.10** and with reference to the decision and utility nodes specified in **Figure 7.11**.

Lastly, **Figure 7.13** compares the Conditional Probability Table (CPT) of node *dysp* in the BN of **Figure 7.10**, to the CPT of utility node *dysp* in the BDN of **Figure 7.12**. Observe that the values in utility *dysp* correspond to the conditional probabilities of state “no” in chance node *dysp*, and this is because the state “no” is set as the maximisation state in *constraintsBDN.csv*, as it can be seen in **Figure 7.11**.

Node properties: dysp

General		Definition		Format		User properties		Value	
									
									
									

bronc		yes		no	
either		no	yes	no	yes
yes	0.80669415	0.88010899	0.10022997	0.75684932	
no	0.19330585	0.11989101	0.89977003	0.24315068	

(a) Chance node *dysp* in the Asia BN of **Figure 7.10**.

Node properties: dysp

General

Definition

Format

User properties

Value

bronc	yes		no	
either	no	yes	no	yes
▶ Value	0.19330585	0.11989101	0.89977003	0.24315068

(b) Utility node *dysp* in the Asia BDN of **Figure 7.12**.

**Figure 7.13.** (a) The CPT of chance node *dysp* in the BN of **Figure 7.10**; (b) the CPT of utility node *dysp* in the BDN of **Figure 7.12**, where state “no” is set as the maximisation value as shown in **Figure 7.11**.

For troubleshooting and things to know regarding the GeNIe BDN model generator implemented in Bayesys, refer to subsection 13.6.

#### 7.4. Generate BN model in AgenaRisk

*Note: The AgenaRisk functions are no longer compatible with the new version of AgenaRisk which requires active license and internet connection.*

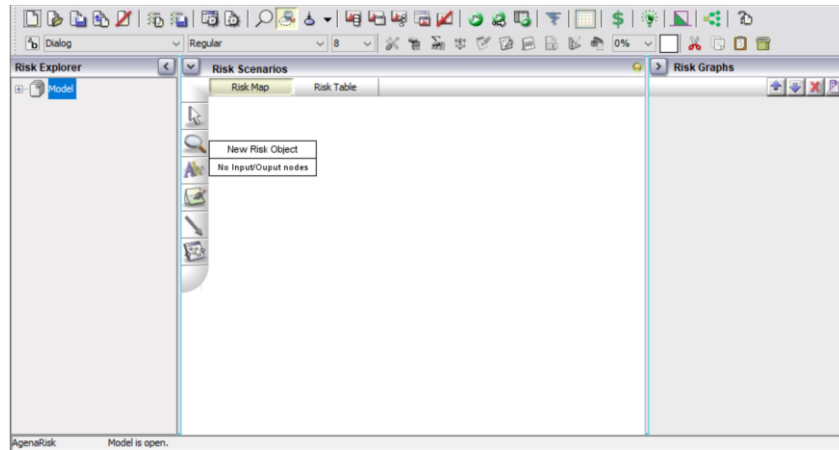
Rerun Bayesys and select *Generate BN model* under tab *Main*, and then unselect the default selection *GeNIe* and select *AgenaRisk* instead. As with *Evaluation*, this method can be performed with or without *Structure learning*, as long as the *trainingData.csv* and *DAGlearned.csv* files are present in folders *Input* and *Output* respectively. Click *Run*. Bayesys will then generate the BN model file *AgenaRisk\_BN.cmd* in folder *Output*. **Figure 7.14** presents the relevant output information generated in the terminal window of NetBeans IDE.

```
run:
_____ Training data info _____
Variables: 8
Sample size: 10000
_____ Bayesian Network model _____
Creating BN model in AgenaRisk...
GUI not initiated
Headless mode: false
AgenaRisk 10 Revision 6120
API professional license files found.
BN model parameterisation progress at 10%
BN model parameterisation progress at 20%
BN model parameterisation progress at 30%
BN model parameterisation progress at 40%
BN model parameterisation progress at 50%
BN model parameterisation progress at 60%
BN model parameterisation progress at 70%
Parameterisation completed.
BN model created and saved [Output/AgenaRisk_BN.cmd].
BUILD SUCCESSFUL (total time: 7 seconds)
```

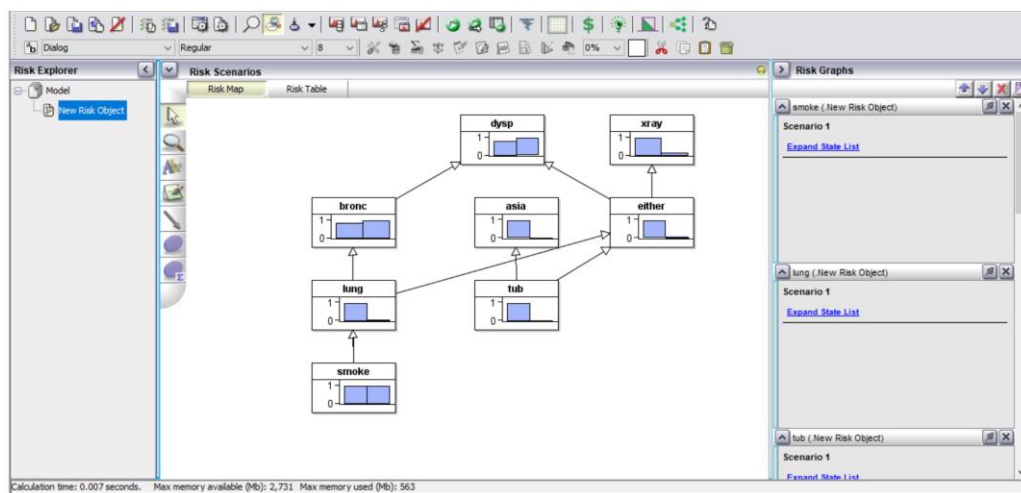
**Figure 7.14.** The output information generated in the terminal window of NetBeans after running the Generate BN model method, with AgenaRisk set as the preferred BN file extension. This output is based on Bayesys v2.3.

Once you load the *AgenaRisk\_BN.cmd* in AgenaRisk, you will be presented with what is shown in **Figure 7.15**. To view the learned BN model and its probability distributions, click and expand *Model* which can be found within the *Risk Explorer* window on the left, and then click on *New Risk Object* (as can be seen in **Figure 7.15**). Then, click on the *Run calculation* button (green play button in UI) to update the distributions. To view the distributions, use your mouse cursor to select all nodes, and double click on one of the nodes. The result of these steps is shown in **Figure 7.16**.

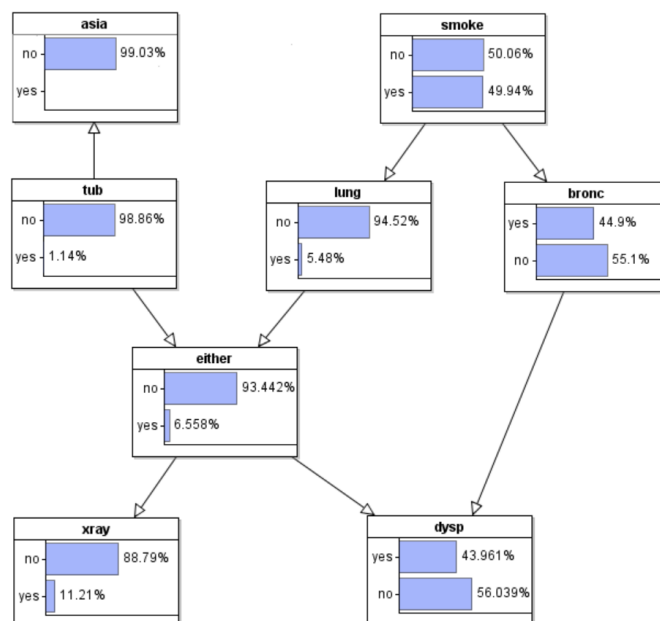
Lastly, **Figure 7.17** presents the graph shown after rearranging the nodes, modifying the size of the nodes, and changing the type of the probability distributions from vertical to horizontal. For further details, refer to the AgenaRisk user manual that comes with the AgenaRisk Desktop version.



**Figure 7.15.** The BN mode we see once we load *AgenaRisk\_BN.cmp* in AgenaRisk. The nodes need to be made visible.



**Figure 7.16.** The BN model of Fig 6.15 after using Risk Explorer to view the nodes.



**Figure 7.17.** The BN model of Fig 6.16 after rearranging the nodes, modifying the size of the nodes, and changing the type of the probability distributions shown for each node.

## 8. Worked examples: Structure learning with knowledge-based constraints

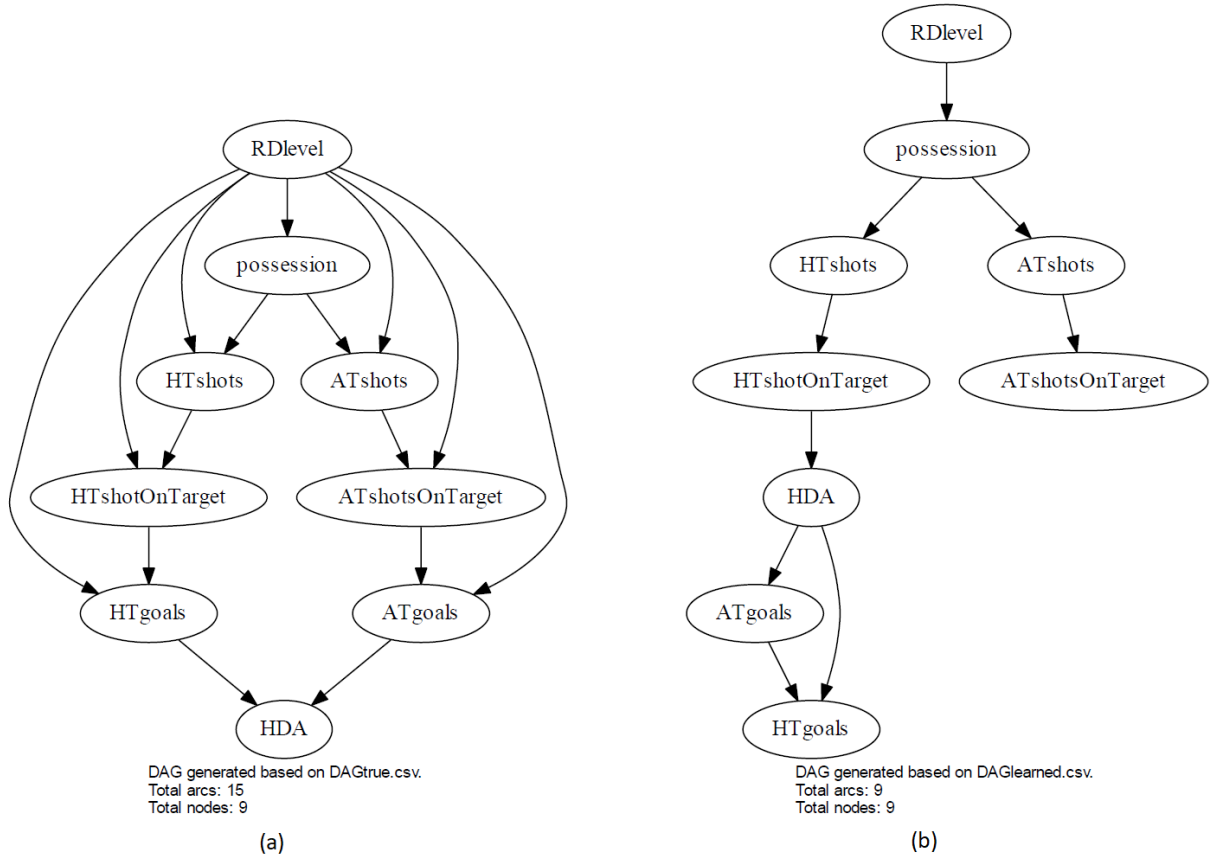
Ensure you have read Section 4 and went through Section 7 before going through this section. The worked examples presented in this section are based on the SPORTS network. The data used to illustrate these examples can be found in project directory *Sample input files/Worked SPORTS example from the manual*. To repeat the examples, copy all those files in folder *Input*.

NOTE: The knowledge constraints selected here are purely for illustration purposes. The aim is to try to ‘correct’ the learned graphs by selecting constraints that are present in the true graph but absent in the learned graph.

### Example 1: No knowledge

Run Bayesys with HC selected as the structure learning algorithm (default selection). Tick *Evaluate graph* as well its third subprocess called *Save learned DAG...*. **Figure 8.1a** shows the true DAG of SPORTS whereas **Figure 8.1b** shows the DAG learned by HC with 1,000 samples, before incorporating any knowledge. Notice that the F1 score under *CPDAG metrics* is 0.5, since the learned graph has missed many edges that are present in the true graph.

All subsequent examples incorporate knowledge-based constraints, and are discussed with reference to the graph in **Figure 8.1b** which is learned from data.



**Figure 8.1.** The a) true DAG of the SPORTS network and b) the DAG learned by HC with 1,000 samples before incorporating any knowledge. The outputs are based on Bayesys v3.3 (Note: output may be slightly different in v3.6 or later).

## Example 2: Directed edge constraints

Repeat the learning process as in **Example 1**, but this time also select *Directed* under tab *Knowledge*. The learning process will now consider the directed edge constraints specified in *constraintsDirected.csv*. These are *HTshotOnTarget*→*HTgoals* and *ATshotsOnTarget*→*ATgoals*, and they represent two edges that are present in the true graph but absent in the learned graph shown in **Figure 8.1b**. Notice that the terminal window in NetBeans indicates the number of constraints taken into consideration during the structure learning process; i.e., in this example, it should state:

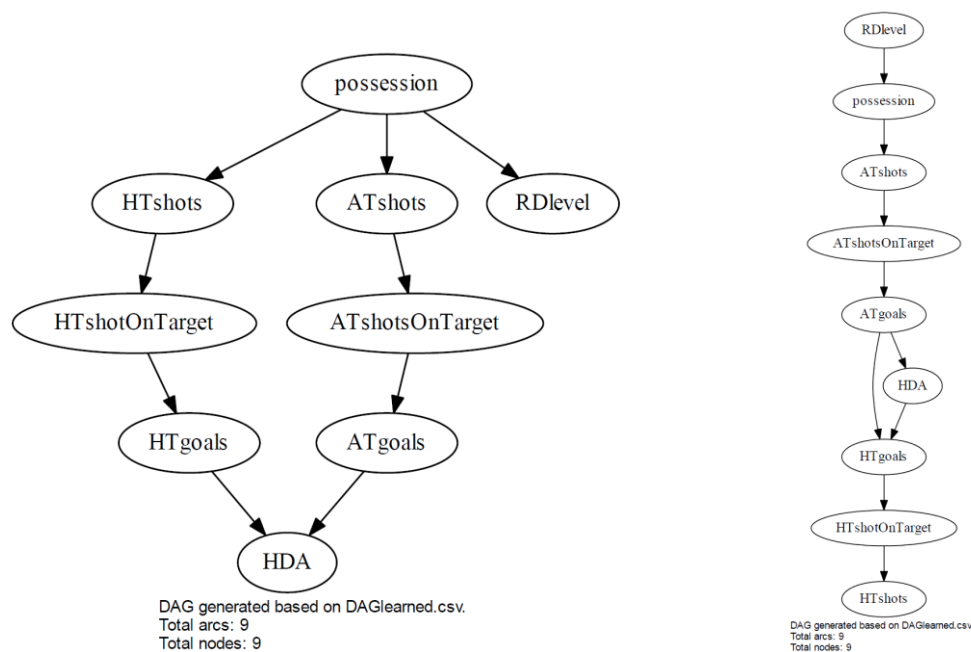
\_\_\_\_\_ Knowledge-based constraints \_\_\_\_\_  
Directed edge constraints specified: 2

**Figure 8.2a** presents the graph learned at this step. Notice that while the constraints have been successfully imposed, they have also caused modifications to other parts of the graph. The revised graph now produces an F1 score of 0.75 for the CPDAG output, up from 0.5 without constraints, and this suggests that the knowledge constraints have helped the algorithm to discover a graph that is closer to the true graph (unsurprisingly, since we know the constraints are correct).

## Example 3: Undirected edge constraints

Repeat the same process and this time select *Undirected* (instead of *Directed*) under tab *Knowledge*. The learning process will now consider the undirected edge constraints specified in *constraintsUndirected.csv*. The constraints are the same as in **Example 2**, but they are now imposed as undirected, rather than directed, edges; i.e., *HTshotOnTarget*–*HTgoals* and *ATshotsOnTarget*–*ATgoals*.

**Figure 8.2b** presents the graph learned at this step. This graph produces an F1 score of 0.583 for the CPDAG output, up from 0.5 without constraints, but lower than 0.75 which was achieved in **Example 2** with the same edges imposed as directed constraints.



**Figure 8.2.** The SPORTS network DAGs learned by HC with directed (left) and undirected (right) constraints. The outputs are based on Bayesys v3.3 (Note: output may be slightly different in v3.6 or later).

#### Example 4: Forbidden edge constraints

Repeat the same process and this time select *Forbidden* under tab *Knowledge*. The learning process will now consider the forbidden edge constraints specified in *constraintsForbidden.csv*. These are *HTshotOnTarget*→*HDA* and *HTgoals*→*ATgoals*, since those two edges are present in the learned graph but not in the true graph.

**Figure 8.3a** presents the graph learned at this step. Notice that this time the constraints were enforced with minor effect to other parts of the graph. The F1 score for the CPDAG has increased (relative to **Figure 8.1b**) from 0.5 to 0.522.

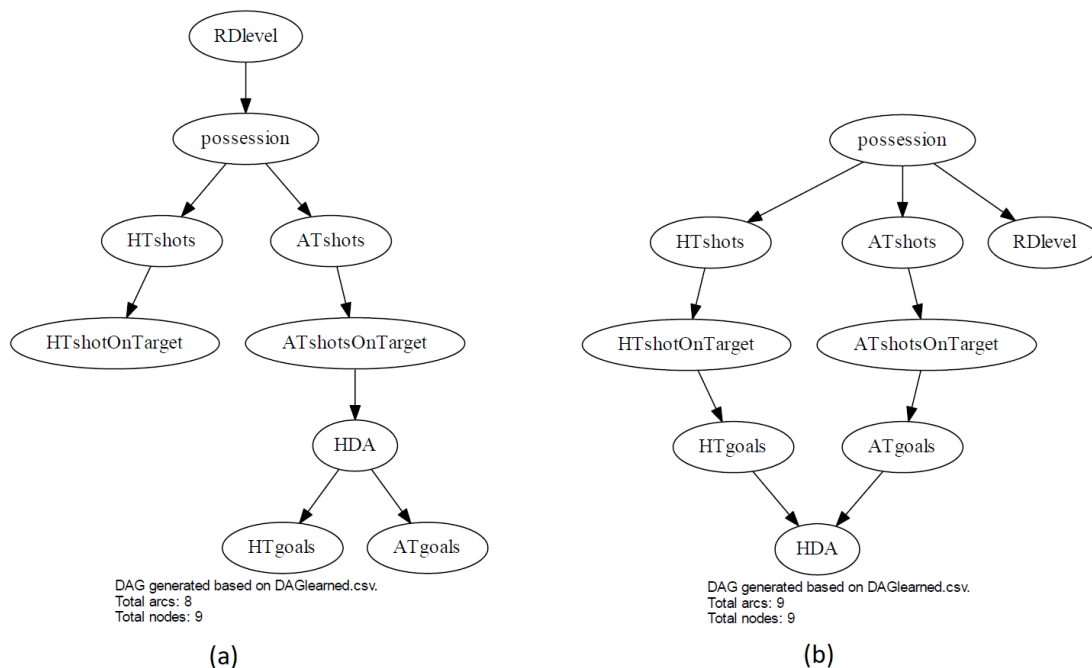
#### Example 5: Temporal constraints

Repeat the same process and this time select *Temporal* together with its subprocess *Prohibit edges between variables of the same tier*, under tab *Knowledge*. The learning process will now consider the temporal constraints specified in *constraintsTemporal.csv*. These are depicted in **Table 8.1**, and aim to correct the orientations between *HDA* and *ATgoals* and *HTgoals* variables, but also to eliminate the edge between the latter two variables consistent with the true graph.

**Figure 8.3b** presents the graph learned at this step. The output is identical (by chance) to that of **Example 2** which involved directed edge constraints. Notice how the graph has been modified to satisfy the parental and ancestral relationships consistent with the temporal orderings described in **Table 8.1**. The nodes *HTgoals* and *ATgoals* are now unconnected since they are part of the same tier in **Table 8.1**; i.e., recall we have selected *to Prohibit edges between variables of the same tier*.

**Table 8.1.** The temporal constraints considered in Example 5.

ID	Tier 1	Tier 2
1	ATgoals	HDA
2	HTgoals	



**Figure 8.3.** The SPORTS network DAGs learned by HC with a) forbidden and b) temporal constraints. The outputs are based on Bayesys v3.3 (Note: output may be slightly different in v3.6 or later).



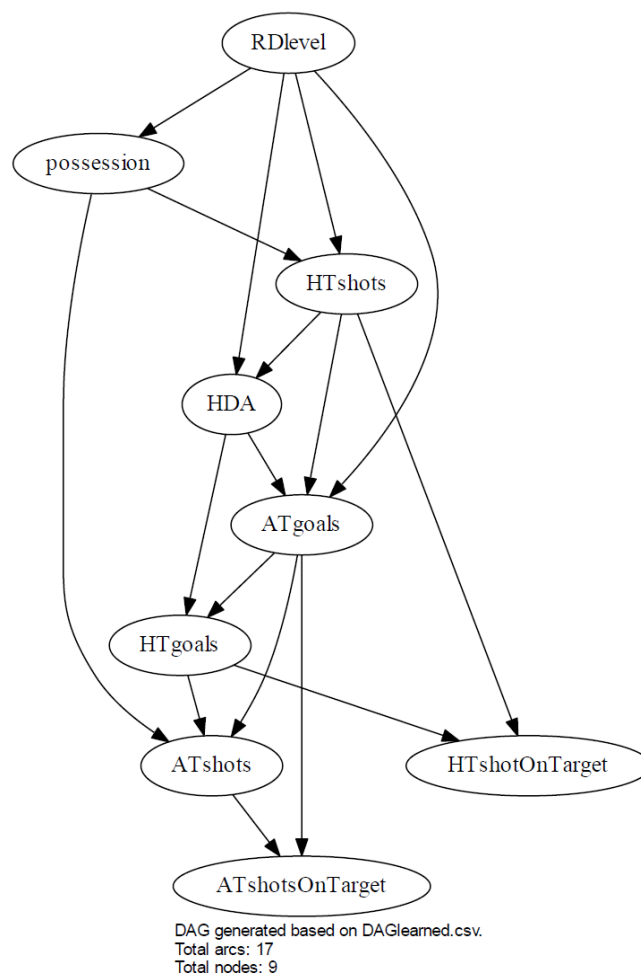
### Example 6: Initial graph

Repeat the same process and this time select *Initial graph* under tab *Knowledge*. The learning process will now consider the edges specified in *constraintsGraph.csv* as being present in the initial graph from which the algorithm will start exploring the search space of DAGs. The constraints considered in this example are the same two edges as in **Example 2** and **Example 3**, and the effect on the learned graph appears to be the same as in **Example 2** and **Example 5** (but this will not always be the case, especially in larger networks).

### Example 7: Target node

Repeat the previous process and this time select *Target nodes* with the *Penalty reduction* ( $r$ ) hyperparameter set to 10, under tab *Knowledge*. Increasing the *Penalty reduction* ( $r$ ) parameter makes it more likely to draw more edges entering targeted nodes. The learning process will now consider the nodes specified in *constraintsTarget.csv* as nodes targeted for relaxed dimensionality penalty, to encourage the algorithm to discover a higher number of parents for those nodes. The seven nodes targeted are those that have two parents in the true graph shown in **Figure 8.1a**, since the algorithm tends to return just one parent for almost all those nodes.

**Figure 8.4** presents the graph learned at this step. Notice how the number of edges has increased. However, not all the newly added edges are correct. Overall, the F1 score for the learned CPDAG has increased from 0.5 to 0.563.



**Figure 8.4.** The SPORTS network DAG learned by HC with targeted node constraints. The output is based on Bayesys v3.3 (Note: output may be slightly different in v3.6 or later).



## 9. Worked example: Performing multiple structure learning experiments

Often, we need to run multiple algorithms on a single data set, apply an algorithm to multiple data sets, or a combination of both. The steps enumerated below illustrate how to do this in Bayesys.

### STEP 1: Prepare the necessary files

For this illustration, we will be using the sample files available in directory *Sample input files/Multiple inputs*. The folder contains the following files:

- Five true graphs, called *DAGtrue\_ALARM.csv*, *DAGtrue\_ASIA.csv*, etc,
- Ten data sets, two for each of the five true graphs, with sample sizes  $10^3$  and  $10^4$ , called *trainingData\_ALARM\_1k.csv*, *trainingData\_ALARM\_10k.csv*, etc,
- A file containing the details of the experiments, called *settings.csv*. This file contains 40 experiments. **Figure 9.1** presents the first 14 experiments, where the first column specifies the algorithm that should be used for structure learning, the second column the data set, and the third column the true graph. The remaining columns specify whether any knowledge-based constraints are to be considered for each knowledge approach, where each knowledge approach requires its own data set (refer to Section 4). In this illustration, we will not be considering the knowledge approaches, which is why all relevant values are set to *FALSE* in **Figure 9.1**.

Copy all the above files in the following input directory *Input/Multiple inputs*.

	A	B	C	D	E	F	G	H	I	J
	Algorithm	trainingDataFileName	DAGtrueFileName	directedConstraintsFileName	undirectedConstraintsFileName	forbiddenConstraintsFileName	temporalConstraintsFileName	strictTemporal?	initialGraphConstraintsFileName	varRelevantConstraints?
1	HC	trainingData_ASIA_1k	DAGtrue_ASIA	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
2	HC	trainingData_ASIA_10k	DAGtrue_ASIA	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
3	HC	trainingData_SPORTS_1k	DAGtrue_SPORTS	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
4	HC	trainingData_SPORTS_10k	DAGtrue_SPORTS	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
5	HC	trainingData_PROPERTY_1	DAGtrue_PROPERTY	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
6	HC	trainingData_PROPERTY_10	DAGtrue_PROPERTY	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
7	HC	trainingData_ALARM_1k	DAGtrue_ALARM	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
8	HC	trainingData_ALARM_10k	DAGtrue_ALARM	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
9	HC	trainingData_FORMED_1k	DAGtrue_FORMED	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
10	HC	trainingData_FORMED_10k	DAGtrue_FORMED	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
11	TABU	trainingData_ASIA_1k	DAGtrue_ASIA	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
12	TABU	trainingData_ASIA_10k	DAGtrue_ASIA	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
13	TABU	trainingData_SPORTS_1k	DAGtrue_SPORTS	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
14	TABU	trainingData_SPORTS_10k	DAGtrue_SPORTS	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
15	TABU	trainingData_PROPERTY_1	DAGtrue_PROPERTY	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
16	TABU	trainingData_PROPERTY_10	DAGtrue_PROPERTY	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
17	TABU	trainingData_ALARM_1k	DAGtrue_ALARM	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
18	TABU	trainingData_ALARM_10k	DAGtrue_ALARM	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
19	TABU	trainingData_FORMED_1k	DAGtrue_FORMED	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
20	TABU	trainingData_FORMED_10k	DAGtrue_FORMED	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
21	SaiyanH	trainingData_ASIA_1k	DAGtrue_ASIA	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
22	SaiyanH	trainingData_ASIA_10k	DAGtrue_ASIA	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
23	SaiyanH	trainingData_SPORTS_1k	DAGtrue_SPORTS	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
24	SaiyanH	trainingData_SPORTS_10k	DAGtrue_SPORTS	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

**Figure 9.1.** Part of the contents of the sample input file *settings.csv*. This input file is based on release Bayesys v3.3 (Note: output may be slightly different in v3.6 or later).

## STEP 2: Running Bayesys

Run Bayesys. Under tab *Main*, click on *Structure learning* and select *Multiple algorithms and/or data sets* using the dropdown menu next to *Algorithm*. Hit run.

Bayesys will spend around five minutes executing the 40 structure learning experiments specified in *settings.csv*. We can view the progress of structure learning by referring to the terminal window of NetBeans IDE. **Figure 9.2a** and **Figure 9.2b** present the output information generated for the first two, out of 40, experiments. The output information is the same as that generated when performing structure learning manually with one algorithm and data set at a time. The only additional information shown during this process involves the number of experiments, which are highlighted in **Figure 9.2**.

```
run:
----- Settings data info -----
Experiments to run: 40
-----
Entering experiment 1/40
-----
Training data info
Variables: 8
Sample size: 1000
-----
Knowledge-based constraints
Directed edge constraints specified: 0
Temporal variables specified: 0 over 0 tiers.
Prohibit edges between variables of the same tier: false
Undirected edge constraints specified: 0
Forbidden edges specified: 0
Input graph relationships specified: 0
Target variables specified: 0.
Decision nodes specified: 0.
Utility nodes specified: 0.
-----
Structure learning
Running HC with settings:
  a) Initial graph: Empty
  b) BIC log: 2
Entering score-based learning HC search...
HC search completed.
Structure learning elapsed time: 0 seconds.
-----
Evaluation
Nodes in DAGtrue: 8
Sample size in trainingData: 1000
Arcs in DAGtrue: 8
Direct independencies in DAGtrue: 20
Arcs in DAGlearned: 8
Direct independencies in DAGlearned: 20
Acyclic graph: true
-----
DAG Confusion Matrix
Arcs discovered (TP) [DAG]: 3.0
Partial arcs discovered (partial-TP) [DAG]: 3.0
False arcs discovered (FP) [DAG]: 2.0
Direct independencies discovered (TN) [DAG]: 18.0
Arcs not discovered (FN) [DAG]: 3.5. [NOTE: # of edges missed is 2.0]
-----
CPDAG Confusion Matrix
Arcs discovered (TP) [CPDAG]: 5.0
Partial arcs discovered (partial-TP) [CPDAG]: 1.0
False arcs discovered (FP) [CPDAG]: 2.0
Direct independencies discovered (TN) [CPDAG]: 18.0
Arcs not discovered (FN) [CPDAG]: 2.5. [NOTE: # of edges missed is 2.0]
-----
DAG metrics
Precision score [DAG]: 0.563
Recall score [DAG]: 0.563
F1 score [DAG]: 0.563
SHD score [DAG]: 5.500
DDM score [DAG]: -0.125
BSF score [DAG]: 0.462
# of independent graphical fragments: 2 (includes 0 single-state variables)
-----
CPDAG metrics
Precision score [CPDAG]: 0.688
Recall score [CPDAG]: 0.688
F1 score [CPDAG]: 0.688
SHD score [CPDAG]: 4.500
DDM score [CPDAG]: 0.125
BSF score [CPDAG]: 0.587
# of independent graphical fragments: 2 (includes 0 single-state variables)
-----
Inference-based evaluation
LL for graph [log2]: -3173.431
BIC score [log2]: -3268.106
# of free parameters 19
```

(a)

```
-----
Entering experiment 2/40
-----
Training data info
Variables: 8
Sample size: 10000
-----
Knowledge-based constraints
Directed edge constraints specified: 0
Temporal variables specified: 0 over 0 tiers.
Prohibit edges between variables of the same tier: false
Undirected edge constraints specified: 0
Forbidden edges specified: 0
Input graph relationships specified: 0
Target variables specified: 0.
Decision nodes specified: 0.
Utility nodes specified: 0.
-----
Structure learning
Running HC with settings:
  a) Initial graph: Empty
  b) BIC log: 2
Entering score-based learning HC search...
HC search completed.
Structure learning elapsed time: 0 seconds.
-----
Evaluation
Nodes in DAGtrue: 8
Sample size in trainingData: 10000
Arcs in DAGtrue: 8
Direct independencies in DAGtrue: 20
Arcs in DAGlearned: 8
Direct independencies in DAGlearned: 20
Acyclic graph: true
-----
DAG Confusion Matrix
Arcs discovered (TP) [DAG]: 8.0
Partial arcs discovered (partial-TP) [DAG]: 0.0
False arcs discovered (FP) [DAG]: 0.0
Direct independencies discovered (TN) [DAG]: 20.0
Arcs not discovered (FN) [DAG]: 0.0. [NOTE: # of edges missed is 0.0]
-----
CPDAG Confusion Matrix
Arcs discovered (TP) [CPDAG]: 8.0
Partial arcs discovered (partial-TP) [CPDAG]: 0.0
False arcs discovered (FP) [CPDAG]: 0.0
Direct independencies discovered (TN) [CPDAG]: 20.0
Arcs not discovered (FN) [CPDAG]: 0.0. [NOTE: # of edges missed is 0.0]
-----
DAG metrics
Precision score [DAG]: 1.000
Recall score [DAG]: 1.000
F1 score [DAG]: 1.000
SHD score [DAG]: 0.000
DDM score [DAG]: 1.000
BSF score [DAG]: 1.000
# of independent graphical fragments: 1 (includes 0 single-state variables)
-----
CPDAG metrics
Precision score [CPDAG]: 1.000
Recall score [CPDAG]: 1.000
F1 score [CPDAG]: 1.000
SHD score [CPDAG]: 0.000
DDM score [CPDAG]: 1.000
BSF score [CPDAG]: 1.000
# of independent graphical fragments: 1 (includes 0 single-state variables)
-----
Inference-based evaluation
LL for graph [log2]: -32145.051
BIC score [log2]: -32264.641
# of free parameters 18
```

(b)

**Figure 9.2.** The output information generated in the terminal window of NetBeans IDE regarding the first two, out of the 40, experiments specified in file *settings.csv*. The outputs are based on Bayesys v3.2 (Note: output may be slightly different in v3.6 or later).

### STEP 3: Reading the results

- The results are generated in the file called *resultsMultiInput.csv*, and this file is generated in folder *Output*.
- The file *resultsMultiInput.csv* is updated after each experiment is completed. In this example, there are 40 experiments. Therefore, the output file *resultsMultiInput.csv* will be updated 40 times.
- If you are running experiments that take hours to complete and would like to view the contents of *resultsMultiInput.csv* during structure learning, you should create a copy of this file and view the copied version. Do *not* open the *resultsMultiInput.csv* file during the structure learning process, as this might cause the system to return an error.
- Because *resultsMultiInput.csv* is updated iteratively, we can stop structure learning whenever we wish and save the results generated up to that point; e.g., by creating a copy of *resultsMultiInput.csv*. We can then remove the completed experiments from the *settings.csv* file, so that we do not repeat those experiments next time we resume structure learning.
- **Figure 9.3** presents the results generated in file *resultsMultiInput.csv*. The first column lists the experiment number, and the columns *B* to *K* are copies of the corresponding columns specified in *settings.csv*. The remaining columns provide all the results we can possibly need, and these include the graphical scores for both DAGs (columns *N* to *R*) and CPDAGs (columns *AB* to *AF*).

Note that columns *U* to *Y* provide pruning information that is only relevant to the MAHC algorithm. As shown in **Figure 9.3**, these values are set to n/a whenever a different algorithm is used for a particular experiment.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH
1	Experiment	Algorithm	trainingDa	sampleSize	DAGtrue	directedC	undirectedC	forbidden	temporalC	strictTemp	initialGraph	varRelevance	#freeParams	BIC	BSF-DAG	SHD-DAG	F1-DAG	Recall-DAC	Precision	Edges learnt	Runtime	Pruning rule	Structure	Level 1 pruned	Level 2 pruned	Level 3 pruned	indepGraph	Acyclic	BSF-CPDAG	SHD-CPDAG	F1-CPDAG	Recall-CPC	Precision-CPDAG	
2	1	HC	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	19	-3268.1	0.462	5.5	0.563	0.563	0.563	8	0	n/a	n/a	n/a	n/a	n/a	2	TRUE	0.587	4.5	0.688	0.688	0.688	
3	2	HC	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	18	-32265	1	0	1	1	1	8	0	n/a	n/a	nul	nul	nul	1	TRUE	1	0	1	1	1	
4	3	HC	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	201	-16909	0.305	11	0.5	0.4	0.667	9	0	n/a	n/a	nul	nul	nul	1	TRUE	0.305	11	0.5	0.4	0.667	
5	4	HC	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	209	-158259	0.6	6	0.75	0.6	1	9	0	n/a	n/a	nul	nul	nul	1	TRUE	0.6	6	0.75	0.6	1	
6	5	HC	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	955	-39697	0.484	20.5	0.544	0.5	0.596	26	0	n/a	n/a	nul	nul	nul	5	TRUE	0.452	21.5	0.509	0.468	0.558	
7	6	HC	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	1647	-347821	0.681	13.5	0.705	0.694	0.717	30	0	n/a	n/a	nul	nul	nul	3	TRUE	0.697	13	0.721	0.71	0.733	
8	7	HC	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	377	-17076	0.636	26	0.652	0.652	0.652	46	0	n/a	n/a	nul	nul	nul	3	TRUE	0.571	29	0.587	0.587	0.587	
9	8	HC	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	612	-154627	0.779	18.5	0.753	0.793	0.716	51	0	n/a	n/a	nul	nul	nul	2	TRUE	0.725	21	0.701	0.739	0.667	
10	9	HC	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	487	-63238	0.456	104	0.5	0.464	0.542	118	3	n/a	n/a	nul	nul	nul	5	TRUE	0.463	103	0.508	0.471	0.551	
11	10	HC	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	900	-608946	0.565	111	0.539	0.58	0.503	159	10	n/a	n/a	nul	nul	nul	1	TRUE	0.565	111	0.539	0.58	0.503	
12	11	TABU	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	19	-3268.1	0.462	5.5	0.563	0.563	0.563	8	0	n/a	n/a	n/a	n/a	n/a	2	TRUE	0.587	4.5	0.688	0.688	0.688	
13	12	TABU	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	18	-32265	1	0	1	1	1	8	0	n/a	n/a	n/a	n/a	n/a	1	TRUE	1	0	1	1	1	
14	13	TABU	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	201	-16909	0.305	11	0.5	0.4	0.667	9	0	n/a	n/a	n/a	n/a	n/a	1	TRUE	0.305	11	0.5	0.4	0.667	
15	14	TABU	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	209	-158259	0.6	6	0.75	0.6	1	9	0	n/a	n/a	n/a	n/a	n/a	1	TRUE	0.6	6	0.75	0.6	1	
16	15	TABU	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	955	-39697	0.484	20.5	0.544	0.5	0.596	26	0	n/a	n/a	n/a	n/a	n/a	5	TRUE	0.452	21.5	0.509	0.468	0.558	
17	16	TABU	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	1647	-347821	0.665	14	0.689	0.677	0.7	30	0	n/a	n/a	n/a	n/a	n/a	3	TRUE	0.697	13	0.721	0.71	0.733	
18	17	TABU	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	369	-17025	0.638	25	0.659	0.652	0.667	45	0	n/a	n/a	n/a	n/a	n/a	3	TRUE	0.583	27.5	0.604	0.598	0.611	
19	18	TABU	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	612	-154627	0.779	18.5	0.753	0.793	0.716	51	1	n/a	n/a	n/a	n/a	n/a	2	TRUE	0.725	21	0.701	0.739	0.667	
20	19	TABU	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	489	-63232	0.451	106.5	0.492	0.46	0.529	120	13	n/a	n/a	n/a	n/a	n/a	5	TRUE	0.444	107.5	0.484	0.453	0.521	
21	20	TABU	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	900	-608946	0.565	111	0.539	0.58	0.503	159	21	n/a	n/a	n/a	n/a	n/a	1	TRUE	0.565	111	0.539	0.58	0.503	
22	21	SaiyanH	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	18	-3261.4	0.763	2.5	0.813	0.813	0.813	8	0	n/a	n/a	n/a	n/a	n/a	1	TRUE	0.825	2	0.875	0.875	0.875	
23	22	SaiyanH	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	18	-32265	0.938	0.5	0.938	0.938	0.938	8	0	n/a	n/a	n/a	n/a	n/a	1	TRUE	1	0	1	1	1	
24	23	SaiyanH	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	209	-16698	0.5	7.5	0.625	0.5	0.833	9	0	n/a	n/a	n/a	n/a	n/a	1	TRUE	0.6	6	0.75	0.6	1	
25	24	SaiyanH	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	279	-158147	0.419	9	0.56	0.467	0.7	10	0	n/a	n/a	n/a	n/a	n/a	1	TRUE	0.519	7.5	0.68	0.567	0.85	
26	25	SaiyanH	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	1001	-40963	0.572	20.5	0.627	0.597	0.661	28	0	n/a	n/a	n/a	n/a	n/a	1	TRUE	0.556	21	0.61	0.581	0.643	
27	26	SaiyanH	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	1457	-358547	0.771	8	0.814	0.774	0.857	28	5	n/a	n/a	n/a	n/a	n/a	1	TRUE	0.755	8.5	0.797	0.758	0.839	
28	27	SaiyanH	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	366	-16883	0.74	17.5	0.767	0.75	0.784	44	1	n/a	n/a	n/a	n/a	n/a	1	TRUE	0.708	19	0.733	0.717	0.75	
29	28	SaiyanH	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	455	-153173	0.888	7	0.901	0.891	0.911	45	7	n/a	n/a	n/a	n/a	n/a	1	TRUE	0.91	6	0.923	0.913	0.933	
30	29	SaiyanH	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	453	-63531	0.508	79.5	0.597	0.511	0.719	98	22	n/a	n/a	n/a	n/a	n/a	1	TRUE	0.522	77.5	0.614	0.525	0.74	
31	30	SaiyanH	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	549	-615399	0.637	52	0.724	0.638	0.838	105	124	n/a	n/a	n/a	n/a	n/a	1	TRUE	0.641	51.5	0.728	0.641	0.843	
32	31	MAHC	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	17	-3259.2	0.638	3.5	0.733	0.688	0.786	7	0	0	0	46.429	23.214	10.714	2	TRUE	0.7	3	0.8	0.75	0.857	
33	32	MAHC	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	19	-32286	0.813	1.5	0.813	0.813	0.813	8	0	0	0	32.143	23.214	16.071	1	TRUE	0.875	1	0.875	0.875	0.875	
34	33	MAHC	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	145	-17115	0.3	10.5	0.45	0.3	0.9	5	0	0	0	50	36.111	0	4	TRUE	0.333	10	0.5	0.333	1	
35	34	MAHC	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	209	-158259	0.567	6.5	0.708	0.567	0.944	9	0	0	0	5.556	59.722	9.722	1	TRUE	0.6	6	0.75	0.6	1	
36	35	MAHC	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	872	-40432	0.539	17	0.654	0.548	0.81	21	0	0	0	82.051	12.108	1.709	6	TRUE	0.474	19	0.577	0.484	0.714	
37	36	MAHC	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	1417	-355305	0.752	9.5	0.81	0.758	0.87	27	0	0	0	78.917	10.541	4.131	4	TRUE	0.72	10.5	0.776	0.726	0.833	
38	37	MAHC	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	333	-17659	0.622	22	0.682	0.63	0.744	39	0	0	0	78.979	13.964	2.853	5	TRUE	0.557	25	0.612	0.565	0.667	
39	38	MAHC	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	446	-154382	0.787	13.5	0.802	0.793	0.811	45	2	1	0	71.922	14.489	6.907	3	TRUE	0.711	17	0.725	0.717	0.733	
40	39	MAHC	trainingDa	1000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	427	-64039	0.525	68.5	0.642	0.525	0.824	88	10	0	9	92.659	4.467	0.954	8	TRUE	0.506	71	0.619	0.507	0.795	
41	40	MAHC	trainingDa	10000	DAGtrue_	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	609	-610503	0.717	41	0.789	0.717	0.876	113	48	5	43	87.539	6.883	2.181	1	TRUE	0.731	39	0.805	0.732	0.894	

**Figure 9.3.** The results generated in output file resultsMultilInput.csv with reference to the 40 experiments specified in file settings.csv. The output is based on Bayesys v3.3 (Note: output may be slightly different in v3.6 or later).

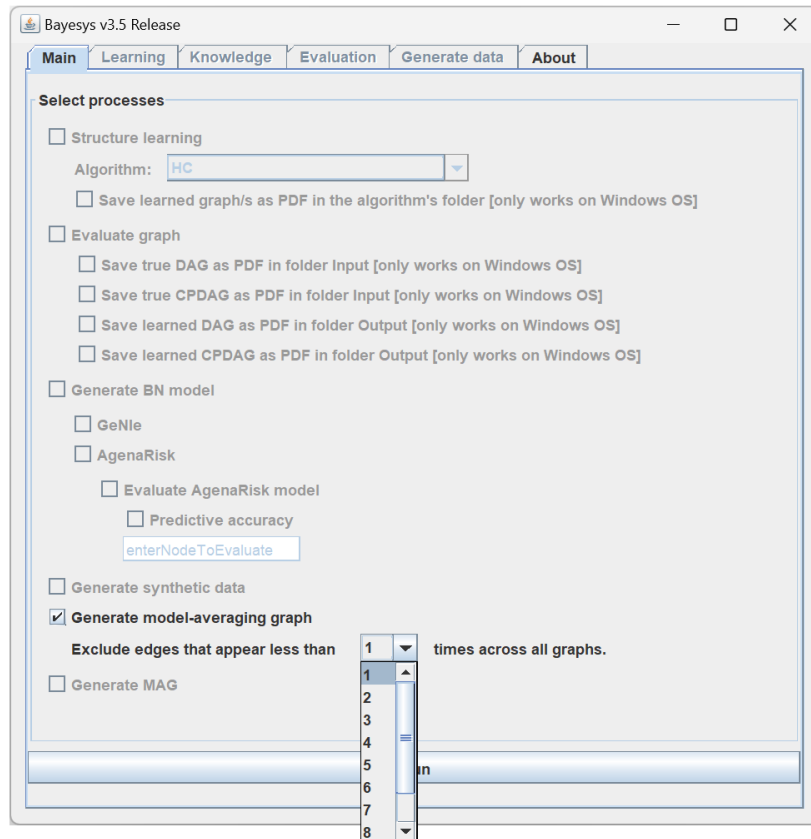
## 10. Worked example: Model-averaging

In addition to investigating the graphs learned by the algorithms independently, we may also want to use a model-averaging approach to obtain some sort of an averaged graph over a set of independent graphs. In Bayesys, this can be done through the model-averaging method highlighted in **Figure 10.1**.

This method takes a CSV file as input containing the edges (including duplicates) across a number of graphs. In constructing the model-averaging graph, this process prioritises directed edges over undirected edges, under the assumption that a directed edge carries higher certainty than an undirected edge, and so it aims to orientate as many edges as possible. Bi-directed edges are ignored since they indicate independence due to confounding, similarly to how the absence of an edge indicates independence. The model-averaging process works as follows:

- i. Add directed edges to the model averaging graph, starting from highest occurrence;
  - a. Skip edge if already added in reverse direction;
  - b. Skip edge if it produces a cycle, reverse it and add it to edge-set C;
- ii. Add undirected edges starting from highest occurrence;
  - a. Skip edge if already added as directed;
- iii. Add directed edges found in C starting from highest occurrence;
  - a. Skip edge if already added as undirected.

The hyperparameter shown in **Figure 10.1** is optional. It enables the user to specify the minimum number of occurrences needed for an edge to be added to the model averaging graph.



**Figure 10.1.** The Model-averaging method available under tab *Main* in Bayesys v3.5.

For a worked example, follow the steps enumerated below which involve generating the model-averaging graph *All\_score-based*, as described in [14]. The input data file represents a set of graphical structures learned by applying different structure learning algorithms to a COVID-19 UK data set.

### STEP 1: Preparing file and directory.

For this illustration, we will be using the sample file available in directory *Sample input files/Model-averaging*. Copy the file *ALL\_score-based.csv* in directory *Input/* and rename the file to *modelAveragingGraphs.csv*. A screenshot of the data contained in this file is shown in **Figure 10.2**.

	A	B	C	D	E	F	G
1	ID	Variable 1	Dependen	Variable 2			
2		1 Season	->	Leisure_activity			
3		2 Deaths_w	->	Patients_in_MVBs			
4		3 Deaths_w	-	Lockdown			
5		4 Deaths_w	->	Second_dose_uptake			
6		5 Patients_i	->	Excess_mortality			
7		6 Majority_	->	Excess_mortality			
8		7 Majority_	-	Tests_across_all_4_Pillars			
9		8 Majority_	->	Season			
10		9 Majority_	-	Reinfections			
11		10 Majority_	->	Second_dose_uptake			
12		11 Lockdown	->	Leisure_activity			
13		12 Lockdown	-	Face_masks			
14		13 Face_mas	->	Patients_in_MVBs			
15		14 Face_mas	-	Majority_COVID_19_variant			
16		15 Face_mas	-	Transportation_activity			
17		16 Transport	-	Work_and_school_activity			
18		17 Patients_i	->	Season			
19		18 Patients_i	-	Deaths_with_COVID_on_certificate			
20		19 Patients_i	-	Hospital_admissions			
21		20 New_infec	-	Positive_tests			
22		1 Positive_tr	-	Reinfections			
23		2 Positive_tr	-	New_infections			
24		3 Excess_m	->	Season			
25		4 Season	->	Leisure_activity			
26		5 Season	->	Face_masks			

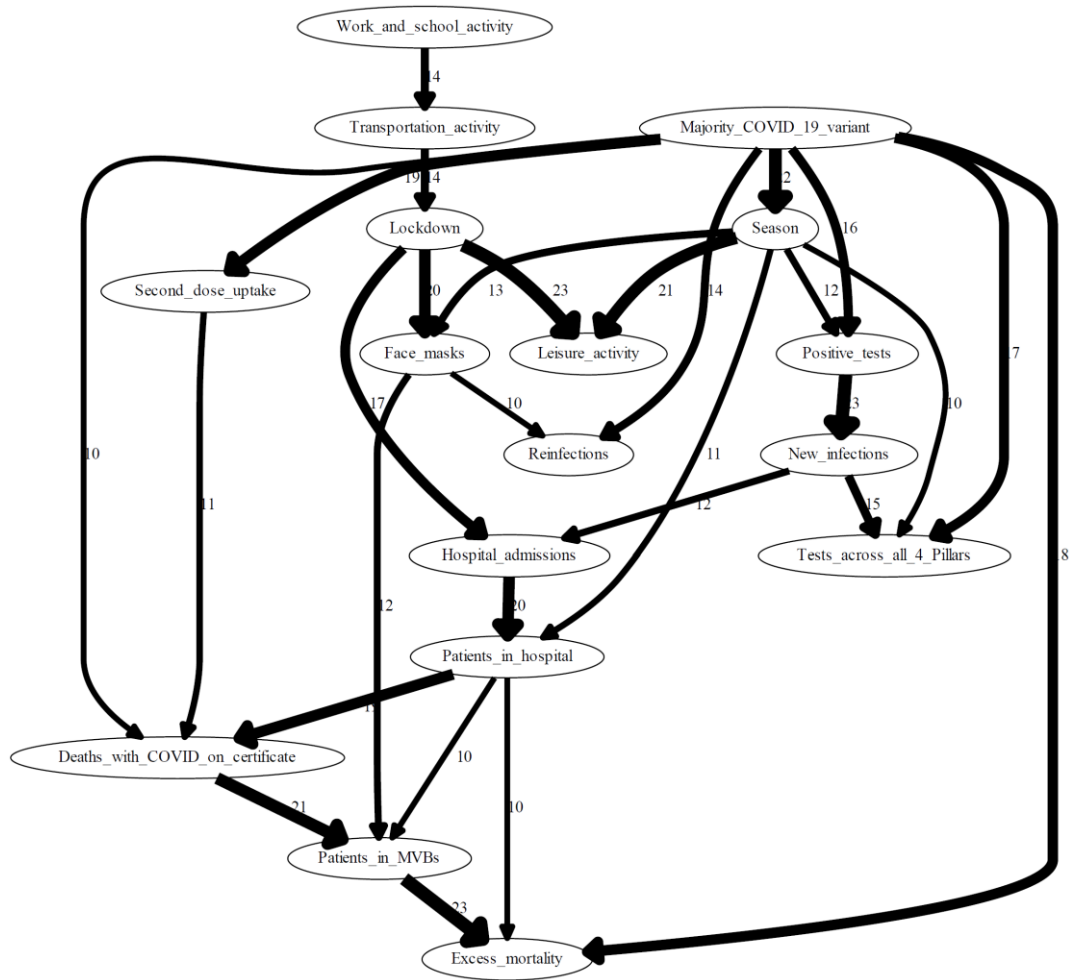
**Figure 10.2.** Part of the contents of the sample input file *modelAveragingGraphs.csv*.

### STEP 2: Running Bayesys.

Run Bayesys. Under tab *Main*, click on *Generate model-averaging graph*. Set the hyperparameter to 10, to be consistent with the results reported in [14]. That is, the input file contains a set of edges corresponding to 30 different structure learning experiments. Setting the hyperparameter threshold to 10 implies that the model-averaging graph will consider edges that appeared in at least one third of the 30 structure learning experiments. Hit *Run*.

### STEP 3: Reading the results.

The output involves the files *Model\_averaging\_graph.pdf* and *modelAveragingGraph.csv*, generated in folder *Output*. Both files contain the same information, in a different format. **Figure 10.3** presents the model-averaging graph generated in the PDF file.



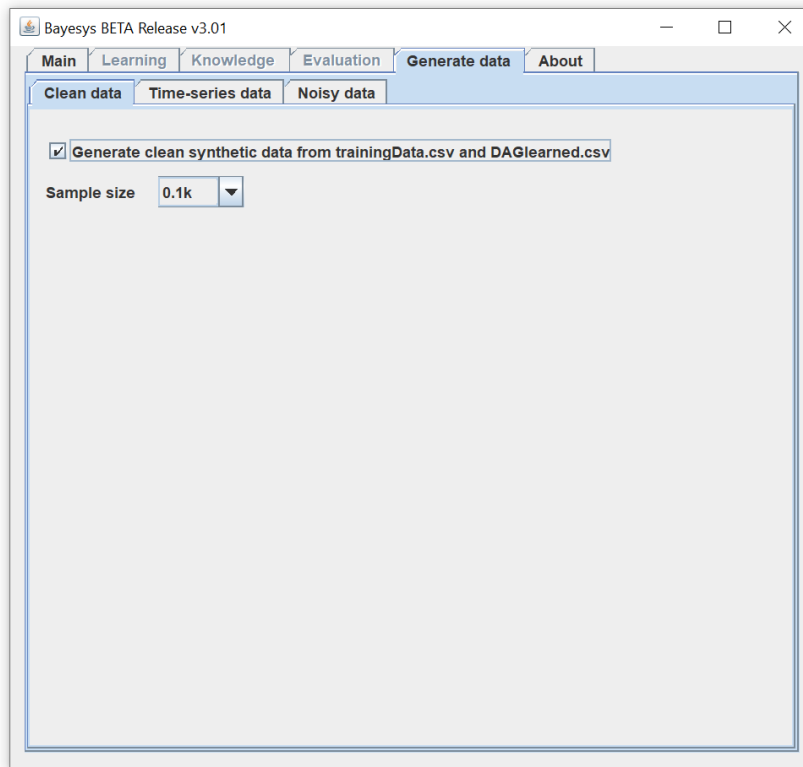
**Figure 10.3.** The *All\_score-based* model-averaging graph generated through the worked example, which is based on an experiment described in [14]. The graph contains a total of 29 edges, where the edge labels represent the number of times the given edge appeared in the input data (i.e., in the 30 independent graphs considered as input), and the width of the edges increases with this number.



## 11. Generate synthetic data

### 11.1. Generate clean data

The method *Clean data* is independent of other methods and can be used to generate clean synthetic data from a given BN model. Selecting *Generate synthetic data* under tab *Main* activates tab *Generate data*, along with its three sub-tabs as shown in **Figure 11.1**. This method requires *DAGtrue.csv* and *trainingData.csv* (to learn the CPTs) in folder *Input*, and the user to specify the required sample size as shown in **Figure 11.1**. After you run and complete the process, you should find the output file *trainingDataClean.csv* in directory *Output/Synthetic data/Clean*.



**Figure 11.1.** Features of tab *Clean data*, which is activated after selecting *Generate synthetic data* under tab *Main*. This illustration is based on Bayesys v3.01.

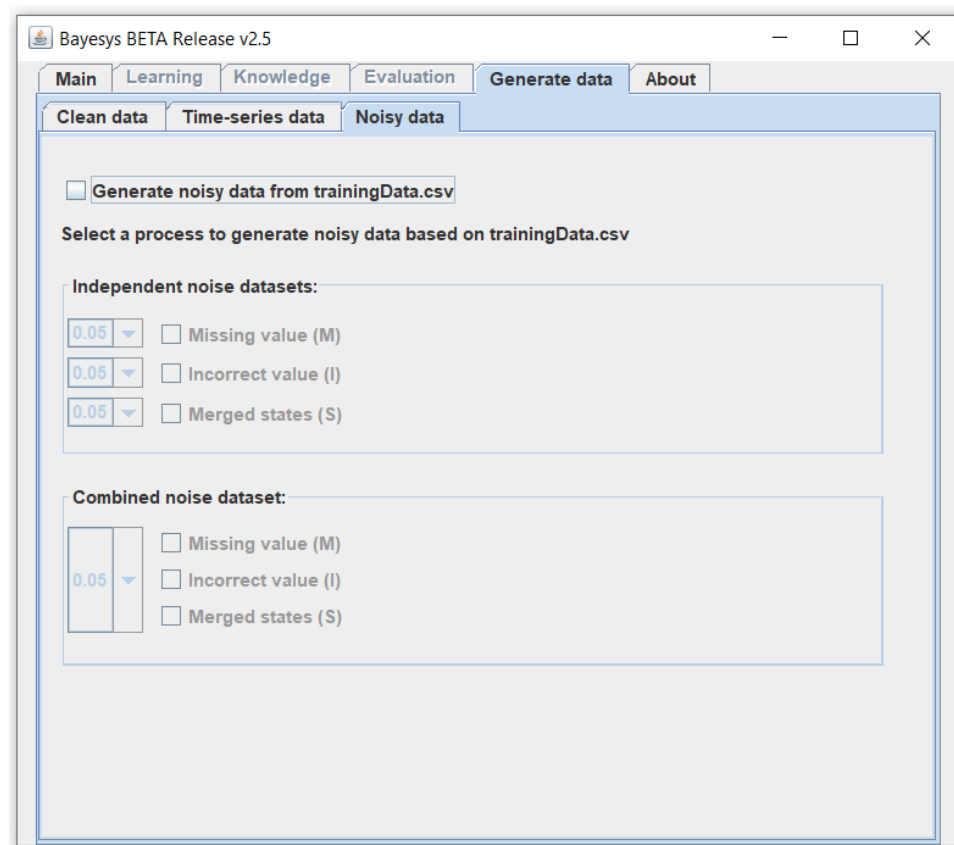
### 11.2. Generate noisy data

The tab *Noisy data* was implemented in Bayesys as part of the empirical evaluation assessments in [5]. Unlike *Clean data*, this method requires only *trainindData.csv* in folder *Input*, and produces all output files in directory *Output/Synthetic data/Noisy*.

Three types of data noise can be simulated, independently or in combinations of two, as shown in **Figure 11.2**. For example, selecting two types of data noise under *Independent noise data sets* should produce two different noisy data sets, one for each type of data noise, whereas selecting two types of data noise under *Combined noise data set* will produce a single data set that incorporates both types of data noise selected. You could combine all three types of data noise into a single data set by repeating this process multiple times, and replacing *trainindData.csv* with the resulting noisy file at each iteration.



The filename of the output file depends on the types of noise selected as well as the associated parameters (see **Figure 11.2**) 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 noise *M* with parameter 0.05 will produce the filename *trainingData\_M5*, whereas selecting both *M* and *I* will produce the filename *trainingData\_MI5*. **Figure 11.3** presents the output we get in the terminal window of NetBeans after running *Data noise* on the ASIA network data set with 10k sample size, and selecting *M* and *I* under *Combined noise data set* with the rate of noise set to 0.05.



**Figure 11.2.** The different types of data noise that can be simulated, found under tab Noisy data, which is activated after selecting the method Generate synthetic data under tab Main. This illustration is based on Bayesys v2.5.

```

_____ 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

```

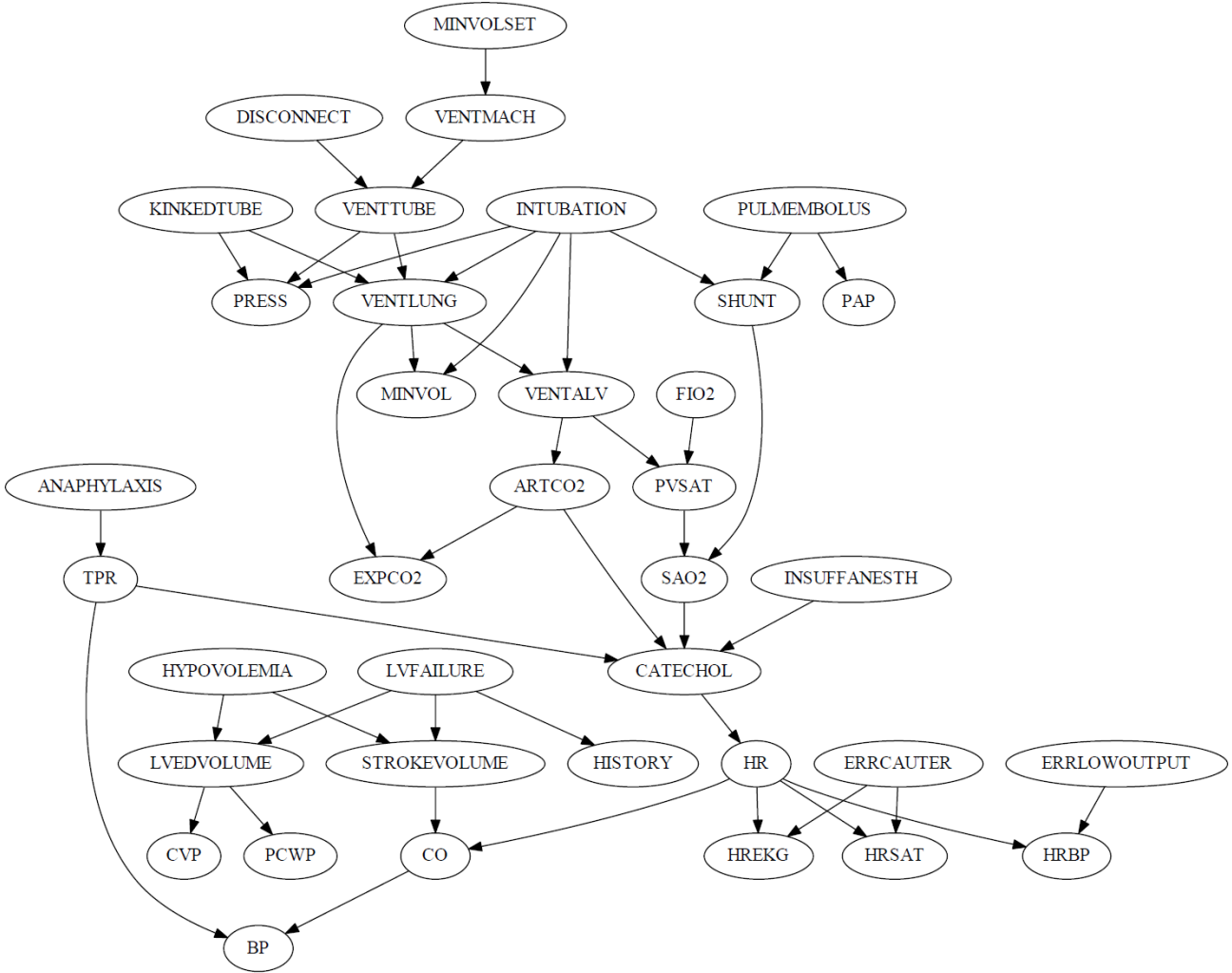
**Figure 11.3.** The output information generated in the terminal window of NetBeans IDE after selecting data noise *M* and *I*, with parameter set to 0.05 for both cases, to be performed on the ASIA network data set with 10k sample size. This illustration is based on Bayesys v1.7.

## 12. Generate MAG

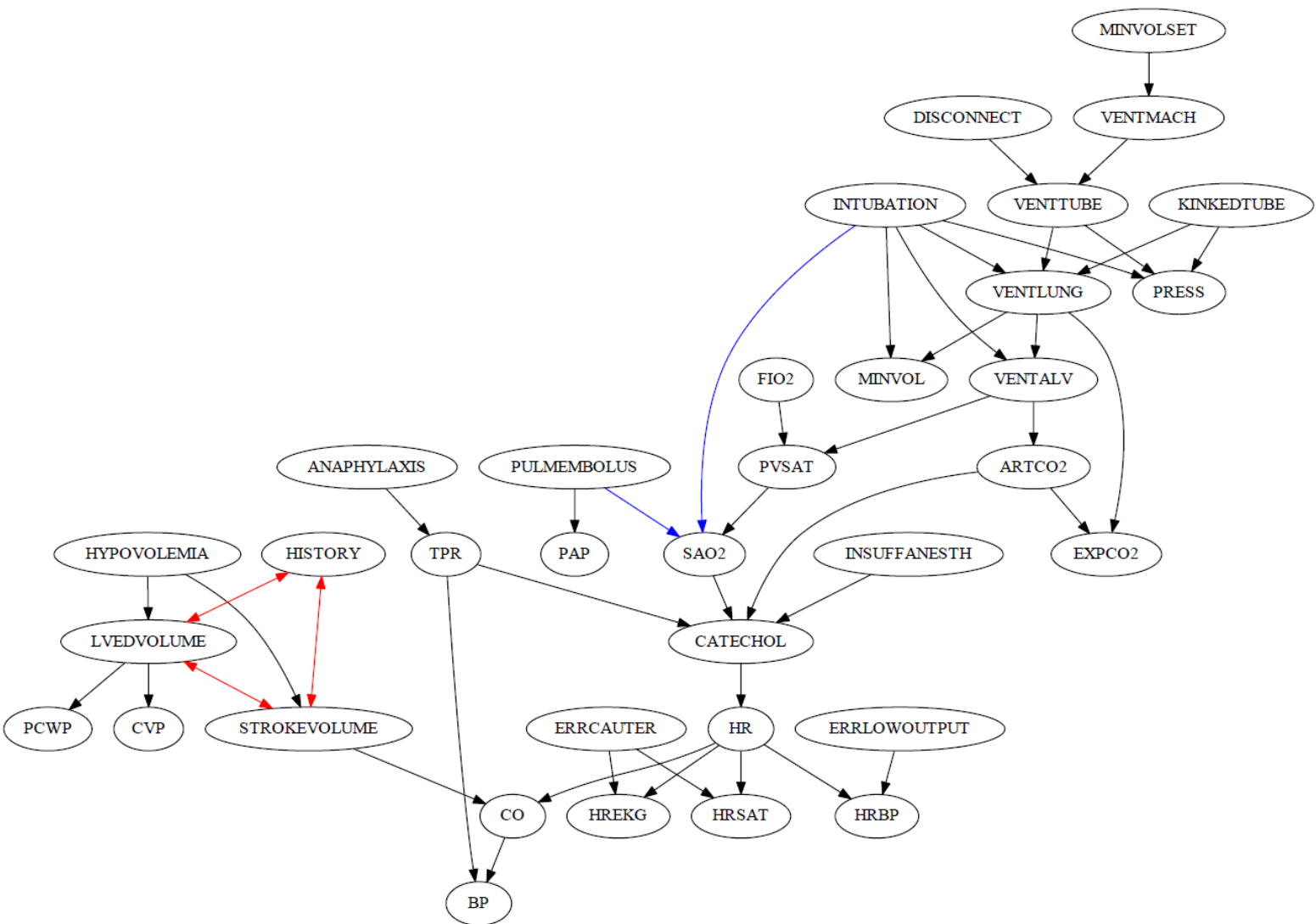
A MAG (i.e., Mixed Ancestral Graph) is an extension of a DAG that captures latent confounders. The method *Generate MAG* was implemented in Bayesys as part of the empirical evaluation assessments in [5], and can be performed independent of other methods.

*Generate MAG* requires three input files: a) *DAGtrue.csv*, b) *trainindData.csv*, and c) *trainingDataMAG.csv*, all present in folder *Input*. The input file *trainingDataMAG.csv* can be viewed as a copy of *trainingData.csv* minus the variables that are assumed to be latent/missing. For example, if *trainingData.csv* contains variables  $\{A, B, C, D, E\}$ , and *trainingDataMAG.csv* 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; i.e., *MAGtrue.csv* captures the edges present in the MAG, whereas *MAGtrue.pdf* draws the MAG with edge revisions highlighted relative to the true DAG. **Figure 12.1** and **Figure 12.2** are taken from [5] [9] and present the true DAG of the Alarm network along with its corresponding MAG when two variables are missing.



**Figure 12.1.** The true DAG of the Alarm network. Number of variables: 37. Number of arcs: 46. The illustration is based on Bayesys v1.7.



**Figure 12.2.** One of the true MAGs of the Alarm network used in [5] [9]. Number of variables: 35. Number of edges: 46. Missing variables: LVFAILURE, SHUNT. Blue arcs and red bi-directed edges represent edges that are not present in the true DAG. The illustration is based on Bayesys v1.7.

## 13. Troubleshooting and things to know

### 13.1. Input data files

- Variable and state names should not include a comma. This is because the data is read as Comma Separated Values (CSV), which means that the system automatically considers a comma as a data value separator.
- Variable names should not start with a numeric value.
- Any variable names specified as knowledge-based constraints (e.g., in *constraintsTemporal.csv*) should match (case sensitive) the variable names specified in *trainingData.csv*.

### 13.2. While running the system

- Before you run a method in Bayesys, make sure that you close ALL the relevant output files. If an output file remains open while running Bayesys, the system might complete the process without returning an error, but will *not* update the output file running in the background. For example, if you run *Structure learning* and the file *DAGlearned.csv* is already present in folder *Output* and is open in Excel, the learning process will complete but will *not* update *DAGlearned.csv*.

### 13.3. NetBeans terminal output errors

- ERROR: “*I/O error while writing the dot source to temp file!*”. This error associates with the PDF graphs generated when calling the Graphviz library. This error occurs when a PDF file being generated by Bayesys is already present in that folder but might be in use by the OS. However, you may sometimes get this error for no apparent reason. When you get this error, check that the relevant PDF files have been updated correctly during structure learning (refer to *Time modified*). Alternatively, rerun the process and, if it was not due to the PDF file being open, the error is unlikely to appear during the second run.
- ERROR: “*Invalid maximum heap size: -Xmx25000m. The specified size exceeds the maximum representable size.*”. This might be because you have installed the 32-bit version of NetBeans. You will have to install the 64-bit version (and the corresponding 64-bit JDK version).
- ERROR: “*java.lang.OutOfMemoryError: Java heap space*”. Memory issues may occur when working with large data sets, or when parameterising enormous CPTs. For example, if you use knowledge-based constraints to force directed edges entering specific nodes, Bayesys will accept up to 11 parents per node. However, a node having 11 parents may produce a CPT size of billions or trillions of parameters, and which may cause an out-of-memory error.

### 13.4. Warnings

- Red coloured warning messages such as “*WARNING: The LL implementation is limited to max node in-degree 8. A graph with max node in-degree >8 has been*”

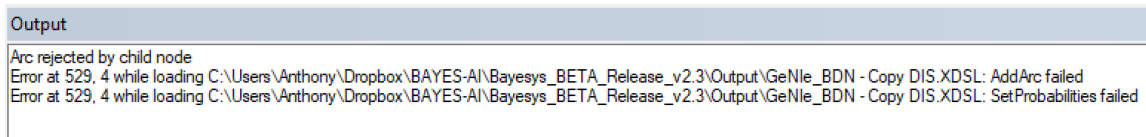
*skipped during score-based learning*" and "*WARNING: Variable variableName has more than 20 states*" are informational only and do not indicate an error.

### 13.5. Computational time

- Large graphs with hundreds of variables increase computational time faster than linear. The computational time of an algorithm 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 and/or edges may also significantly contribute to the overall computational time.

### 13.6. BN model generator (GeNIe)

- ERROR: "*Arc rejected by child node*" (full error output shown in **Figure 13.1**). This error is generated in the Output window of GeNIe when the Bayesys generated XDSL file contains a Utility node serves as a parent of a chance node. This is because, in GeNIe, children of Utility nodes can only be MAU/ALU nodes (refer to the GeNIe manual [13]). One way of resolving this issue is to make use of the Temporal constraints and add all Utility nodes in Tier 2, for example, and all other nodes in Tier 1, to ensure that the learned graph will comply with the graphical restriction in GeNIe.



**Figure 13.1.** The error generated in GeNIe when the XDSL file contains Utility nodes that are parents of chance nodes. In this example, note that the error also states that the problem can be found in line-of-code 529, at character position 4.

- Because GeNIe replaces many non-alphabetic values (including single integers) with characters “x” and “\_”, it is recommended that you replace integer state values with words (e.g., by changing state “1” to “one”); otherwise, you will not be able to differentiate these states in GeNIe.

## Appendix A: Revision notes

Old revision notes may point to sections, figures and tables that have had their numbering changed in future revisions, or to information that might no longer be available in the latest document.

### v1.5 Main revision notes – Feb 2020:

- The way *Precision* score is computed has been revised (this also influences the *F1* score).
- The *Evaluate graph* method now gives the option to the user to generate *DAGlearned.pdf*, corresponding to the graph of *DAGlearned.csv*.
- A new main method has been added, called *Add noise to data* (refer to Section 11).
- A new main method has been added, called *Generate MAG* (refer to Section 12).
- SaiyanH will infrequently fail to orientate all edges during phase 2. When this happens, the orientation of the edges will be randomised. The terminal window of NetBeans now outputs this information.

### v1.6 Main revision notes- Mar 2020:

- *DAGlearned.pdf* generator, under method *Evaluate graph*, is now optional and supports edges “o→” and “o—o”.
- The number of independent graphical fragments is now represented by an independent evaluation method, under tab *Evaluation*, and requires *trainingData.csv* present in folder *Input* to run.

### v1.7 Main revision notes - May 2020:

- Improvements have been made to the code that reads data from CSV files. Bayesys should now automatically resolve the following issues while reading CSV data:
  - Error caused by data sets with excess commas, often caused due to heavy file editing.
  - Error caused by data sets that incorporate hidden values. These values were neither visible in CSV format, nor in TXT format, 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.
- The system now stops the structure learning process if the input data set is found to incorporate missing data values (i.e., empty data cells), and informs the user in the terminal window of NetBeans.
- The system now stops the structure learning process if the name of a variable starts with a numeric character, and informs the user in the terminal window of NetBeans.
- The system now warns the user, in the terminal window of NetBeans, when the name of a variable in *DAGlearned.csv* does not match any of the variable names in *DAGtrue.csv*.
- The system now warns the user, in the terminal window of NetBeans, when a variable contains more than 20 states.
- Corrected a bug in the computation of the BIC score.
- The *Evaluation* now returns  $F1=0$  when  $F1=NaN$ ; i.e., in the case of  $Recall = Precision = 0$ .
- The runtime information for SaiyanH now includes the time spent in each of the three learning phases of SaiyanH.
- A correction has been made to the function that outputs information about the number of nodes present in *DAGtrue.csv*.

### v2.0 Main revision notes – Nov 2020:

- Improved the computational efficiency of score-based learning, constraint-based learning, and cycle detection. These revisions have improved the learning speed of SaiyanH by approximately 2.38 times (refer to subsection 3.1 for details).
- Bug fixes have slightly improved the learning accuracy of SaiyanH (refer to subsection 3.1 for details).
- Implemented two score-based structure learning algorithms based on traditional heuristics; namely the Hill-Climbing (HC) and TABU algorithms.
- All structure learning implementations now restrict the search space of graphs to max in-degree eight.
- The UI has gone through modifications to support the new implementations.
- SaiyanH will no longer produce an edge between variables that produce 0 association. This can happen when a variable in the data consists of just one state.
- Score-based learning and structure learning evaluation can now be performed with different log scales for the objective BIC function.
- The *Evaluation* method now includes information about the Log-Likelihood score, and whether the learned graph is acyclic (refer to Fig 5.4).
- Graphs generated in PDF now include disjoint nodes with no parents or children.

- Different types of knowledge-based constraints can now be incorporated into the structure learning process (refer to Section 4).

#### **v2.2 Main revision notes – Jan 2021:**

- Implemented additional knowledge-based constraints and revised Section 4 accordingly. More details about the knowledge-based constraints can be found in [6].
- Fixed a null exception bug that occasionally occurred when trying to perform BIC evaluation on a given *DAGlearned.csv* without previously running *Structure learning*.

#### **v2.3 Main revision notes – Feb 2021:**

- Extended the BN model generator method to support GeNIe BN models (in addition to AgenaRisk BN models), with file extension XDSL. For details, refer to subsection 6.1. For an example, refer to subsection 7.3. For troubleshooting and things to know, refer to subsection 13.6.
- Updated the set-up instructions of Section 1. Links to the appropriate Java JDK and NetBeans versions have also been updated.

#### **v2.4/2.41/2.421 Main revision notes – Feb/Mar 2021:**

- Fixed a bug during *Evaluate graph* where the system would return an error for BIC calculation, despite BIC not being selected under the *Evaluation* tab. Moreover, unselecting BIC now automatically disables the subprocess for the LL score.
  - Fixed a subsequent bug in v2.41 that sometimes caused the system not to output inference-based evaluation scores in the terminal window of NetBeans.
- Fixed a UI bug; now checkbox ‘Save graphs’ is always enabled irrespective of the algorithm selected.
- Updated the set-up instructions of Section 1.

#### **v3.01 Main revision notes – Oct 2021:**

- Fixed a UI bug: BIC selection under tab *Evaluation* now correctly modifies the BIC selection for the structure learning algorithms.
- BIC score comparisons are no longer truncated to two decimal points.
- BIC scores can now be computed for both DAGs and MAGs (refer to Section 5).
- Fixed a bug that occurred during the GeNIe model generating process and caused a “node position” error when loading the models in GeNIe versions older than version 3.
- The GeNIe XMLS generator now automatically replaces special characters that cause an error in the terminal window of GeNIe, when loading the learned models into GeNIe.
- The *Evaluate graph* process now enables users to generate PDFs of the true DAG and CPDAG in folder *Input*, and of the learned DAG and CPDAG in folder *Output*.
  - When the option *Saved learned DAG as PDF...* under *Evaluate graph* is combined with the *Decision network* process under tab *Knowledge*, the generated graph will now be a Bayesian Decision Network named *BDNlearned.PDF*, instead of a DAG named *DAGlearned.pdf*. This process takes into consideration both the *DAGlearned.csv* and *constraintsBDN.csv* files (refer to Fig 3.6) and Section 5).
- Added a new process that enables users to generate clean synthetic data from BN models. The UI involving the *Noisy data* generator process has been modified to include this new process. For details, refer to subsection 11.
- Fixed a bug that caused the Reverse arc operation during structure learning to violate some of the knowledge-based constraints.
- Fixed a bug that caused SaiyanH to violate the maximum node in-degree during Phase 2. This fix will cause SaiyanH to sometimes not carry an edge from Phase 1 into Phase 2, due to the maximum in-degree parameter input which is set to 8 by default.
- Implemented the new score-based Model-Averaging Hill-Climbing (MAHC) structure learning algorithm (refer to subsection 3.1 and relevant paper [7]).

#### **v3.2/3.21 Main revision notes – Jan 2022:**

- Made minor revisions to the way score comparisons are generated between the learned DAG and true DAG, and between the learned CPDAG and true CPDAG. The detailed evaluation criteria can be found in **Table 5.1** and **Table 5.2** for DAGs and CPDAGs respectively.



- Modified the HC and TABU algorithms to be able to explore the CPDAG space, in addition to the DAG space. The naming of the algorithms has changed to reflect this change; i.e., HC\_DAG, HC\_CPDAG, TABU\_DAG, and TABU\_CPDAG. The UI and the manual have also been modified accordingly.
- Minor bug corrections might have marginally influenced the learning performance of some algorithms. Refer to the new **Table 3.1** for the accuracy scores and learning times for each algorithm available in Bayesys v3.2, with reference to the six BNs available in the repository.
- Extended the manual to include a worked example on how to incorporate knowledge-based constraints.

### v3.3 Main revision notes – June 2022:

- Corrected a bug that involved shielded colliders when converting a DAG into CPDAG. This has also affected the evaluation scores under CPDAG.
- Removed the HC\_CPDAG and TABU\_CPDAG variants and renamed HC\_DAG and TABU\_DAG to HC and TABU respectively.
  - The number of max Tabu escapes for TABU algorithm is now set to  $|V|$  (i.e., number of variables), down from  $|V|(|V| - 1)$ . This makes TABU many times faster in exchange for accuracy.
- All structure learning algorithms now read the data variables as they appear in the data set, from left-to-right (previously data were read from right-to-left). Because the algorithms are sensitive to the order of the variables read from data, this influences the scores of HC and TABU to a large degree, and the scores of SaiyanH and MAHC to a small degree. For details about this issue see [10].

### v3.5/3.51/3.52/3.53/3.54/3.55 Main revision notes – Mar/Apr/May/Oct 2023:

- Added the structure learning algorithm GES by Chickering [8]. Details about the implementation of this algorithm and its performance can be found in subsection 3.1.
- Added a model-averaging approach that takes a set of graphical structures as input, and outputs a single graph that is representative of all independent input graphs.
- Max node in-degree is now set to 11, up from 8, for all structure learning, evaluation and BN model processes.
- Removed a process linked to AgenaRisk that involved evaluating the predictive accuracy of a given node.
- Corrected a bug that sometimes affected the edge-counts in model-averaging graphs (v3.55).
- Corrected a bug that sometimes caused the model-averaging method not to generate any edges in the relevant CSV file (v3.55).

### v3.6 Main revision notes – Jun 2024:

- **Objective score BIC improvement:** Restructured the computation and storage of the objective scores to improve efficiency.
- **TABU algorithm modification:** Modified the way the TABU algorithm escapes local maxima. TABU now attempts to escape local maxima by performing hill-climbing on the  $|V|$  neighbouring graphs that minimally decrease the objective score. If a higher-scoring graph is discovered, TABU will save it as the preferred graph and repeat the process. This modification also affects SaiyanH, which performs TABU search during phase 3.
- **MAHC pruning and TABU integration:** TABU can now be combined with MAHC pruning through hyperparameter selection.
- **New case studies:** Added two new case studies to the Bayesys repository: DIABETES and COVID-19 [14] [15].
- **Bug fixes:** Resolved a bug causing incorrect application of temporal constraints when performing structure learning with multiple algorithms and multiple sets of temporal constraints.



## Appendix B: Stats for nerds

**Table B.1.** Evolution of the Bayesys NetBeans IDE project.

Release date	Bayesys version	Java classes	Lines of code (LOC) <sup>4</sup>
Feb 2019	v1	18	5,598
Mar 2019	v1.1	19	6,806
Jun 2019	v1.2	23	8,183
Aug 2019	v1.28	25	9,367
Jan 2020	v1.4	20	9,382
Feb 2020	v1.5	22	11,469
Mar 2020	v1.6	22	11,304
May 2020	v1.7	24	13,346
Nov 2020	v2	25	14,321
Jan 2021	v2.2	29	15,490
Feb 2021	v2.3	29	16,162
Oct 2021	v3	34	20,382
Jan 2022	v3.2	34	21,699
Jun 2022	v3.3	35	23,197
Mar 2023	v3.5	37	27,154
Jun 2024	v3.6	36	27,260

**Table B.2.** Evolution of the Bayesys manual.

Release date	Document version	Page count	Word count
Mar 2019	v1	27	5,503
Jun 2019	v1.2	27	6,262
Aug 2019	v1.28	30	6,870
Jan 2020	v1.4	18	3,055
Mar 2020	v1.6	22	3,926
May 2020	v1.7	24	4,602
Nov 2020	v2	27	6,361
Jan 2021	v2.2	31	8,071
Feb 2021	v2.3	47	10,224
Oct 2021	v3	52	12,382
Jan 2022	v3.21	53	12,754
Jun 2022	v3.3	53	12,863
Mar 2023	v3.5	60	13,963
Jun 2024	v3.6	58	14,926

---

<sup>4</sup> Include comments and unused functions.

## References

- [1] A. C. Constantinou, “Bayesian Artificial Intelligence for Decision Making under Uncertainty,” Engineering and Physical Sciences Research Council (EPSRC), EP/S001646/1, 2018.
- [2] AgenaRisk, “AgenaRisk: Bayesian Network Software for Risk Analysis and Decision Making,” 2019. [Online]. Available: <https://www.agena.ai>.
- [3] E. R. Gansner and S. C. North, “An open graph visualization system and its applications to software engineering,” *Software – Practice and Experience*, vol. 30, no. 11, p. 1203–1233, 2000.
- [4] A. C. Constantinou, “Learning Bayesian networks that enable full propagation of evidence,” *IEEE Access*, vol. 8, p. 124845–124856, 2020.
- [5] A. C. Constantinou, Y. Liu, K. Chobtham, Z. Guo and N. K. Kitson, “Large-scale empirical validation of Bayesian Network structure learning algorithms with noisy data,” *International Journal of Approximate Reasoning*, vol. 131, pp. 151-188, 2021.
- [6] A. C. Constantinou, Z. Guo and N. K. Kitson, “The impact of prior knowledge on causal structure learning,” *Knowledge and Information Systems*, no. 65, p. 3385–3434, 2023.
- [7] A. Constantinou, Y. Liu, N. K. Kitson, K. Chobtham and Z. Guo, “Effective and efficient structure learning with pruning and model averaging strategies,” *International Journal of Approximate Reasoning*, vol. 151, p. 292–321, 2022.
- [8] D. Chickering, “Learning equivalence classes of Bayesian-network structures,” *Journal of Machine Learning Research*, vol. 2, pp. 445-498, 2002.
- [9] A. C. Constantinou, Y. Liu, K. Chobtham, Z. Guo and N. K. Kitson, “The Bayesys data and Bayesian network repository,” Bayesian AI research lab, MInDS research group, Queen Mary University of London, London, UK., <http://bayesian-ai.eecs.qmul.ac.uk/bayesys/>, 2020.
- [10] N. K. Kitson and A. Constantinou, “The impact of variable ordering on Bayesian network structure learning,” *Data Mining and Knowledge Discovery*, pp. <https://doi.org/10.1007/s10618-024-01044-9>, 2024.
- [11] Y. Liu and A. Constantinou, “Improving the imputation of missing data with Markov blanket discovery,” in *In Proceedings of the 11th International Conference on Learning Representations (ICLR)*, Kigali, Rwanda, 2023.
- [12] A. Constantinou, “Evaluating structure learning algorithms with a balanced scoring function,” arXiv:1905.12666 [cs.LG], <https://arxiv.org/abs/1905.12666>, 2019.
- [13] BayesFusion, “GeNIe Modeler User Manual,” 2020. [Online]. Available: <https://support.bayesfusion.com/docs/GeNIe.pdf>.
- [14] A. C. Constantinou, N. K. Kitson, Y. Liu, K. Chobtham, A. Hashemzadeh, P. A. Nanavati, R. Mbuva and B. Petrungaro, “Open problems in causal structure learning: A case study of COVID-19 in the UK,” *Expert Systems with Applications*, no. 234, p. Article 121069, 2023.
- [15] S. Zahoor, A. Constantinou, T. M. Curtins and M. Hasanuzzaman, “Investigating the validity of structure learning algorithms in identifying risk factors for intervention in patients with diabetes,” <https://arxiv.org/abs/2403.14327>, 2024.
- [16] A. C. Constantinou, “Learning Bayesian networks with the Saiyan algorithm,” *ACM Transactions on Knowledge Discovery from Data*, vol. 14, no. 4, p. Article 44, 2020.