A workshop on Bayesian networks

Bojan Mihaljevic

Universidad Politécnica de Madrid

Analyx
Poznań, Poland

Table of Contents

The workshop

Code, slides and data

```
# install.packages('usethis')
usethis::use_course("https://bit.ly/2IDJoKH")
```

Your instructor

- Post-doc in the Computational Intellence Group
- Instructor and coordinator at the Advanced Statistics and Data Mining Summer School
- Author of the bnclassify R package

Overview

- Representation, reasoning, inference, learning;
- Datasets/networks: earthquake, asia, marks, breast cancer, car, alarm;
- Hands-on examples:
  - mainly discrete variables, also Gaussian;
  - main aim is to illustrate basic concepts;
- Coding: for most examples code is provided and you need only run it; in some cases you will need to uncomment and complete provided code.
- Feel free to interrupt and ask questions

Bayesian networks

Bayesian networks

Judea Pearl received the 2011 ACM Turing Award (before it amounted to 1M USD) for "developing a calculus for probabilistic and causal reasoning". As summarized by Stuart Russel:

> *[Judea Pearl] is credited with the invention of Bayesian networks, a mathematical formalism for defining* ***complex probability models****, as well as the principal algorithms used for* ***inference*** *in these models. This work not only revolutionized the field of* ***artificial intelligence*** *but also became an important tool for many other branches of engineering and the natural sciences. He later created a mathematical frame work for causal inference that has had significant impact in the social sciences.*

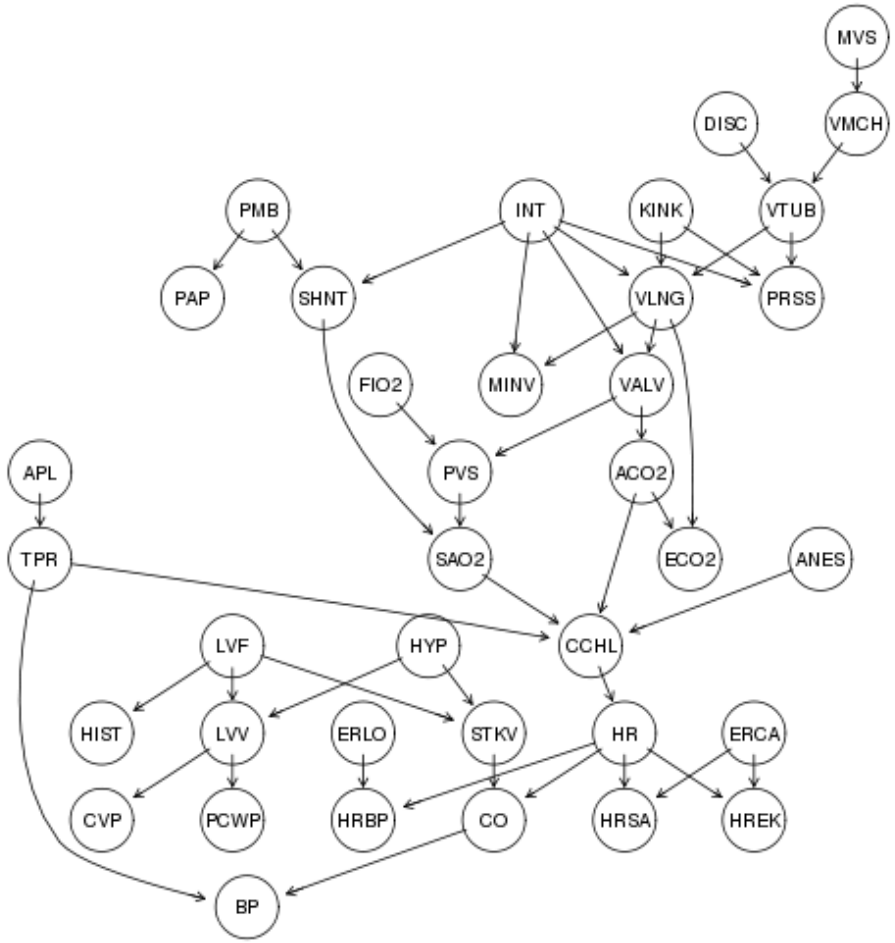
Reasoning with uncertainty

Bayesian networks (BNs) were designed to enable experts systems that can reason with uncertainty, e.g., medical diagnosis tools. Previously, representing complex probability distributions was not feasible.

Early examples of BNs were built by from human knowledge by domain experts. With the advent of machine learning, the learning of BNs from data developed.

Bayesian networks

- Compactly represent a joint probability distribution . . .
- . . . by exploiting conditional independencies
- Allow for inference and learning from data



JPD intractable

JPD size with binary variables

- $n = 10$: $|\boldsymbol{\theta}| = 2^{10} - 1 = 1023$
- $+10$ variables $\Rightarrow |\boldsymbol{\theta}| = |\boldsymbol{\theta}| \times 1024$
- n: $|\boldsymbol{\theta}| = 2^n - 1$

Medium to large n: the JPD is intractable.

- Computationally: cannot store and manipulate $\boldsymbol{\theta}$ in memory;
- Statistically: not enough data to estimate $\boldsymbol{\theta}$ robustly;
- Cognitively: acquire that many numbers from an expert; the numbers are small correspond to events hard to reasonably contemplate.
- 1000s, millions of variables are common (e.g., genetics)

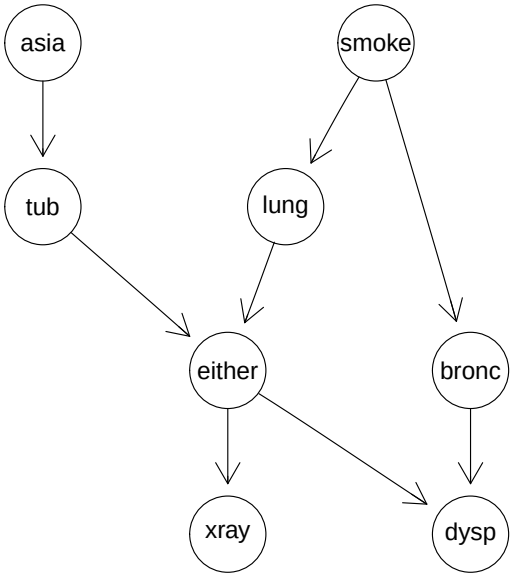
Compact representation with conditional independencies

Marginal independence: down to n parameters (binary variables):

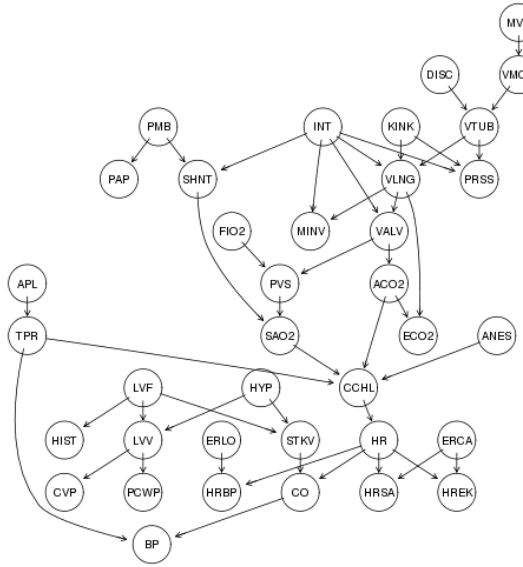
$$P(X_1, \ldots, X_n) = P(X_1) \ldots P(X_n) \qquad 2^n - 1 \text{ vs. } n \text{ parameters}$$

Examples of Bayesian networks:

- JPD: $2^8 - 1 = 255$
- BN: 18 (8 nodes, 8 arcs)



- JPD: $2^{37} - 1 = 134217727$
- BN: 509 (37 nodes, 46 edges)



Representation, inference, learning

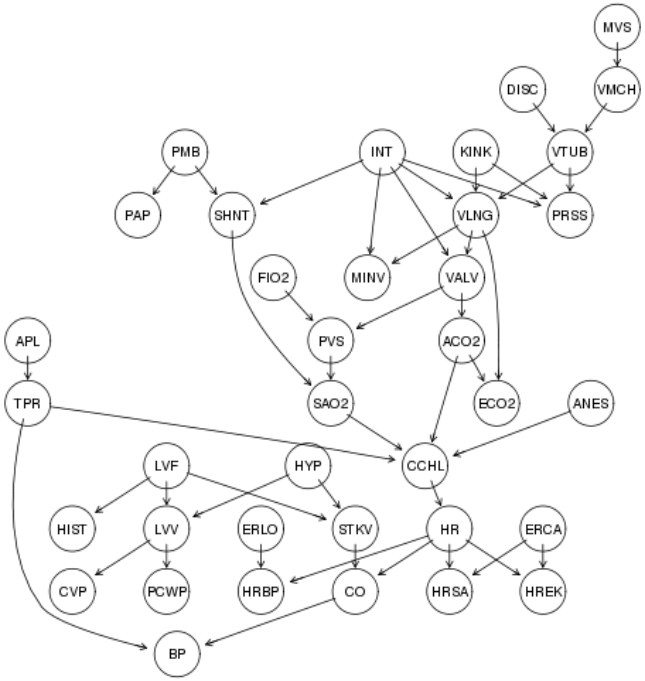
On the one hand, we have a declarative representation for representing probabilistic knowledge. It can represent a complex probability distribution by exploiting conditional conditional independencies among random variables.

On the other hand, we have algorithms for inference on this representation, as well as algorithms for learning the representation from data. Both inference and learning from data are computationally prohibitive in the general case: we are, after all, dealing with complex probability distributions. We often recurr to heuristics, with different methods providing different trade-offs.

Note that there is nothing inherently Bayesian, in terms of Bayesian statistics, about Bayesian networks.

Applications examples

Expert systems: Alarms for intensive care patients



37 variables

- 8 diagnoses
  - APL: Anaphylaxis
- 13 intermediate variables
- 16 measurements
  - HRBP: heart rate / blood pressure
  - CO: cardiac output

Reasoning under uncertainty

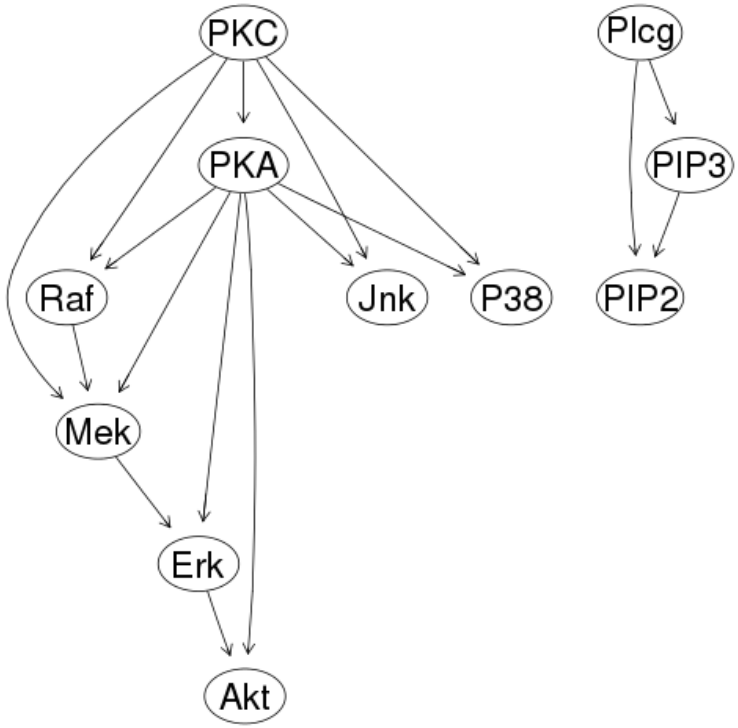
Evidence: (HRBP high, CO high) $= (HRBP = h^1,\ C0 = c^1)$

$$P(APL = a^1 \mid HRBP = h^1, C0 = c^1)?$$

Systems biology

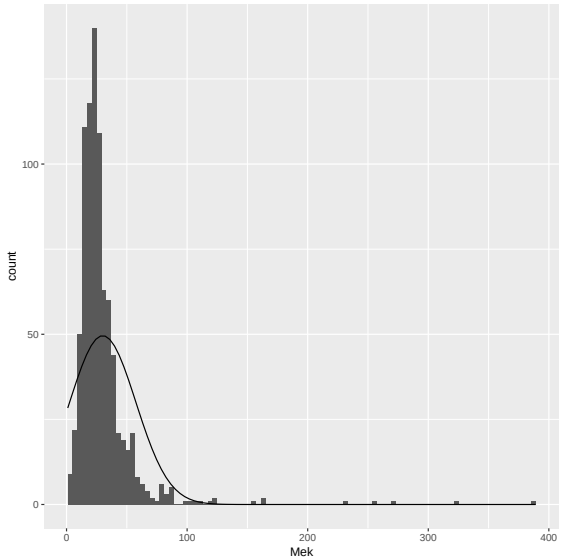
Causal Protein-Signalling Networks Derived from Multiparameter Single Cell Data. Karen Sachs, et al., *Science*, 308, 523 (2005).

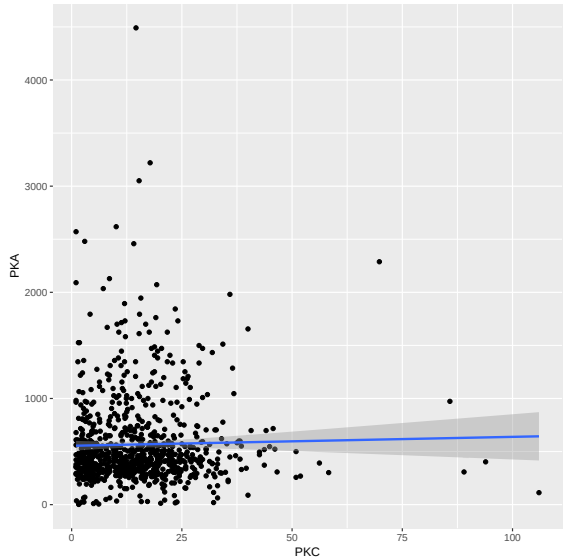
- 11 proteins
- 5400 measurements
- Goal: learn links among 11 proteins
- Validated:
  - Some relationships already known
  - Others confirmed by authors' further experiments
  - Obtained a DAG equivalent to the validated one



Systems biology

Raf	Mek	Plcg	PIP2	PIP3	Erk	Akt	PKA	PKC	P38	Jnk
26.4	13.2	8.82	18.30	58.80	6.61	17.0	414	17.00	44.9	40.0
35.9	16.5	12.30	16.80	8.13	18.60	32.5	352	3.37	16.5	61.5
59.4	44.1	14.60	10.20	13.00	14.90	32.5	403	11.40	31.9	19.5
73.0	82.8	23.10	13.50	1.29	5.83	11.8	528	13.70	28.6	23.1

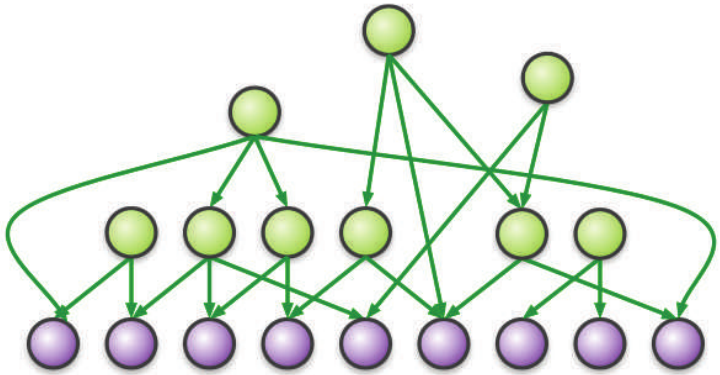




- Not Gaussian and dependencies not linear
- Discretize into three levels with k-means: low, average and high

Google Rephil

- Used for AdSense
- Learn concepts (clusters of co-ocurring words)
  - "apple pie" in concept of "chocolate cake"
  - "apple pie" not in concept of "apple ipod"



Nodes: presence/absence of words in query; concepts.
Source: Murphy, 2012

- Learned from 100 billion queries
- 12 million word nodes, 1 million concepts, 350 million edges
- Exact inference infeasible; approximate inference in 15 milliseconds

BNs in machine learning: Unsupervised

- Association discovery
- Clustering
- Expert systems: medical systems, troubleshooting
- Collaborative filtering
- Density estimation
- Causality discovery

BNs in machine learning: Supervised

- Handle and impute missing data
- Interpetable
- Include prior knowledge
- Generative: added bias and reduced variance by making assumptions regarding $P(\mathbf{X})$, rather than just $P(C \mid \mathbf{X})$
  - may fare better with less data, worse with more data

BNs in machine learning: Supervised

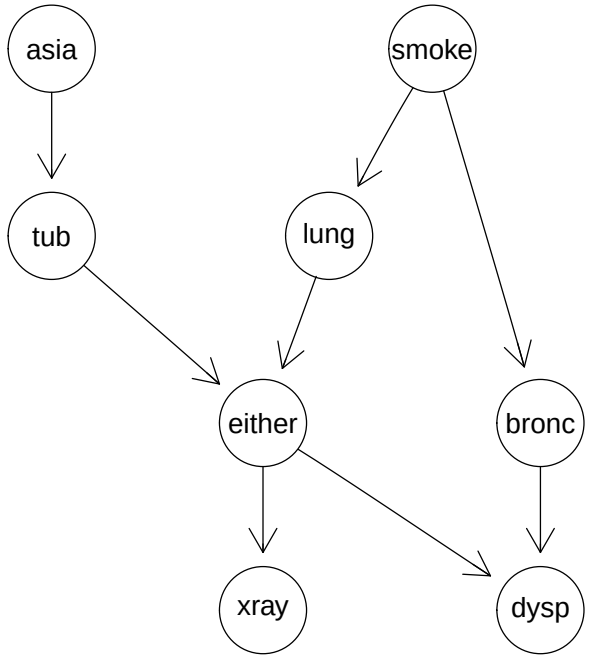
- A general tool for a number of different tasks
- Difficulties handling non-Gaussian continuous variables
- Useful when we are interested in the model itself, rathar than just its prediction, such as in scientific studies

Table of Contents

The chest clinic

Shortness-of-breath (dyspnoea) may be due to tuberculosis, lung cancer or bronchitis, or none of them, or more than one of them. A recent visit to Asia increases the chances of tuberculosis, while smoking is known to be a risk factor for both lung cancer and bronchitis. The results of a single chest X-ray do not discriminate between lung cancer and tuberculosis, as neither does the presence or absence of dyspnoea.

8 binary (true or false) variables:



- T: tuberculosis
- L: lung cancer
- B: bronchitis
- D: dyspnoea
- A: recent visit to Asia
- S: smoking
- X: X-ray
- E: either T or L

This network from Lauritzen & Spiegelhalter (1988) is sometimes called 'Asia' or 'lung cancer' network.

A joint probability distribution (JPD)

A probability for each possible combination of domain variables $\mathbf{X}$

L	B	S	P
yes	yes	yes	0.030
no	yes	yes	0.270
yes	no	yes	0.020
no	no	yes	0.180
yes	yes	no	0.002
no	yes	no	0.148
yes	no	no	0.003
no	no	no	0.346

$n = 3$ variables

- $\mathbf{X} = (L, A, S)$
- $l^1 := L = \text{yes}$, $l^0 := L = \text{no}$

JPD size

- Entries: $2^3 = 8$ entries
- Free parameters $\boldsymbol{\theta}$: $2^3 - 1 = 7$

$$P(l^1) = 0.030 + 0.020 + 0.002 + 0.003 = 0.055$$

$$P(l^1 \mid b^1) = \frac{0.030 + 0.002}{0.030 + 0.002 + 0.270 + 0.148} = 0.071$$

Independence

A	S	P
yes	yes	0.005
no	yes	0.495
yes	no	0.005
no	no	0.495

$$P(A) = 0.005 + 0.005 = 0.01$$

$$P(S) = 0.005 + 0.495 = 0.5$$

$$P(s \mid a) = P(s) \forall s, a$$

$$P(a^1, s^1) = P(a^1)P(s^1) = 0.01 \times 0.5 = 0.005$$

We denote with $A \perp\!\!\!\perp S$

Conditional independence

Conditional independence (CI) may hold when independence does not

$$L \not\perp\!\!\!\perp B$$

$$P(l^1) \neq P(l^1 \mid b^1)$$

$$0.055 \neq 0.071$$

$$L \perp\!\!\!\perp B \mid S$$

$$P(l^1 \mid s^1) = P(l^1 \mid s^1, b^1)$$

$$0.100 = 0.100$$

L	B	S	P
yes	yes	yes	0.030
no	yes	yes	0.270
yes	no	yes	0.020
no	no	yes	0.180
yes	yes	no	0.002
no	yes	no	0.148
yes	no	no	0.003
no	no	no	0.346

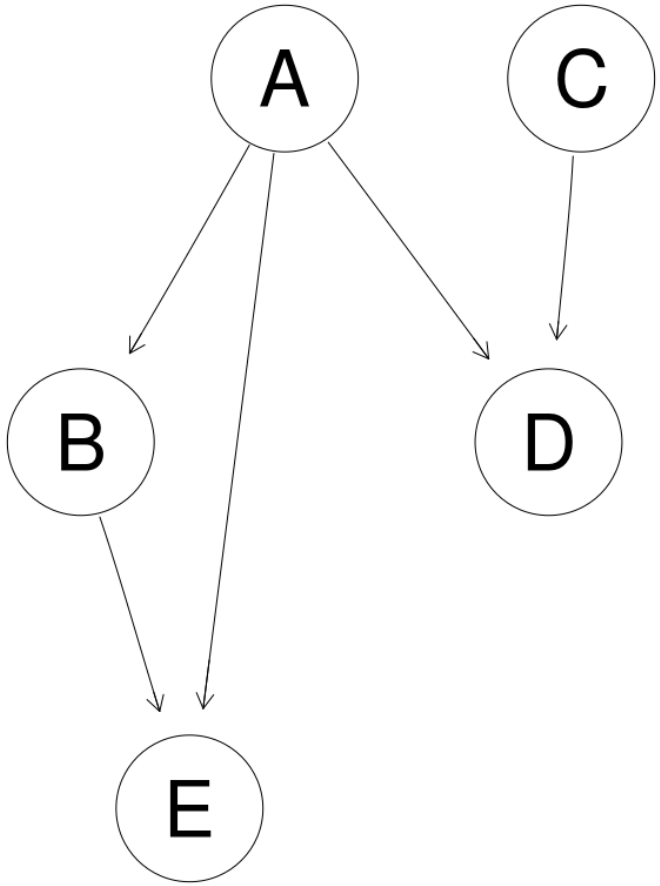
$$P(l^1 \mid b^1) = \frac{0.030 + 0.002}{0.030 + 0.002 + 0.270 + 0.148} = 0.071$$

Directed acyclic graphs

A graph $\mathcal{G}$ is a pair $(\mathbf{V}, \mathbf{E}_{\mathcal{G}})$: $\mathbf{V}$ are nodes and $\mathbf{E}_{\mathcal{G}}$ edges connecting the nodes. An edge can be undirected ($\{V_i, V_j\}$) or directed (an arc; $V_i \to V_j$). A directed path is a sequence of edges over $V_i, \ldots, V_k$ such that for each $i \in \{1, \ldots, k-1\}$, either $V_i \to V_{i+1}$ or $V_i - V_{i+1}$ is in $\mathbf{E}_{\mathcal{G}}$, with at least one edge being directed. A cycle is a directed path $V_i, \ldots, V_k$ such that $V_i = V_k$.

A directed acyclic graph (DAG):

- contains only directed arcs;
- contains no cycles.



Directed acyclic graphs

From each DAG we can derive at least one *topological ordering* $X_1, \ldots, X_n$, such that $V_i \rightarrow V_j$ implies $i < j$.
Examples for this graph are: (A, C, D, B, E) and (C, A, B, E, D)
A directed graph defines useful relationships such as: parent, child, ancestor, descendent, non-descendent.

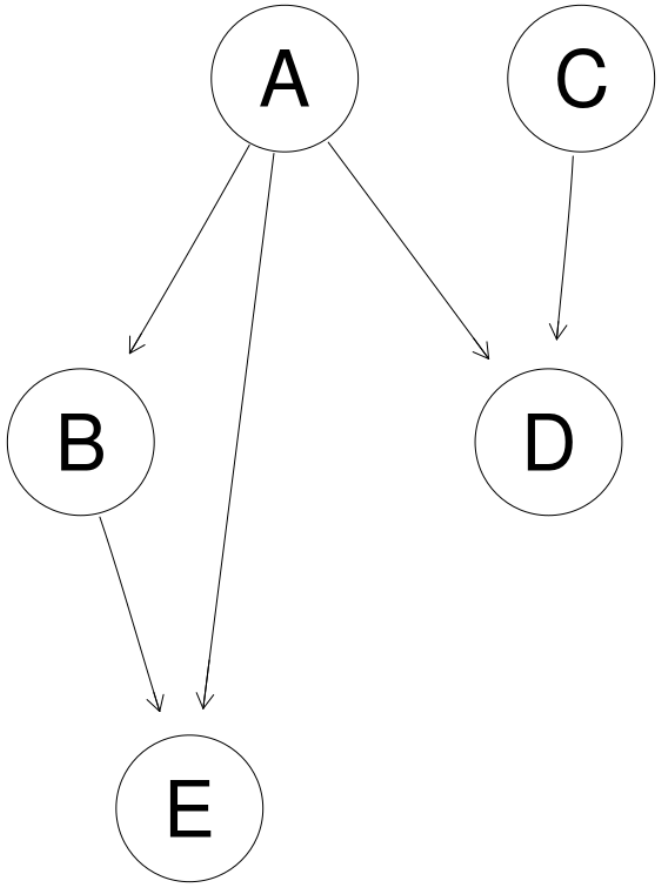


Table of Contents

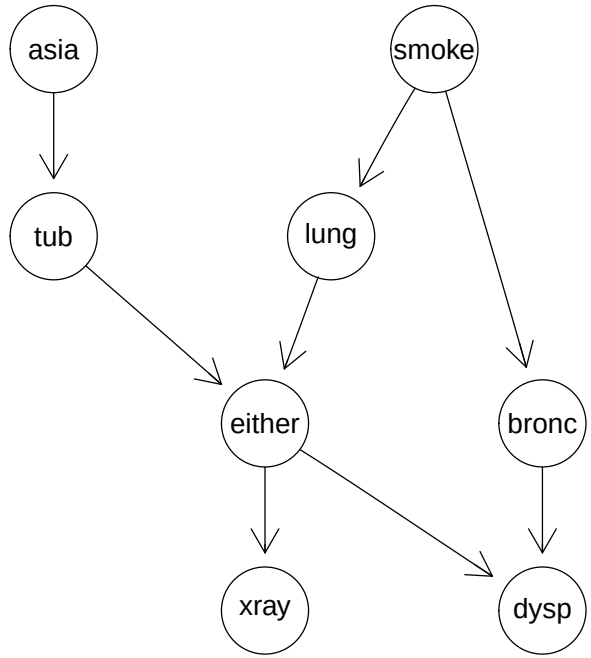
Structure

Chest clinic

A Bayesian network encodes a JPD by specifying conditional independencies among its variables. The basic set of independencies is

> *A variable is independent of its non-descendants given its parents,*

and additional independencies are derived from it. The conditional independencies are also refered to as constraints or relationships.



For example, lung cancer is not independent of bronchitis. Intuitively, having lung cancer increases our belief that the patient is a smoker, which is a risk factor for bronchitis. However, if we know that the patient is a smoker, then knowing whether he/she suffers from lung cancers does not add new information regarding bronchitis.

Reasoning patterns

Since the networks encodes the JPD we can perform probability queries of the type $P(\mathbf{Q} \mid \mathbf{E} = \mathbf{e})$, where $\mathbf{e}$ is evidence that we have observed on variables $\mathbf{E}$. Observing the evidence can change the probabilities of interest. These are examples of three main reasoning patterns with such queries:

- causal reasoning or prediction (downstream effects): we learn that a patient has been to Asia. Now the probability of him/her having tuberculosis goes up;
- evidential reasoning or explanation (from effects to causes) we learn that a patient has dyspnea; the probability of him/her having bronchitis goes up;

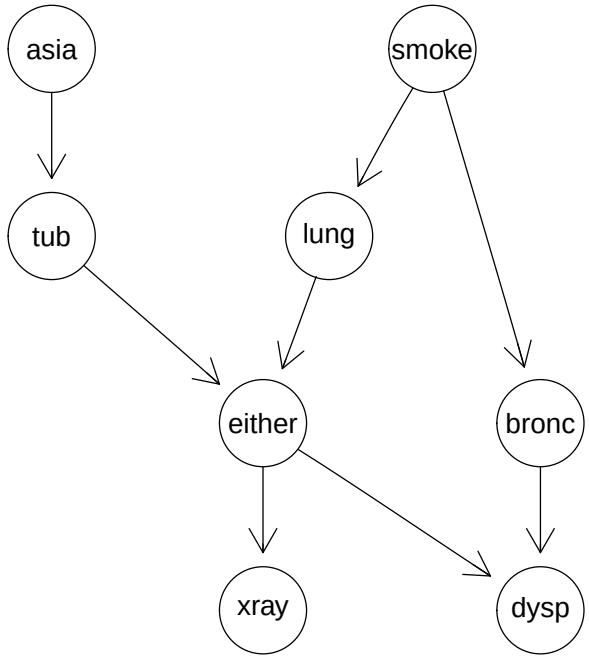
Reasoning patterns

Since the networks encodes the JPD we can perform probability queries of the type $P(\mathbf{Q} \mid \mathbf{E} = \mathbf{e})$, where $\mathbf{e}$ is evidence that we have observed or variables $\mathbf{E}$. Observing the evidence can change the probabilities of interest. These are examples of three main reasoning patterns with such queries:

- intercausal reasoning (interaction among causes of an effect): having lung cancer does not give us information about whether the patient has tuberculosis. However, once we know that a patient has either of them, it does. Namely, if we were now to learn that the patient has tuberculosis, the probability of him/her having lung cancer would go down. Thus, tuberculosis *explains away* lung cancer.

Reasoning patterns

Note that the directions of the arcs in the graph do not suggest an anscestor is a cause and a descendant is an effect.
Above, we discussed causes and effects in the sense as we interpret them in the real world: smoking can actually cause lung cancer.
Our Bayesian network gives a prognostic view of the domain, with causes upstream and effects downstream in the network. A diagnostic view of the same domain might encodes the same independencies but have the "effects" upstream and "causes" downstream.



Definition

A Bayesian network is a model for a probability distribution P. It is defined by:

- A directed acyclic graph $\mathcal{G} = (\mathbf{V}, \mathbf{E}_{\mathcal{G}})$ with vertices $\mathbf{V}$ and edges $\mathbf{E}_{\mathcal{G}}$, such that each $v_i \in \mathbf{V}$ corresponds to a random variable X_i in $\mathbf{X}$
- Parameters $\boldsymbol{\theta}$, with $\Theta_i \subset \boldsymbol{\theta}$ specifying the conditional probability distribution of X_i given its parents in $\mathcal{G}$, $\mathbf{Pa}_{\mathcal{G}}(X_i)$

By definition, $\mathcal{G}$ specifies the following conditional independences in the model:

A variable is independent of its non-descendants given its parents.

These conditional independence allow us to factorize $P_{\mathcal{G}}(\mathbf{x})$ as:

$$P_{\mathcal{G}}(\mathbf{x}) = \prod_{i=1}^{n} P_{\mathcal{G}}(x_i \mid \mathbf{pa}_{\mathcal{G}}(x_i)).$$

This factorization is the *chain rule for Bayesian networks*.

Graphs and distributions

$\mathcal{G}$ encodes a set of conditional independence assumptions regarding the distribution of $\mathbf{X}$. We say that, for disjoint subsets $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ of $\mathbf{X}$:

$\mathcal{G}$ is a dependency map (D-map) for P when

$$\mathbf{A} \perp_P \mathbf{B} \,|\, \mathbf{C} \Longrightarrow \mathbf{A} \perp_G \mathbf{B} \,|\, \mathbf{C}$$

$\mathcal{G}$ is a independency map (I-map) for P when

$$\mathbf{A} \perp_P \mathbf{B} \,|\, \mathbf{C} \Longleftarrow \mathbf{A} \perp_G \mathbf{B} \,|\, \mathbf{C}$$

$\mathcal{G}$ is a perfect map (P-map) for P when

$$\mathbf{A} \perp_P \mathbf{B} \,|\, \mathbf{C} \Longleftrightarrow \mathbf{A} \perp_G \mathbf{B} \,|\, \mathbf{C}$$

Local Markov property

Assuming that $\mathcal{G}$ is an I-map for P suffices to factorize P with respec to $\mathcal{G}$, that is, according to the chain rule for Bayesian networks:

In the other direction, if P factorizes according to $\mathcal{G}$ then it is an I-Map for $\mathcal{G}$. Thus, for any P that factorizes according to $\mathcal{G}$, the following set of independence relationships holds: $X_i \perp$ NonDescendants $_{X_i} | \mathrm{Pa}_{X_i}^{\mathcal{G}}$.

These independencies are referred to as the *local Markov independencies*. We have seen a number of examples in the chest clinic network.

From local Markov property to factorization

Factorize $P(\mathbf{x})$ by the chain rule for probabilities, following a topological order A, S, T, L, B, E, X, D:

$$P(\mathbf{x}) = P(a)P(s \mid a)P(t \mid a, s)P(l \mid a, s, t)P(b \mid a, s, t, l)$$
$$P(e \mid a, s, t, b)P(x \mid a, s, t, b, e)P(d \mid a, s, t, b, e, x)$$

Take L. Since $L \perp\!\!\!\perp B \mid S$, we can remove L from B's conditioning set

$$P(b \mid a, s, t) \text{ instead of } P(b \mid a, s, t, l)$$

Thus

$$P(\mathbf{x}) = P(a)P(s \mid a)P(t \mid a, s)P(l \mid a, s, t)P(b \mid a, s, t)$$
$$P(e \mid a, s, t, b)P(x \mid a, s, t, b, e)P(d \mid a, s, t, b, e, x)$$

We can do this for all variables.

From local Markov property to factorization

- Assume $X_1, \ldots, X_n$ is a topological ordering for $\mathcal{G}$
- Factorize $P(\mathbf{X})$ by the chain rule for probabilities

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | X_1, \ldots, X_{i-1})$$

- Consider $P(X_i | X_1, \ldots, X_{i-1})$
  - Because $\mathcal{G}$ is an I-map for P, we have $X_i \perp$ NonDescendants $_{X_i} | \mathrm{Pa}_{X_i}^{\mathcal{G}}$
  - By assumption, $\{X_1, \ldots, X_{i-1}\} = \mathrm{Pa}_{X_i} \cup \mathbf{Z}$, where $\mathbf{Z} \subseteq$ NonDescendants $_{X_i}$
  - From the local Markov property and the decomposition property we have: $P(X_i | X_1, \ldots, X_{i-1}) = P(X_i | \mathrm{Pa}_{X_i})$
- Thus:

$$P_{\mathcal{G}}(\mathbf{x}) = \prod_{i=1}^{n} P_{\mathcal{G}}(x_i \mid \mathbf{pa}_{\mathcal{G}}(x_i))$$

From local Markov property to factorization

We can now complete the factorization of P for chest clinic according to $\mathcal{G}$.

$$\text{From chain rule : } P(\mathbf{x}) = P(a)P(s \mid a)P(t \mid a, s)P(l \mid a, s, t)P(b \mid a, s, t, l)$$
$$P(e \mid a, s, t, b)P(x \mid a, s, t, b, e)P(d \mid a, s, t, b, e, x)$$

Remove non-parents from conditioning sets (e.g., $A \notin \mathbf{Pa}_{\mathcal{G}}(S)$)

$$P(\mathbf{x}) = P(a)P(s)P(t \mid a)P(l \mid s)P(b \mid s)P(e \mid l, t)P(x \mid e)P(d \mid b, e)$$

Graphical separation

Additional independence constraints $(\mathbf{A} \perp \mathbf{B} \mid \mathbf{C})$ imposed by $\mathcal{G}$ can be derived from the local Markov property. We will analyze them in terms of graphical separation and consider direct and indirect connections in $\mathcal{G}$.

Consider a direct connection between X and Y in $\mathcal{G}$, say $X \rightarrow Y$. We can specify a P^* that factorizes over $\mathcal{G}$ and is such that $X \not\perp Y|\mathbf{C}$, for any $\mathbf{C}$, by setting $Y = X$ and X is uniformly distributed. Therefore, $X \rightarrow Y$ does not imply $X \perp Y|\mathbf{C}$, because there exists P^*, which factorizes over $\mathcal{G}$, with $X \not\perp Y|\mathbf{C}$.

Conditional independencies are symmetric and therefore the graphical separation criterion does not imply $Y \perp X|\mathbf{C}$ either. Thus, $\mathcal{G}$ does not imply that parents are independent of their children nor the children of their parents.

Graphical separation

Consider a trail between X and Y via Z. For three-node networks, there are four cases. The first three are equivalent in terms of graphical separation:

- $X \leftarrow Z \leftarrow Y$
- $X \rightarrow Z \rightarrow Y$
- $X \leftarrow Z \rightarrow Y$

We have discussed the $X \leftarrow Z \rightarrow Y$ pattern for lung cancer (X) and bronchitis conditional (Y) on smoker (Z): they are not independent marginally but are independent conditional on smoker.

$X \rightarrow Z \leftarrow Y$ (converging connection) We have discussed this pattern for intercausal reasoning between lung cancer (X) and tuberculosis (Y) given either (Z). They are marginally indepent but not independent conditioned on either (Z). Importantly, they are not independent even if we do not observe Z but only its descendent: knowing that a patient has dyspnea increases the probability of either, which suffices to correlate lung cancer and tuberculosis.

Graphical separation

Let W be a descendant of Z in $\mathcal{G}$. Graphical separation depends on the arc pattern and on evidence:

	No evidence	Evidence
$X \to Z \to Y$		
$X \leftarrow Z \leftarrow Y$	$X \not\perp\!\!\!\perp_{\mathcal{G}} Y \mid \emptyset$	$X \perp\!\!\!\perp_{\mathcal{G}} Y \mid Z$
$X \leftarrow Z \to Y$		
$X \to Z \leftarrow Y$	$X \perp\!\!\!\perp_{\mathcal{G}} Y \mid \emptyset$	$X \not\perp\!\!\!\perp_{\mathcal{G}} Y \mid Z$ $X \not\perp\!\!\!\perp_{\mathcal{G}} Y \mid W$

Graphical separation

The connection patterns in three-node networks can be extended to arbitrary-length trails $X_1 \rightleftharpoons \ldots \rightleftharpoons X_n$. Intuitively, X_1 affects X_n only if no node on the trail is blocking the influence. For disjoint subsets **A B C**, $\mathbf{A} \perp \mathbf{B} \mid \mathbf{C}$ if all trails between **A** and **B** are blocked given **C**.

The above provides a basis for the definition of the formal criterion of graphical separation:

d-separation

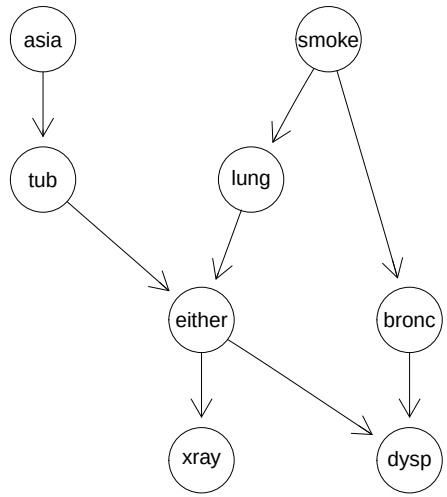
Z d-separates **X** from **Y** if along every trail between a node in **X** and a node in **Y** there is a node v satisfying one of the following:

1. v has converging edges and none of v or its descendants are in **Z**
2. v is in **Z** and does not have converging edges

Graphical separation

The following are some independencies implied by d-separation in our chest clinic model:

- $S \perp\!\!\!\perp_{\mathcal{G}} D \mid L, B$
- $S \perp\!\!\!\perp_{\mathcal{G}} A$ ($S \not\perp\!\!\!\perp_{\mathcal{G}} A \mid E$, $S \not\perp\!\!\!\perp_{\mathcal{G}} A \mid D$)
- $T \perp\!\!\!\perp_{\mathcal{G}} L$
- ($L \not\perp\!\!\!\perp_{\mathcal{G}} B \mid E, S$) (yet, $L \not\perp\!\!\!\perp_{\mathcal{G}} B \mid S$)



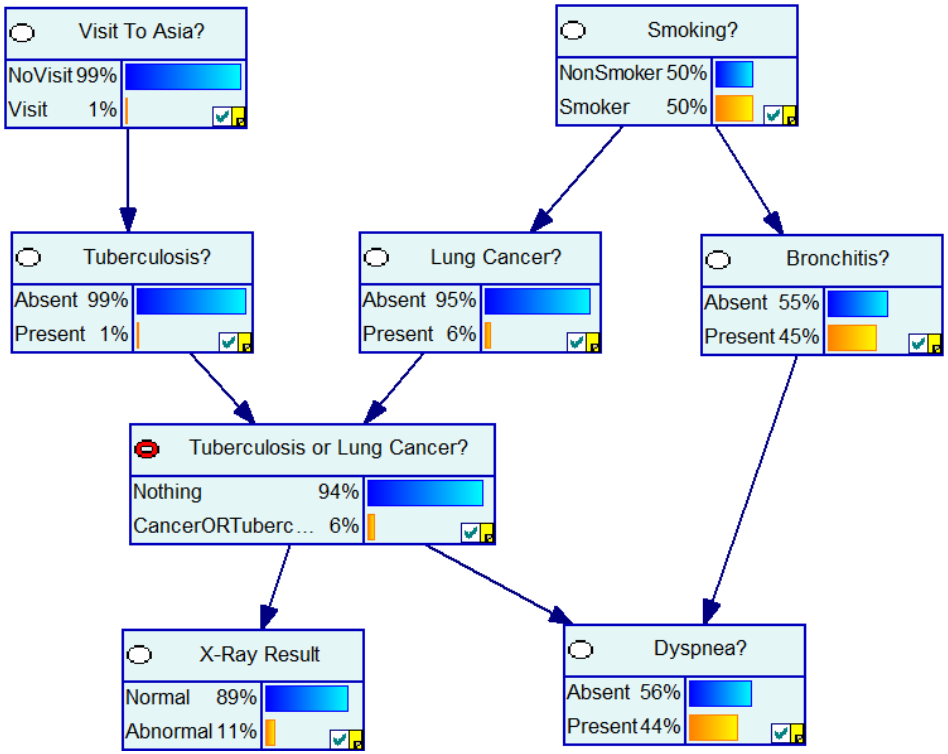
Let $I(\mathcal{G})$ of all independence constraints implied by d-separation:
$I(\mathcal{G}) = (\mathbf{X} \perp\!\!\!\perp_{\mathcal{G}} \mathbf{Y} \mid \mathbf{Z}) : \text{d-sep}_{\mathcal{G}}(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z})$

$I(\mathcal{G})$ is the set of *global Markov independencies* of $\mathcal{G}$ and corresponds to all the independencies associated with $\mathcal{G}$.

Graphical separation

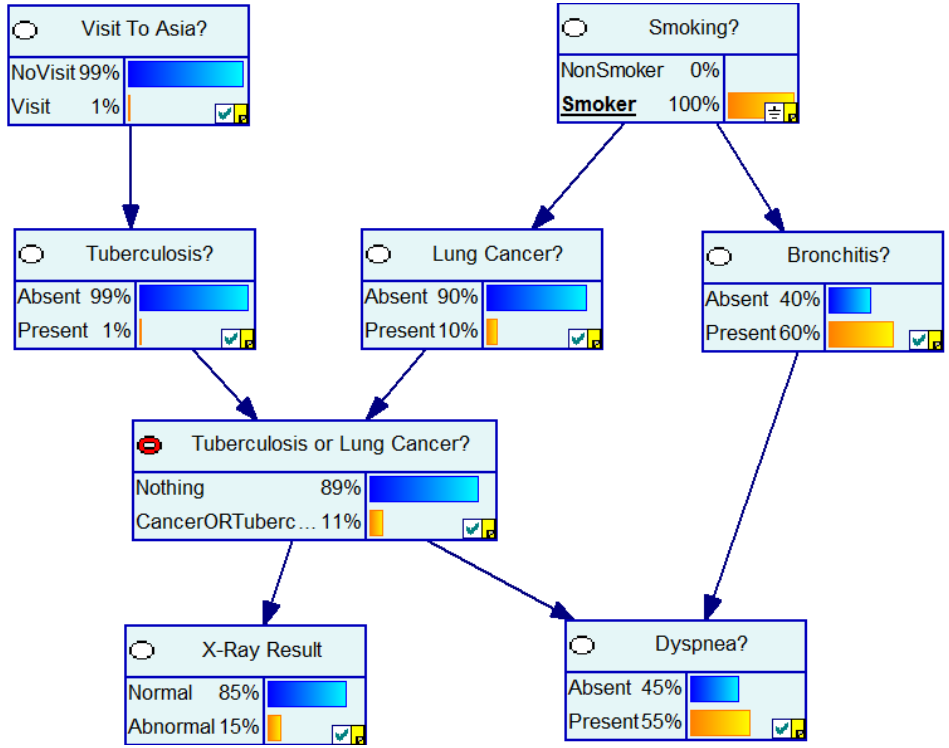
Examples by computing (conditional) marginal distributions.

No evidence



- $P(b^1) = .45$
- $P(d^1) = .44$
- $P(a^1) = .01$

A person is a smoker



- $P(b^1 \mid s^1) = .60$
- $P(d^1 \mid s^1) = .55$
- $P(a^1 \mid s^1) = .01$

Markov blanket

The *Markov blanket* of X is the set of variables that d-separates X from all other variables. It consists of its:

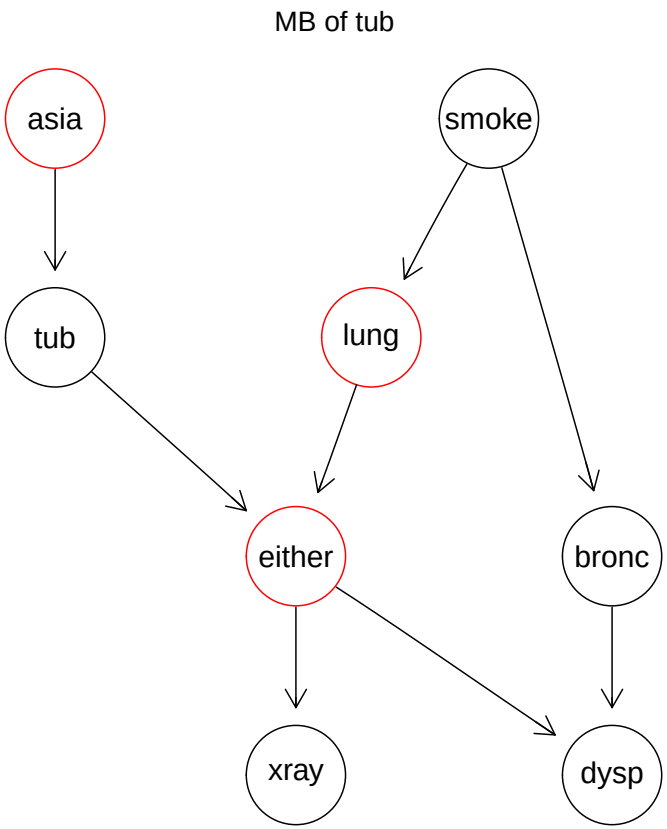
- parents ($\mathbf{Pa}_{\mathcal{G}}(X)$;
- children ($\mathbf{Ch}_{\mathcal{G}}(X)$);
- spouses, other nodes sharing a child ($\mathbf{Pa}_{\mathcal{G}}(\mathbf{Ch}_{\mathcal{G}}(X))$).

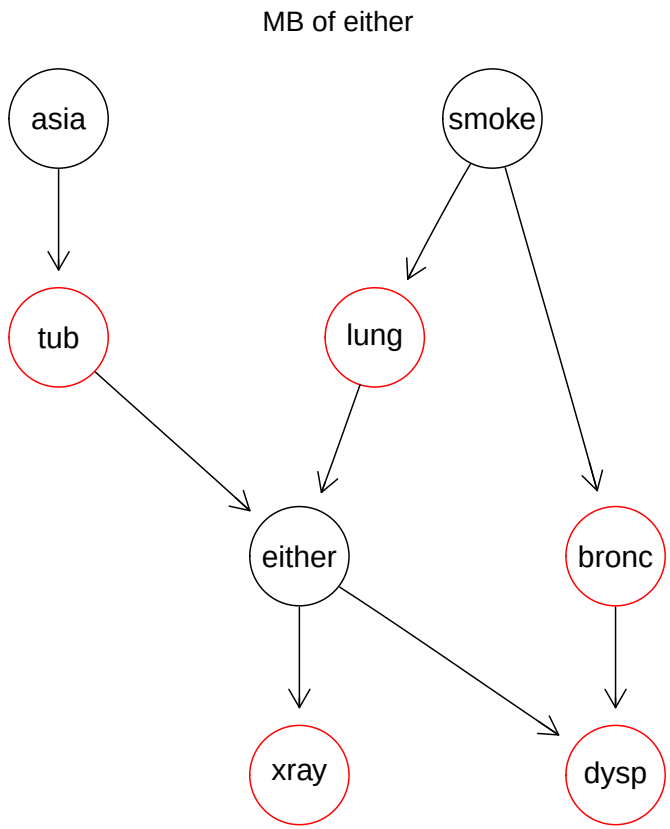
In other words, $MB(X) = \mathbf{Pa}_{\mathcal{G}}(X) \cup \mathbf{Ch}_{\mathcal{G}}(X) \cup \mathbf{Pa}_{\mathcal{G}}(\mathbf{Ch}_{\mathcal{G}}(X))$ and $X \perp\!\!\!\perp \mathbf{X} \setminus MB(X) \mid MB(X)$.

For any inference regarding X it suffices that we know $MB(X)$; we can ignore the remaining variables. Thus, learning a Markov blanket provides a principled approach to feature selection in machine learning.

Markov blanket

Note that, because we have observed the children of X, we need to account for its spouses.





Equivalence classes

Consider the networks $X \leftarrow Z \leftarrow Y$, $X \rightarrow Z \rightarrow Y$, and $X \leftarrow Z \rightarrow Y$. They encode identical independence assumptions: $\{(X \perp Y|Z)\}$. We say that these DAGs are *I-equivalent*.

A P that factorizes over $\mathcal{G}$ has no intrinsic property that allows us to associate it with one graph rather than an equivalent one.

Graphs $X \leftarrow Z \leftarrow Y$, $X \rightarrow Z \rightarrow Y$, and $X \leftarrow Z \rightarrow Y$ same *skeleton*, a necessary condition for different DAGs to enconde the same conditional independences:

> *A skeleton of a Bayesian network $\mathcal{G}$ over $\mathbf{X}$ is an undirected graph over $\mathbf{X}$ that contains an edge $\{X, Y\}$ for every edge (X, Y) in $\mathcal{G}$.*

Equivalence classes

$X \rightarrow Z \leftarrow Y$ has the same skeleton yet encodes different conditional independence relationships: $\{(X \perp Y)\}$. Unlike the above graphs, it contains a *v-structure*: two nodes with a common child that are not connected by an arc.

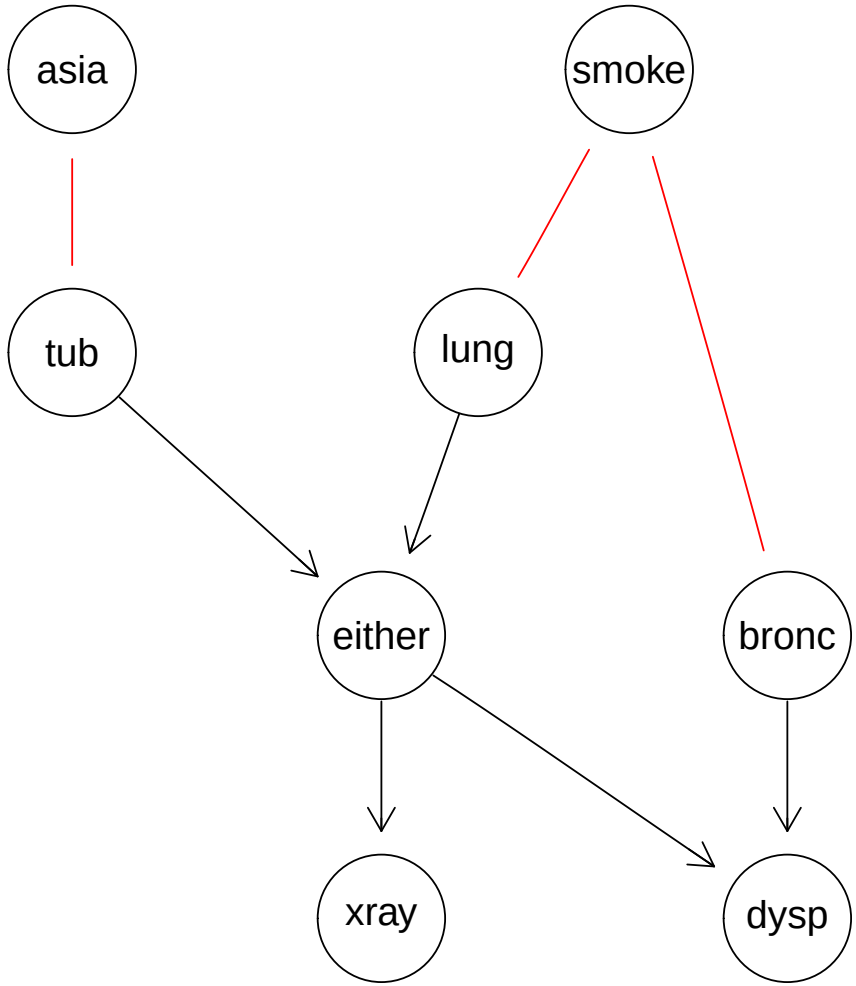
The following result holds: Graphs $\mathcal{G}$ and $\mathcal{H}$ over $\mathbf{X}$ have the same skeleton and the same set of v-structures if and only if they are I-equivalent. The relation of I-equivalence is reflexive, transitive, and symmetric, and partitions the space of DAGSs into a set of equivalence classes.

A completed partially DAG (CPDAG) uniquely identifies a class of equivalent DAGs. The directions of the arcs in the CPDAG can be either:

1. uniquely identifiable because changing them would introduce cycles or change the v-structures (directed arcs in the CPDAG);
2. completely undetermined (undirected arcs in the CPDAG).

Equivalence classes

Below is the CPDAG for the chest clinic. Reversing $A \rightarrow T$, $S \rightarrow L$, or $S \rightarrow B$ would give an identical model to the original one.



From distributions to graphs

Assuming that we have access to conditional independence properties in P, how do we build a DAG $\mathcal{G}$? If our only requirement is that $\mathcal{G}$ be an I-map of P, we may use the complete DAG; it encodes no conditional independence and thus implies all those that hold in P. A more useful notion is that of a *minimal I-map*:

> *A graph $\mathcal{H}$ is a minimal I-map is a for a set of independencies I if it is an I-map for I, and if the removal of even a single edge from $\mathcal{H}$ renders it not an I-map.*

Given a topological ordering of variables in P, we can uniquely determine its minimal I-map DAG (if P is strictly positive). However, we usually do not have an ordering when learning $\mathcal{G}$, and can obtain different DAGs with different orderings.

From distributions to graphs

The figure below shows minimal I-maps for three different orderings for the same P. Often, a more natural, causal ordering, can lead networks with less arcs.

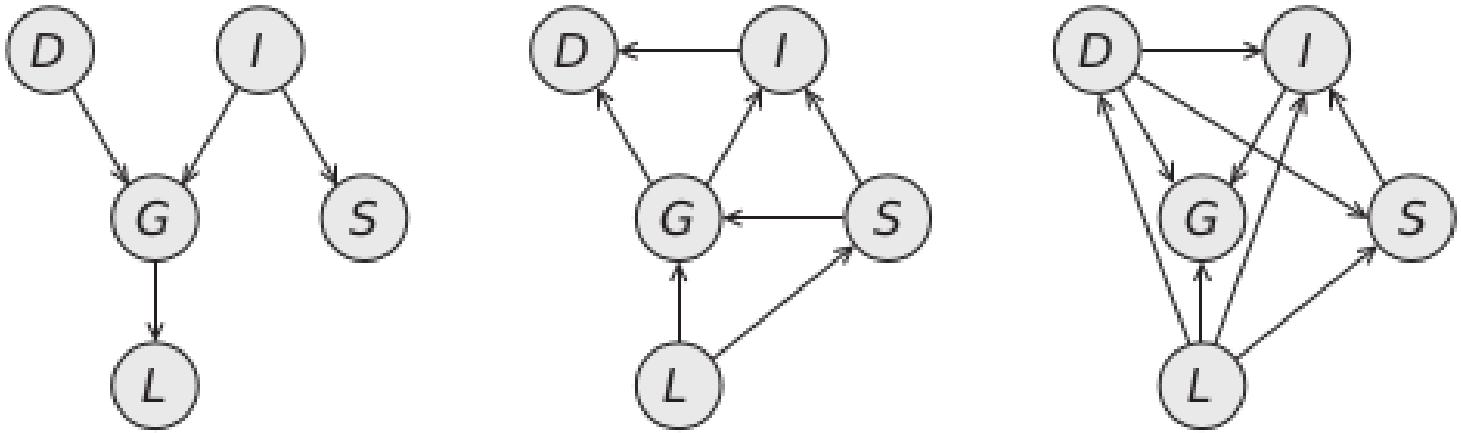
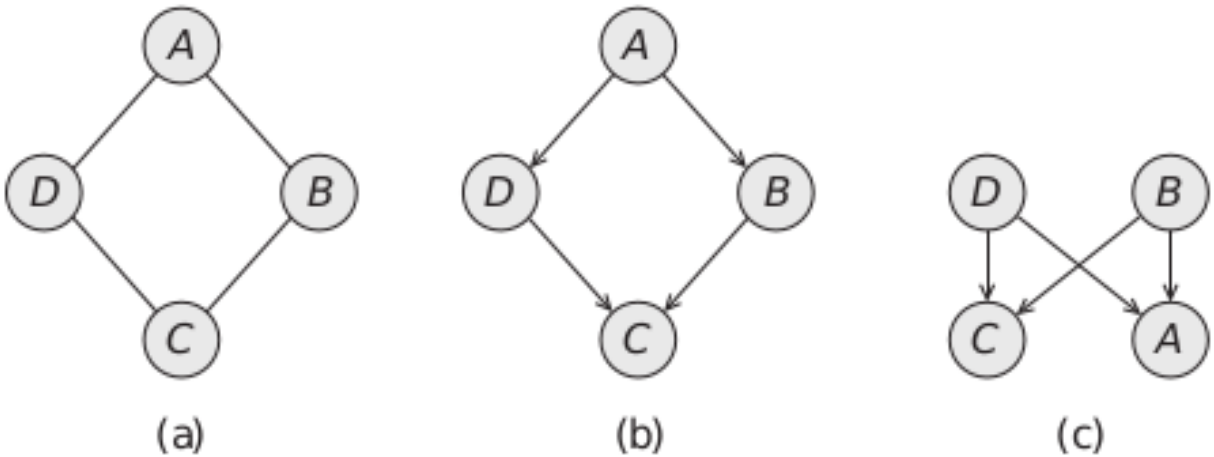


Figure 3.8 Three minimal I-maps for $P_{Student}$, induced by different orderings: (a) D, I, S, G, L; (b) L, S, G, I, D; (c) L, D, S, I, G.

From distributions to graphs

Not all distributions P have a P-map BN. Consider a P with the following conditional independences: $\{(A \perp C|\{B, D\}), (B \perp D|\{A, C\})\}$. Two attempted BNs are shown in the figure:



DAG (b) indeed encodes the independence assumption $(A \perp C|\{B, D\})$. However, it also implies $(B \perp D|\{A\})$ but $(B \not\perp D|\{A, C\})$. Hence, it is not a P-map for P. Neither is the DAG (c).

From distributions to graphs

Thus, some distributions do not have a P-map BN, and encoding an I-map BN for them may result in an overly dense DAG (a complete DAG in the worst case). In particular, symmetric independencies are not naturally expressed with Bayesian networks, since they impose directionality. Such relationships arise naturally in domains such as image processing, where relationships between nearby pixels are symmetric. Such distributions may be better represented with an undirected probabilistic graphical model, a Markov network (the undirected graph in the previous slide).

> *Directed models can encode the independencies associated with immoralities, whereas undirected models cannot; conversely, undirected models can encode a symmetric diamond, whereas directed models cannot.*

Sparse and dense DAGs

A DAG can be more or less dense. An extreme example is the complete DAG, which encodes on conditional independences and thus is no different than specifying the full JPD. Dense graphs are hard to understand, and eliciting parameters can be difficult. Since inference in Bayesian networks depends strongly on DAG connectivity, such DAGs are also more complex to use.

An example of a relatively sparse DAG is a tree.

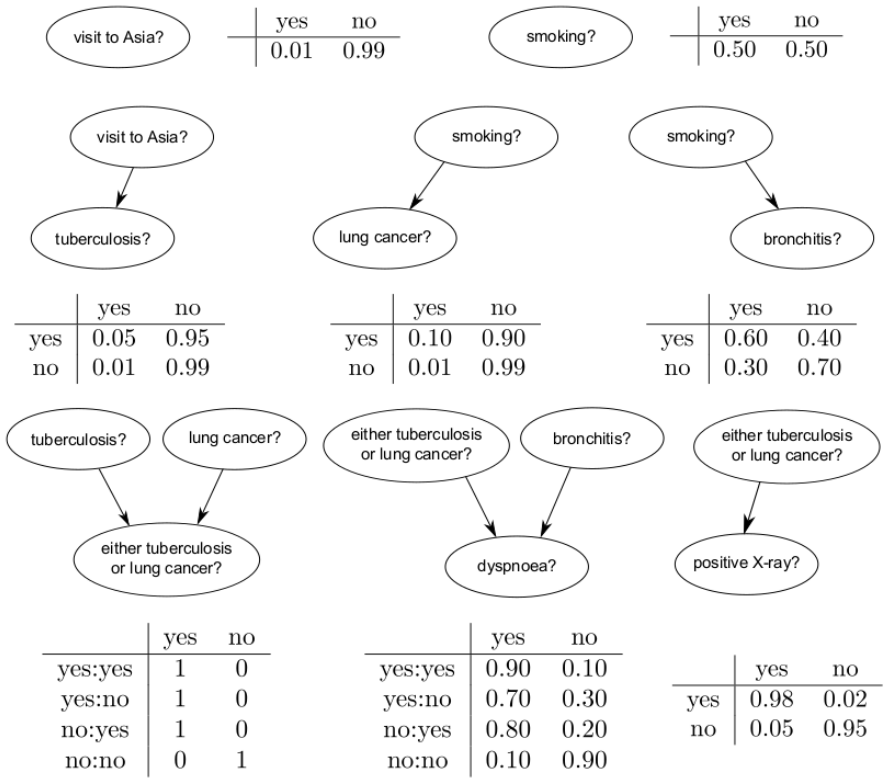
> *A DAG $\mathcal{G}$ is tree-structured if each variable X has at most one parent in $\mathcal{G}$.*

Sparser DAGs are much easier to understand, to inference on, and to elicit or learn parameters for. When learning DAGs from data, this is a form of the bias-variance trade-off: by not modelling some weak influences we may get more robust models.

Local probability distributions

Local probability distributions

The second component of a BN is $\boldsymbol{\theta}$, the set of parameters of the conditional probability distributions associated with the nodes of $\mathcal{G}$. These local probability distributions are also refered to as conditional probability distributions (CPDs). On the right we see the CPDs for the discrete variables in chest clinic.



Conditional probability distributions (CPDs)

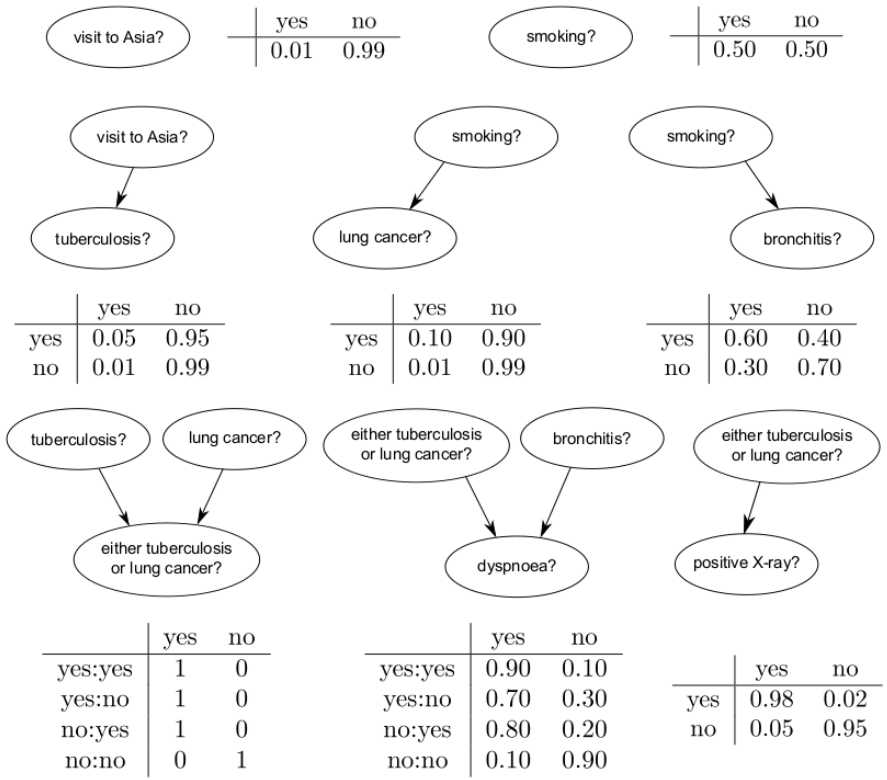
There are three common distributional assumptions in practice:

- Discrete:
  - $\mathbf{X}$ univariate multinomial
  - $X_i \mid \mathbf{Pa}_{\mathcal{G}}(X_i)$ multinomial
- Gaussian:
  - $\mathbf{X}$ multivariate Gaussian
  - $X_i \mid \mathbf{Pa}_{\mathcal{G}}(X_i)$ univariate Gaussian
- Conditional linear Gaussian (CLG):
  - $\mathbf{X}$ mixture of multivariate Gaussians
  - $X_i \mid \mathbf{Pa}_{\mathcal{G}}(X_i)$ multinomial, univariate Gaussian or mixtures of Gaussians

The main reason for these assumptions is that exact inference algorithms are applicable for the above distributions, albeit for a limited class of networks in the CLG case.

Discrete CPDs

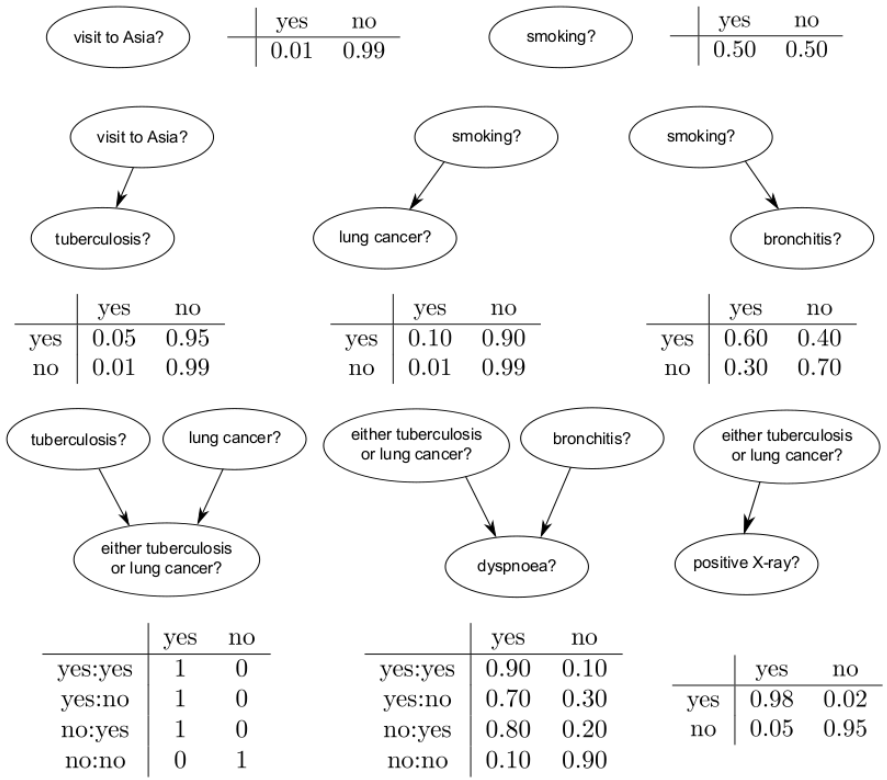
For each node, we have a conditional probability table (CPTs): that is, a multinomial distribution for each configuration of values of the parents. $\theta_{ijk} = p(X_i = k | \mathbf{Pa}(X_i) = \mathbf{pa}_i^j)$: probability that X_i takes value k if its parents take value j.



$$\boldsymbol{\theta}_A = (P(A = a^1) = .01, P(A = a^0) = .99)$$
$$\boldsymbol{\theta}_{T|a^1} = (P(T = t^1 \mid = a^1) = .05, P(T = t^0 \mid a^1) = .95)$$

Discrete CPDs

The number of parameters in a CPT is exponential in the number of parents. Thus, for a binary variable with k binary parents, we have $(2-1) \times 2^k = 2^k$ parameters, because there are $(2-1)$ free parameters in each of the 2^k multinomials. This indicates that the parents determines the statistical complexity (i.e., the number of parameters) of a network.



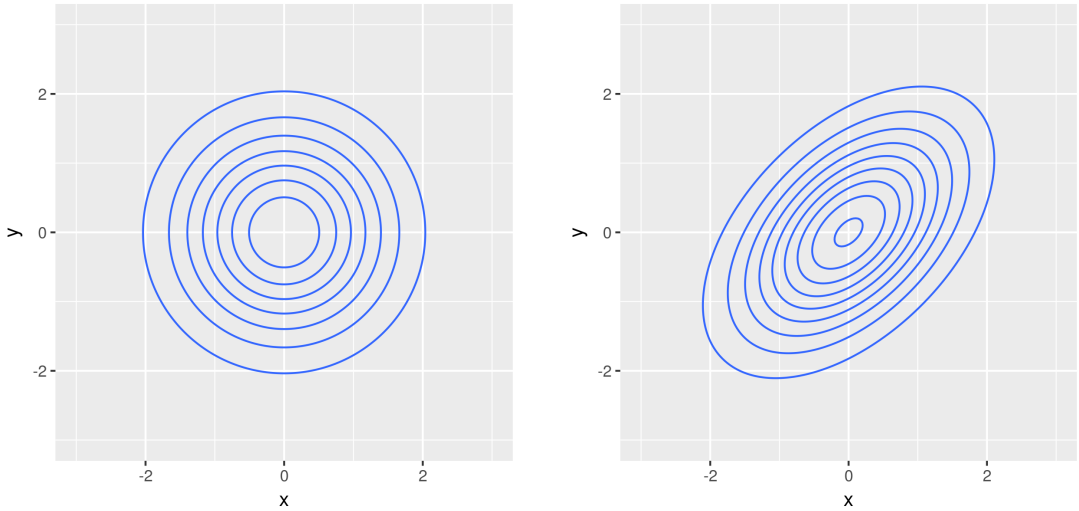
Continuous variables

Many variables take values in a continuous space: e.g., speed, distance, price. In principle, we can use any parametric form to describe a local distribution. However, arbitrary representations of densities are usually not closed under factor marginalization and multiplication, and are thus not useful for Bayesian networks. One exception are Gaussian distributions, where conditional and marginal distributions are also Gaussians. Hence, a common way of dealing with continuous variables is to assume they are Gaussian.

An alternative approach is to discretize continuous variables and model them with multinomial distributions.

Multivariate Gaussian (MVN) distributions

Let $\mathbf{X} \sim N(\boldsymbol{\mu}, \Sigma)$ with covariance matrix Σ. The distribution specifies ellipsiodal contours around $\boldsymbol{\mu}$. Each marginal distribution is a Gaussian and the dependencies among variables are linear. The figures show distributions with mean $\mathbf{0}$ and zero (left) and positive (right) covariances.



$X_i \perp X_j$ if and only if $\Sigma_{ij} = 0$, whereas $X_i \perp X_j | \mathbf{X} \setminus \{X, X_j\}$ if and only if $\Sigma^{-1}_{ij} = 0$.

Multivariate Gaussian Bayesian networks (GBN)

An MVN can be equivalently represented with n linear regression models. Namely, each conditional distribution $P(X \mid \mathbf{pa}(\mathbf{x}))$ is a linear regression with an independent error term:

$$p(X|\mathbf{pa}(\mathbf{x})) = \mathcal{N}\left(\beta_0 + \boldsymbol{\beta}^T \mathbf{pa}(\mathbf{x}); \sigma^2\right)$$

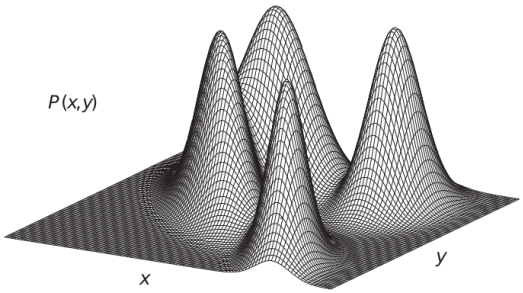
The partial correlation $\Sigma_{ij}^{-1} \propto \beta_j$ for X_i. The Bayesian network can make explicit independencies that are visible in Σ_{ij}^{-1}, i.e., such that $X_i \not\perp X_j|\mathbf{X}\setminus\{X, X_j\}$ but $X_i \not\perp X_j|\mathbf{U}$ for some $\mathbf{U} \subset \mathbf{X}\setminus\{X, X_j\}$.

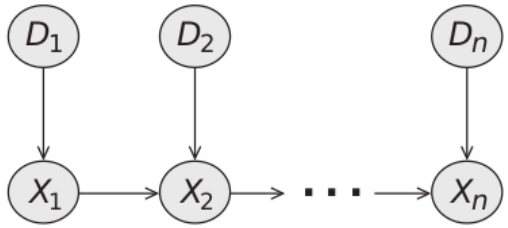
The number of parameters is at most quadratic in n whereas in multinomial distributions it is exponential in number of parents. This suggests that the Gaussian model is less expressive.

Conditional linear Gaussian (CLG) networks

Contains both multinomial and Gaussian nodes. Gaussian nodes cannot be parents of multinomial ones. For each combination of discrete parents, we have a linear regression model with continuous parents as regressors.

The CLG distribution is a mixture of Gaussians. For k binary discrete variables, we may have 2^k Gaussians. A mixture of Gaussians with sufficient components can approximate an arbitrary distribution.





Continuous variables

- Strong asssumption of marginal normal distributions and linear dependence relationships;
- Discretisation may discard useful information, but it allows for non-linear dependencies;
- The CLG model adds constraints, as a continuous node cannot be the parent of a discrete one.

Hands-on

Quiz

Let us try an specify a Bayesian network for the Monty Hall puzzle.

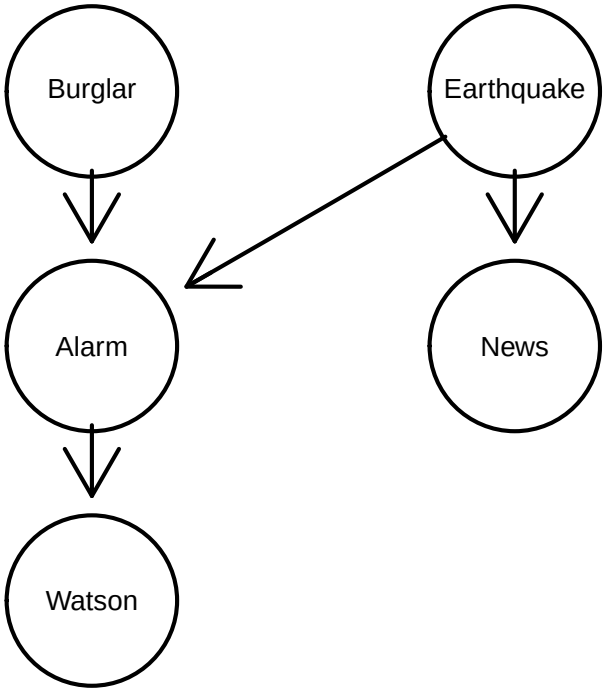
The Monty Hall puzzle gets its name from an american TV game show, "Let's make a deal", hosted by Monty Hall. In this show, you have the chance to win some prize if you are lucky enough to find the prize behind one of three doors. The game goes like this:

- You are asked to select one of the three doors.
- Monty Hall (who knows where the prize is) opens another door which does not contain the prize.
- You are asked if you want to redo your selection (select one of the two closed doors).
- You get the prize if it is behind the door you selected.

The earthquake network

Mr. Holmes is working in his office when he receives a phone call (W) from his neighbor, who tells him that Holmes' burglar alarm (A) has gone off. Convinced that a burglar has broken into his house (B), Holmes rushes to his car and heads for home. On his way, he listens to t he radio, and in the news it is reported (N) that there has been a small earthquake (E) in the area. Knowing that earthquakes have a tendency to turn on burglar alarms, he returns to work.

The earthquake network



bnlearn Bayesian network models

We will look at how bnlearn represents a Bayesian network and the basic operations we can perform on it. We start a clean R session and load the bnlearn package

```
library(bnlearn)
# help(package='bnlearn')
```

Load utility functions that we will use.

```
source('functions.R')
```

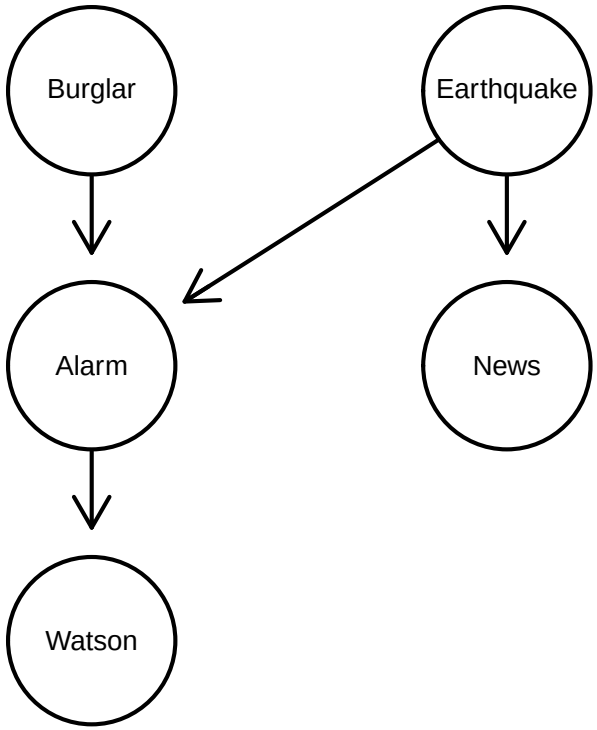
bnlearn provides two S3 classes for Bayesian networks:

- "bn": the DAG
- "bn.fit": a fully specified network with parameters

bnlearn functionalities

A compact way to specify a "bn" is the model string:

```
string <- paste('[Burglar][Earthquake][Alarm|Burglar:Earthquake]',
        '[News|Earthquake][Watson|Alarm]', sep = '')
bl.alarm <- model2network(string )
plot(bl.alarm)
```



bnlearn functionalities

We can add, remove or reverse arcs, with the acyclic constraint enforced:

```
modified.alarm <- drop.arc(bl.alarm, "Earthquake", "News")
modified.alarm <- reverse.arc(modified.alarm , "Alarm", "Burglar")
modified.alarm <- set.arc(modified.alarm, "Earthquake", "Burglar")
# plot(modified.alarm)
```

bnlearn enforces the acyclicity constraint:

```
modified.alarm <- set.arc(modified.alarm, "Watson", "Earthquake")
```

```
## Error in arc.operations(x = x, from = from, to = to, op = "set",
```

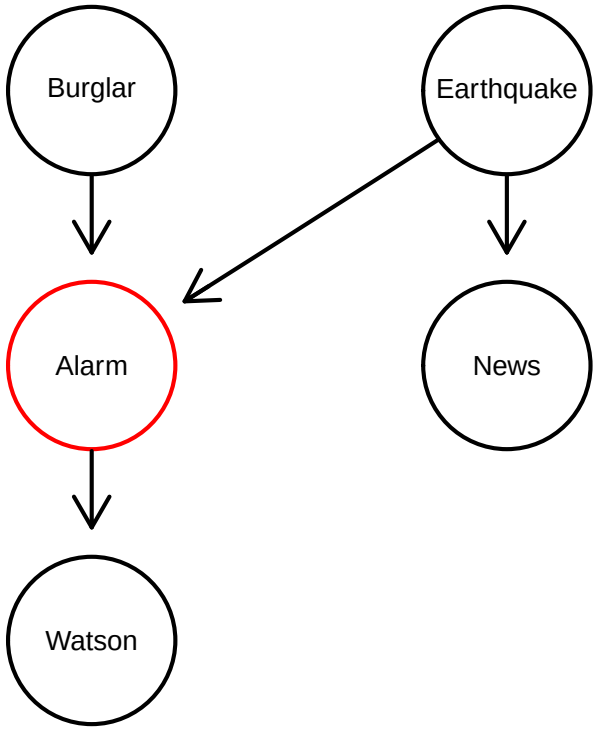
bnlearn functionalities

Printing the object to console provides basics information:

```
bl.alarm
```

We can customize the network plot in `plot()` to some degree.

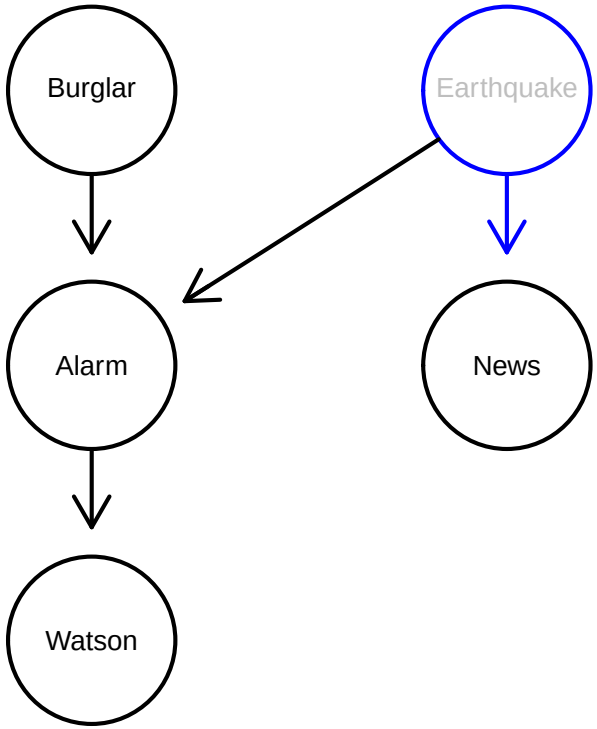
```
plot(bl.alarm, highlight = list(nodes='Alarm'))
```



bnlearn functionalities

Or use `Rgraphviz` for more advanced options:

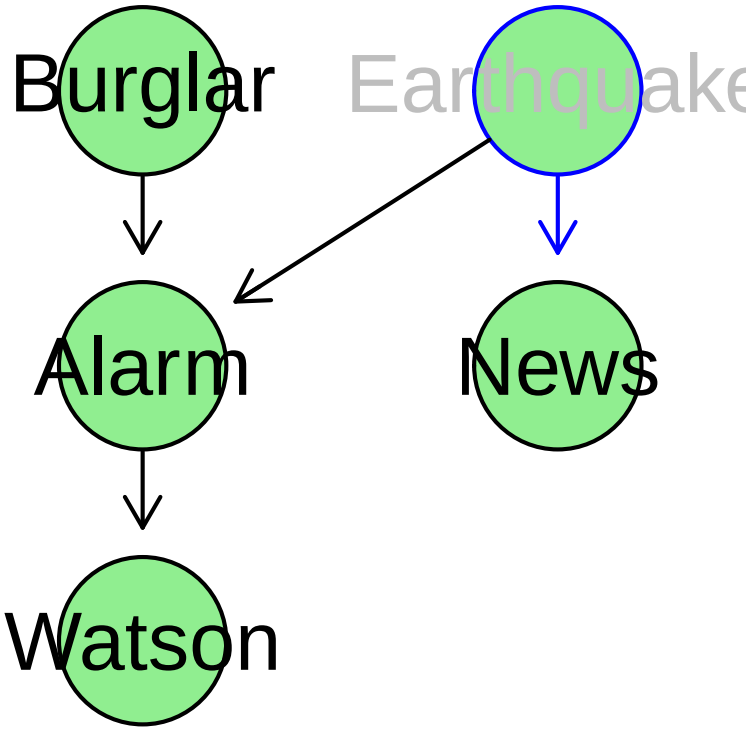
```
hlight <- list(nodes = c("Earthquake"), arcs = c("Earthquake", "News
pp <- graphviz.plot(bl.alarm, highlight = hlight, render = FALSE)
Rgraphviz::renderGraph(pp)
```



bnlearn functionalities

Setting font size can sometimes be useful.

```
graph::nodeRenderInfo(pp) <- list(fill="lightgreen", fontsize=40)
Rgraphviz::renderGraph(pp)
```



bnlearn functionalities

When the network is too large for plotting, the model string can be useful. We can also ask about the nodes close to a particular node:

```
nbr(bl.alarm, node = 'Alarm')
```

```
## [1] "Burglar"    "Earthquake" "Watson"
```

```
parents(bl.alarm, node = 'Alarm')
```

```
## [1] "Burglar"    "Earthquake"
```

```
children(bl.alarm, node = 'Alarm')
```

```
## [1] "Watson"
```

```
mb(bl.alarm, node = 'Alarm')
```

```
## [1] "Burglar"    "Earthquake" "Watson"
```

bnlearn functionalities

We can list all arcs or ask whether there is a path between a pair of nodes:

```
arcs(bl.alarm)
```

```
##       from          to
## [1,] "Burglar"    "Alarm"
## [2,] "Earthquake" "Alarm"
## [3,] "Earthquake" "News"
## [4,] "Alarm"      "Watson"
```

```
path(bl.alarm, from = "Burglar", to = "Watson")
```

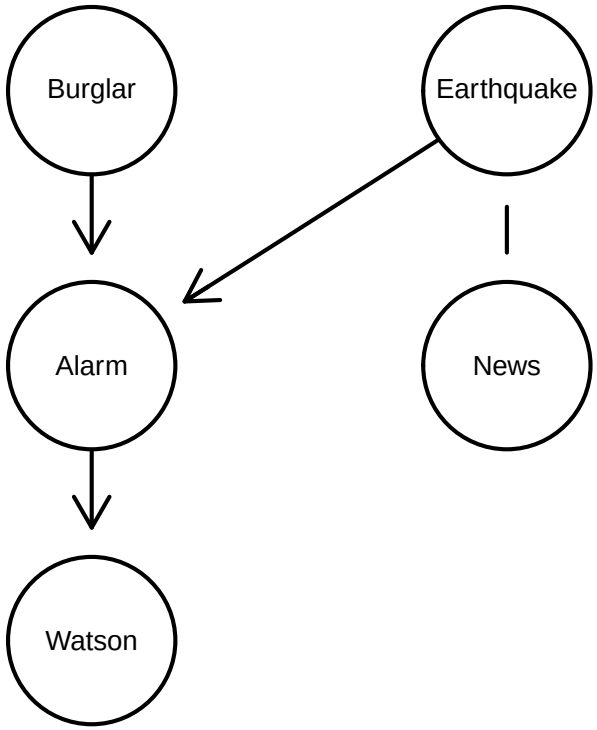
```
## [1] TRUE
```

```
path(bl.alarm, from = "Watson", to = "Burglar")
```

```
## [1] FALSE
```

bnlearn functionalities

```
dsep(bl.alarm, 'Watson', 'News')
dsep(bl.alarm, 'Watson', 'News', 'Alarm')
plot(cpdag(bl.alarm))
```




```
## [1] FALSE
## [1] TRUE
```

bnlearn functionalities

Passing a list of local distributions to `custom.fit()` produces a `"bn.fit"`. With discrete variables, we specify a CPT for each node:

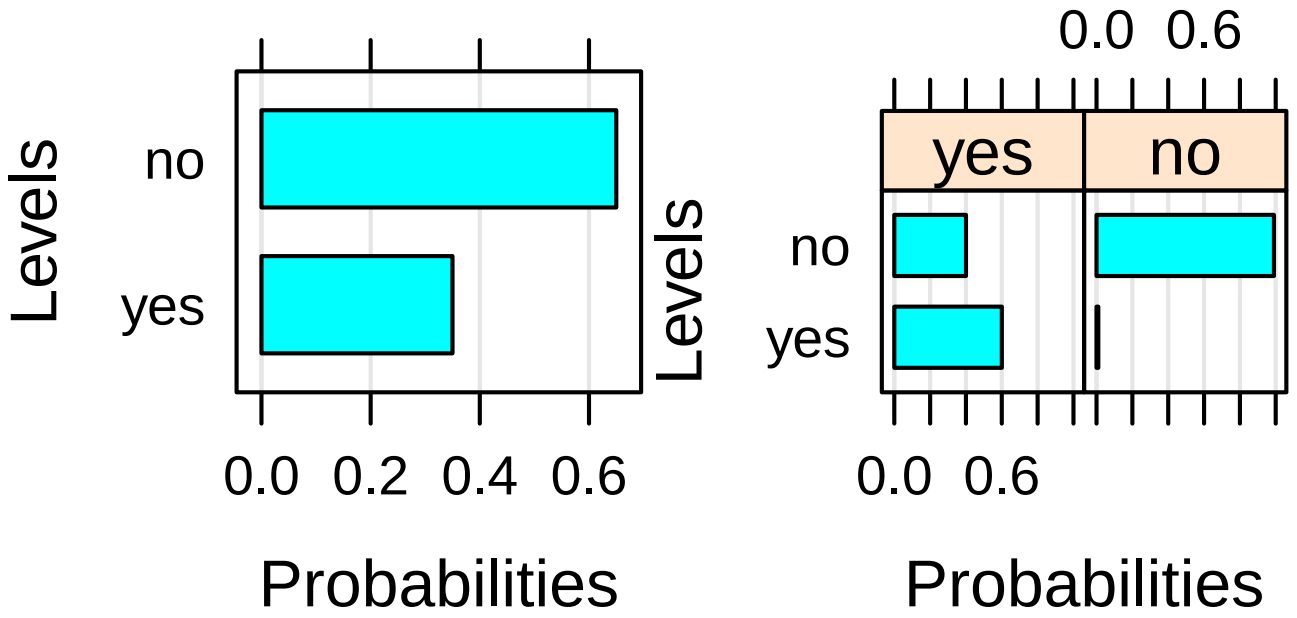
```
yn <- c("yes","no")
B  <- array(dimnames = list(Burglar = yn), dim = 2, c(0.30,0.70))
E  <- array(dimnames = list(Earthquake = yn), dim = 2, c(0.35,0.65))
A  <- array(dimnames = list(Alarm = yn, Earthquake = yn,
                            Burglar = yn), dim = c(2, 2, 2),
            c(0.95,0.05,0.90,0.10,0.60,0.40,0.01,0.99))
W  <- array(dimnames = list(Watson = yn, Alarm = yn), dim = c(2, 2))
N  <- array(dimnames = list(News = yn, Earthquake = yn), dim = c(2,
cpts <- list(Burglar = B, Earthquake = E, Alarm = A, Watson = W, New
bl.alarm.fit = custom.fit(bl.alarm, cpts)
```

bnlearn functionalities

```
bl.alarm.fit$Earthquake$prob
```

```
## Earthquake
##  yes   no
## 0.35 0.65
```

```
bn.fit.barchart(bl.alarm.fit$Earthquake, main = '')
bn.fit.barchart(bl.alarm.fit$News, main = '')
```



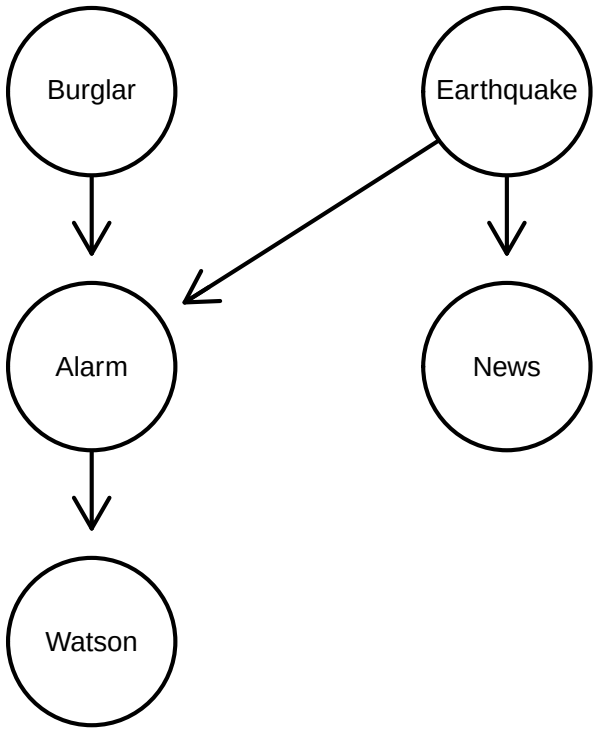
bnlearn functionalities

In order to plot `bn.fit`, we convert it to a `bn` with `bn.net()`.

```
plot(bl.alarm.fit) # Fails
```

```
## Error in xy.coords(x, y, xlabel, ylabel, log): 'x' is a list, bu
```

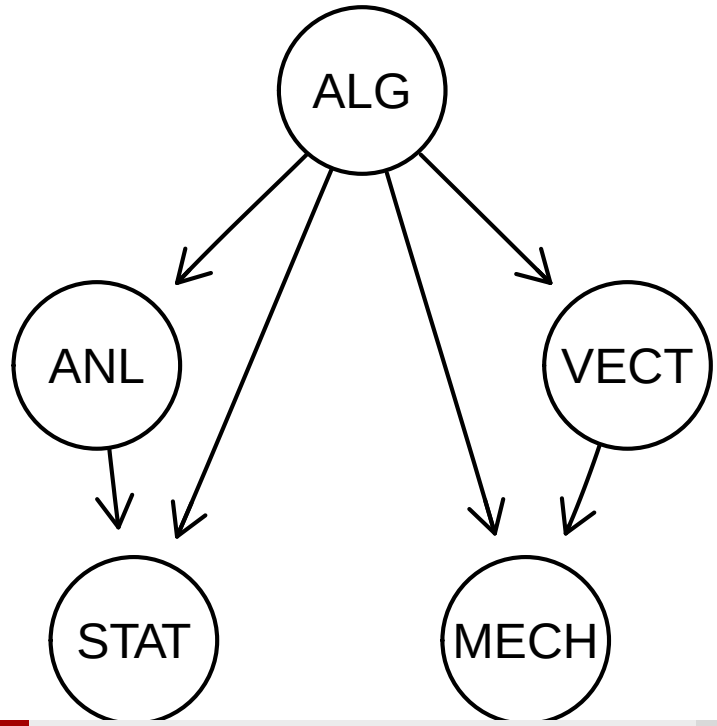
```
plot(bn.net(bl.alarm.fit))
```



The marks model

The following is a model of students' scores, from 0 to 100, in mechanics, vectors, algebra, analysis and statistics, from Mardia, Kent & Bibby (1979).

```
string <- paste("[ALG][ANL|ALG][MECH|ALG:VECT]",
          "[STAT|ALG:ANL][VECT|ALG]" , sep = "")
marks.dag = model2network(string)
plot(marks.dag)
```



bnlearn functionalities

Specify the regression coefficients and the standard deviation of the residuals for the local models. The call `custom.fit()`.

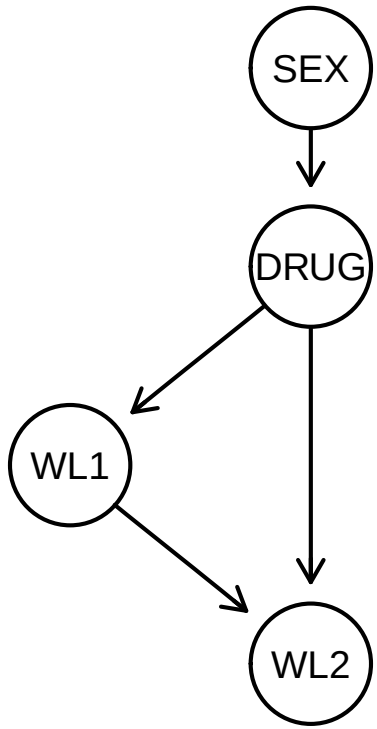
```
ALG.dist = list(coef = c("(Intercept)" = 50.60), sd = 10.62)
ANL.dist = list(coef = c("(Intercept)" = -3.57, ALG = 0.99), sd = 1
MECH.dist = list(coef = c("(Intercept)" = -12.36, ALG = 0.54, VECT =
STAT.dist = list(coef = c("(Intercept)" = -11.19, ALG = 0.76, ANL =
VECT.dist = list(coef = c("(Intercept)" = 12.41, ALG = 0.75), sd =
ldist = list(ALG = ALG.dist, ANL = ANL.dist, MECH = MECH.dist,
STAT = STAT.dist, VECT = VECT.dist)
marks.fit = custom.fit(marks.dag, ldist)
marks.fit[c("MECH", "STAT")]$MECH$coefficients
```

```
## (Intercept)         ALG        VECT
##      -12.36        0.54        0.46
```

The rats data

For conditional linear Gaussian models, consider the rats network, with discrete variables DRUG and SEX and continuous WL1 and WL2, corresponding to weight loss in week one and week two:

```
rats.dag = model2network("[SEX][DRUG|SEX][WL1|DRUG][WL2|WL1:DRUG]")
plot(rats.dag)
```



bnlearn functionalities

For the discrete variables, we have CPTs.

```
SEX.lv = c("M", "F")
DRUG.lv = c("D1", "D2", "D3")
SEX.prob = array(c(0.5, 0.5), dim = 2,
                 dimnames = list(SEX = SEX.lv))
DRUG.prob = array(c(0.3333, 0.3333, 0.3333,
                    0.3333, 0.3333, 0.3333),
dim = c(3, 2), dimnames =
  list(DRUG = DRUG.lv, SEX = SEX.lv))
```

bnlearn functionalities

For Gaussian variables, we have a linear regression for each combination of discrete parents. For WL1 the only coefficient is the intercept, since it has no Gaussian parents.

```
WL1.coef = matrix(c(7, 7.50, 14.75), nrow = 1, ncol = 3,
dimnames = list("(Intercept)", NULL))
WL1.dist = list(coef = WL1.coef, sd = c(1.58, 0.447, 3.31))
```

```
WL2.coef = matrix(c(1.02, 0.89, -1.68, 1.35, -1.83, 0.82),
                  nrow = 2, ncol = 3,
dimnames = list(c("(Intercept)", "WL1")))
WL2.dist = list(coef = WL2.coef, sd = c(1.78, 2, 1.37))
```

bnlearn functionalities

Columns corresponds the to value of the discrete parent DRUG, row to the coefficients (the intercept and one for each continuous parent).

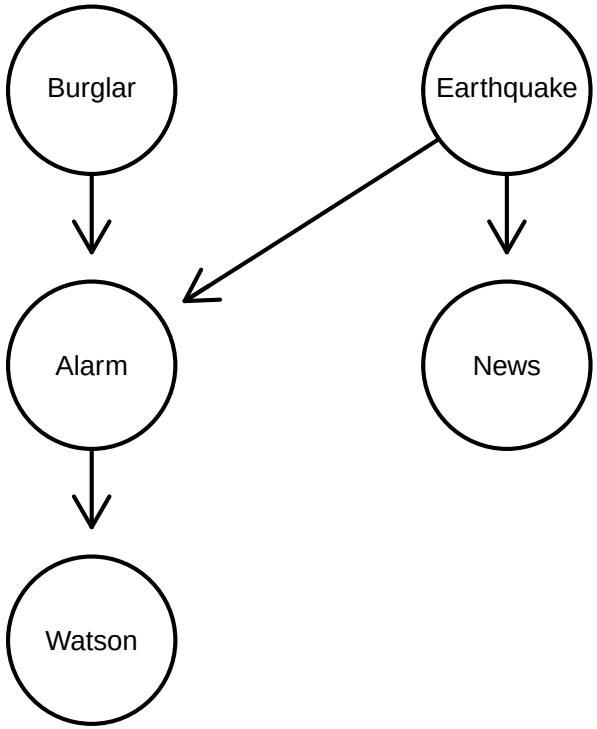
```
ldist = list(SEX = SEX.prob, DRUG = DRUG.prob, WL1 = WL1.dist, WL2 =
rats.fit = custom.fit(rats.dag, ldist)
rats.fit$WL2$coefficients
```

```
##                   0     1     2
## (Intercept) 1.02 -1.68 -1.83
## WL1          0.89  1.35  0.82
```

The earthquake network

We will analyze the DAG for earthquake in detail.

```
plot(bl.alarm)
```



Which node has most free parameters associated? How many? How many parameters in total? Compute it by hand knowing that all variables are binary.

The earthquake network

How many parameters in a full joint? Where does the reduction come from?

What are the v-structures in the network?

```
vstructs(bl.alarm)
```

```
##      X          Z       Y
## [1,] "Burglar" "Alarm" "Earthquake"
```

State two independencies that hold in absence of evidence. Which probability queries would let us verify them?

The earthquake network

- Looking at the network, do you think these hold:
  - $I(B; W \mid A)$?
  - $I(B; N \mid A)$?
- Why/why not?

Check the above using the `dsep()` function.

```
dsep(bn=bl.alarm, x='Burglar', y=, z=)
```

- Is Watson independent of News given Earthquake $I(N; W \mid E)$?
- And the other way around $I(W; N \mid E)$?
- Why/why not?

The earthquake network

Can you identify the Markov blanket of Earthquake? Try verifying it by checking d-separation.

```
mb(x=bl.alarm, node='Earthquake')
mbe <- subgraph(bl.alarm, nodes = mb(x=bl.alarm, node='Earthquake'))
plot(mbe)
```

The following function with plot all the Markov blankets.

```
par(mfrow=c(2, 3))
print_mblankets(bl.alarm)
par(mfrow=c(1, 1))
```

Given Burglar, News, and Alarm, is Earthquake independent of Watson?

The earthquake network

Different networks can encode same conditional independencies.

- How could we obtain an equivalent DAG?
- How many DAGs in this equivalence class?
- Is News independent of Watson given Earthquake ($I(N; W \mid E)$) in this equivalent DAG?
- Are Markov blankets the same in the equivalent DAG?

We can obtain a representative model of the equivalence class, a complete partially directed acyclic graph, with `cpdag()`

```
cpdag.alarm <- cpdag(bl.alarm)
plot(cpdag.alarm)
```

The earthquake network

We will check d-separation by performing queries on the network. We will use the `gRain` package for exact inference. We can convert the `bnlearn` objects to `gRain` ones. To keep our fully specified network, we will use `bl.alarm.fit` rather than `bl.alarm`.

```
library(gRain)
gr.alarm <- as.grain(bl.alarm.fit)
```

The earthquake network

Alternatively, we can construct the Earthquake network (specify its DAG and conditional probability tables (CPTs)) with the `grain` package, similarly to `bnlearn`. The network below is identical to `gr.alarm`.

```
yn <- c("yes","no")
B  <- cptable(~Burglar, values=c(30,70),levels=yn)
E  <- cptable(~Earthquake, values=c(35,65),levels=yn)
A  <- cptable(~Alarm+Earthquake+Burglar, values=c(95,5,90,10,60,40,
W  <- cptable(~Watson+Alarm, values=c(80,20,40,60),levels=yn)
N  <- cptable(~News+Earthquake, values=c(60,40,1,99), levels=yn)
```

```
cptlist <- compileCPT(list(B, E, A, W, N))
gr.alarm.orig <- grain(cptlist)
```

Plot the network.

```
plot(gr.alarm.orig)
```

The earthquake network

- What is marginal distribution over Earthquake ($P(E)$)?

```
gr.alarm.orig$cptlist$Earthquake
```

```
## Earthquake
##  yes   no
## 0.35 0.65
## attr(,"class")
## [1] "parray" "array"
```

- How does this relate to the CPT specification above?
- What is $P(B)$?

The earthquake network

- What is the probability that nothing occurs $P(B = b^0, E = e^0, A = a^0, W = w^0, N = n^0)$? Compute it using the CPTs and the chain rule.

```
bn <- gr.alarm.orig$cptlist$Burglar['no']
en <- gr.alarm.orig$cptlist$Earthquake['no']
an <- gr.alarm.orig$cptlist$Alarm['no', 'no', 'no']
nn <- gr.alarm.orig$cptlist$News['no', 'no']
wn <- gr.alarm.orig$cptlist$Watson['no', 'no']
bn * en * an * nn * wn
```

Instead of us summing and multiplying the probabilities in the CPTs, the `grain` object does inference for us.

The earthquake network

First, we need to *compile* the `grain` network.

```
gr.alarm <- compile(gr.alarm.orig) # Compile first
gr.alarm
```

```
## Independence network: Compiled: TRUE Propagated: FALSE
##   Nodes: chr [1:5] "Burglar" "Earthquake" "Alarm" "Watson" "News"
```

The earthquake network

Now we can query the network:

```
querygrain(object=gr.alarm, nodes="Earthquake")
```

```
## $Earthquake
## Earthquake
##   yes    no
## 0.35 0.65
```

```
querygrain(object=gr.alarm, nodes="Burglar")
```

```
## $Burglar
## Burglar
## yes  no
## 0.3 0.7
```

The earthquake network

Use `setEvidence()` and `pEvidence()` to get the probability that nothing occurs. Complete the missing arguments. What is the probability?

```
no <- rep("no", 5)
nodes <- c('Burglar', 'Earthquake', 'Alarm', 'Watson', 'News')
gr.alarm <- setEvidence(object=gr.alarm, nodes=, states=)
pEvidence(gr.alarm)
```

The earthquake network

Finally, we will retract the evidence to return the network to initial state:

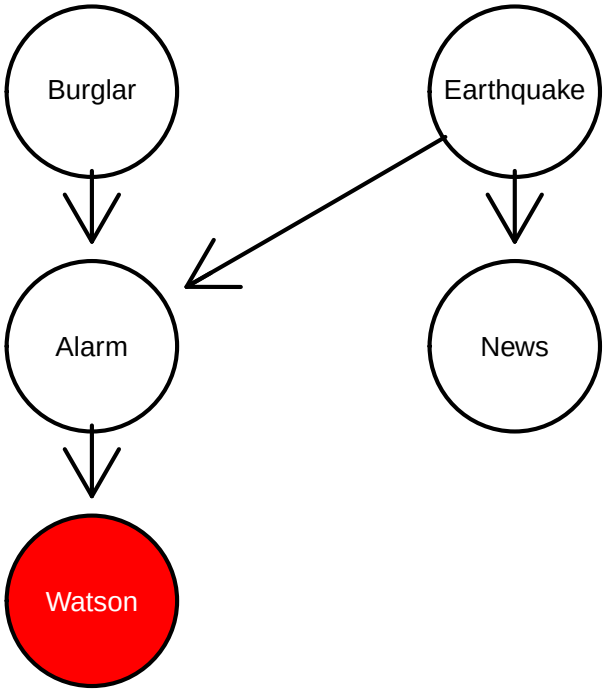
```
gr.alarm
```

```
## Independence network: Compiled: TRUE Propagated: TRUE
##   Nodes: chr [1:5] "Burglar" "Earthquake" "Alarm" "Watson" "News"
##   Evidence:
##        nodes is.hard.evidence hard.state
## 1    Burglar             TRUE         no
## 2 Earthquake             TRUE         no
## 3      Alarm             TRUE         no
## 4     Watson             TRUE         no
## 5       News             TRUE         no
##   pEvidence: 0.267567
```

```
gr.alarm <- retractEvidence(gr.alarm, nodes)
# gr.alarm
```

The earthquake network

Watson called. Set that as evidence.



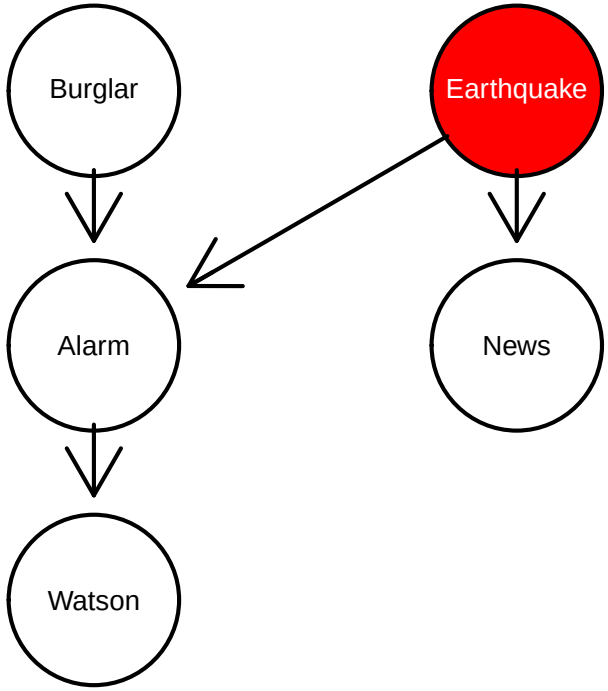
The earthquake network

- Is earthquake more likely now ($P(e^1 \mid w^1) > P(e^1)$)?
- Is burglary more likely ($P(b^1 \mid w^1) > P(b^1)$)?
- Which reasoning pattern was this?

Before continuing, retract the evidence on Watson.

The earthquake network

Now, let us consider that an earthquake has occurred. Set it as evidence.

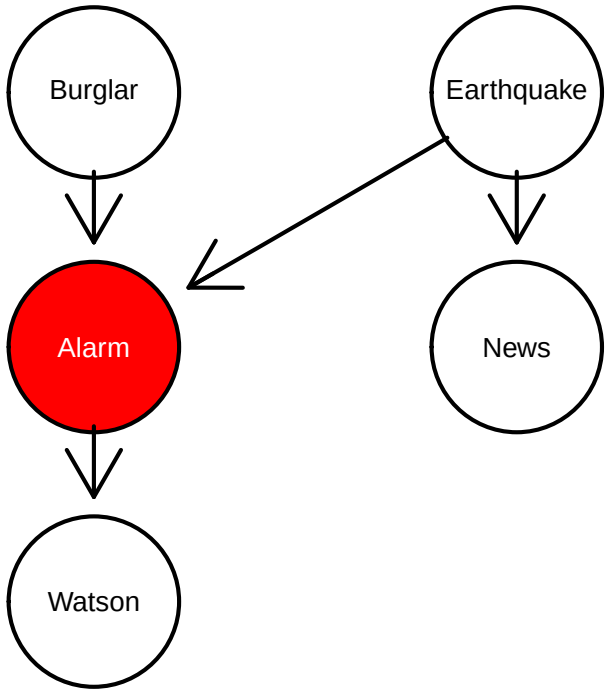



```
gr.alarm <- setEvidence(object=gr.alarm, nodes="Earthquake", states=
```

- Is Watson more likely to call ($P(w^1 \mid e^1) > P(w^1)$)?
- Is burglary more likely now ($P(b^1 \mid e^1) > P(b^1)$)? Why/why not?
- Which reasoning pattern did we apply?

The earthquake network

Now consider only the alarm went off.

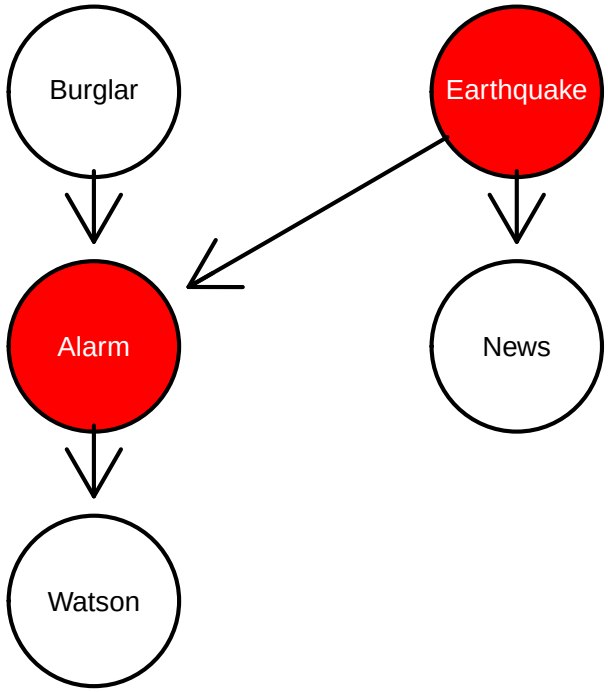



```
gr.alarm <- retractEvidence(gr.alarm, nodes="Earthquake")
gr.alarm <- setEvidence(object=gr.alarm, nodes="Alarm", states="yes"
```

Is burglary more likely now ($P(b^1 \mid a^1) > P(b^1)$)?

The earthquake network

Now, we also learn that an earthquake occured.



- What about burglary now ($P(b^1 \mid a^1) < P(b^1 \mid a^1, e^1)$)?
- Which reasoning pattern was this?

The earthquake network

What is the most likely state for `W`, `B` and `N` given the evidence on earthquake and alarm? ($\arg\max_{w,b,n} P(W, B, N|e^1, a^1)$)?

Get the joint posterior over the corresponding nodes.

```
q.wnb <- querygrain(gr.alarm, type = "joint", nodes = )
q.wnb
```

```
##          Watson         yes                    no
##          News           yes          no        yes          no
## Burglar
## yes              0.19404255 0.12936170 0.04851064 0.03234043
## no               0.28595745 0.19063830 0.07148936 0.04765957
```

What is the most likely state? What is its probabillity?

No specialized algorithm in `gRain` for finding a MAP assignment; we need infer a joint distribution over the nodes of interest. Thus, only feasible for a marginal MAP over few variables.

Table of Contents

Queries and inference

Given a Bayesian network, we are generally interested in posterior probabilities or densities or variables of interest $\mathbf{Q}$, given some evidence $\mathbf{E} = \mathbf{e}$. Then $\mathbf{X} = \mathbf{Q} \cup \mathbf{E} \cup \mathbf{Z}$, where $\mathbf{Z}$ are neither evidence nor of interest, and that we need to marginalize out.

The questions we ask about the probabilities of interest are called queries and the process of computing them is called inference. A query corresponds either to an event, such as `smoker = "yes"` or to a distribution over a set of variables.

When $\mathbf{E} = \emptyset$, we talk of prior probabilities.

Main types of queries

Marginal posterior probability distribution

$$\mathrm{P}(Q|\mathbf{E}=\mathbf{e}) = \sum_{Z} \mathrm{P}(Q, Z|\mathbf{e})$$

Posterior probability of an event

$$\mathrm{P}(e) = \sum_{Z} \mathrm{P}(e, Z)$$

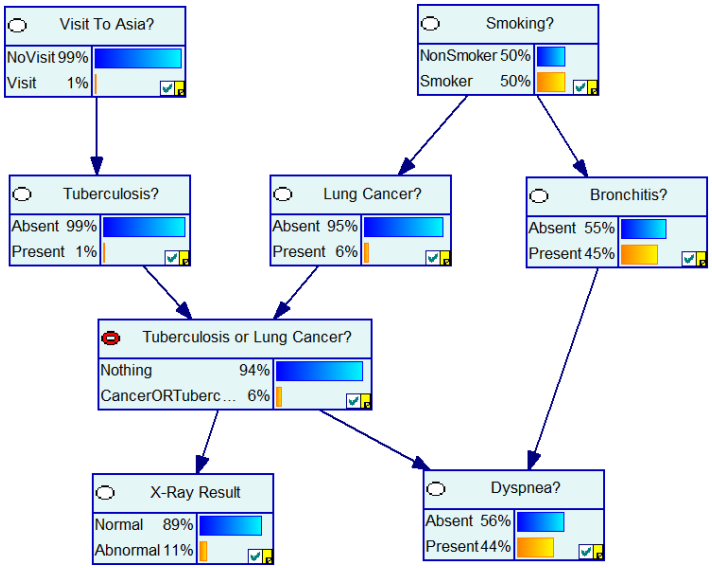
$$\mathrm{P}(q \mid e) = \sum_{Z} \mathrm{P}(q, Z \mid e)$$

These are queries that we already saw on the earthquake network.

Main types of queries

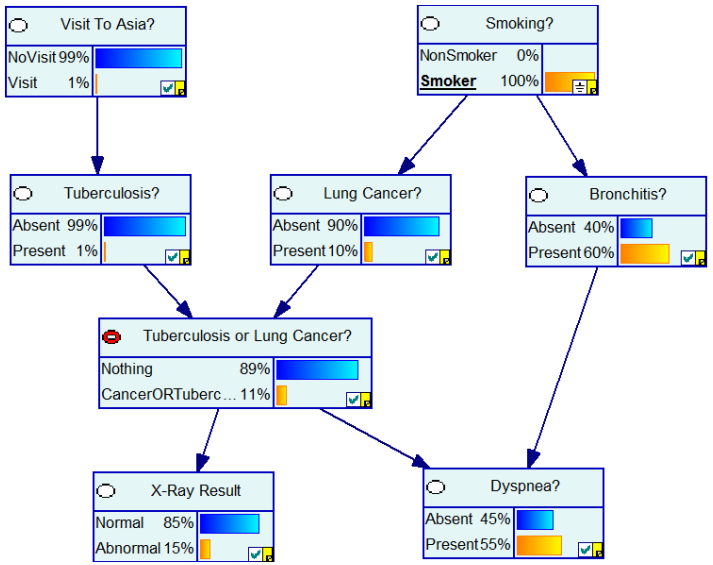
$\mathbf{E} = \emptyset$

(no evidence)



$\mathbf{E} = \{S = s^1\}$

(the patient is a smoker)



$$P(D) = \sum_{a,s,t,l,b,e,x} P(a,s,t,l,b,e,x)$$

$$= \begin{cases} .56 & d^0 \\ .44 & d^1 \end{cases}$$

$$P(D \mid s^1) = \sum_{a,t,l,b,e,x} P(a,s^1,t,l,b,e,x)$$

$$= \begin{cases} .45 & d^0 \\ .55 & d^1 \end{cases}$$

Main types of queries

Maximum a posteriori (MAP) or *most probable explanation* (MPE)

$$q^* = \arg\max_Q \mathrm{P}(Q \mid e),$$

the most likely assignment to the non-evidence variables (All non-evidence variables are of interest, $\mathbf{Z} = \emptyset$).

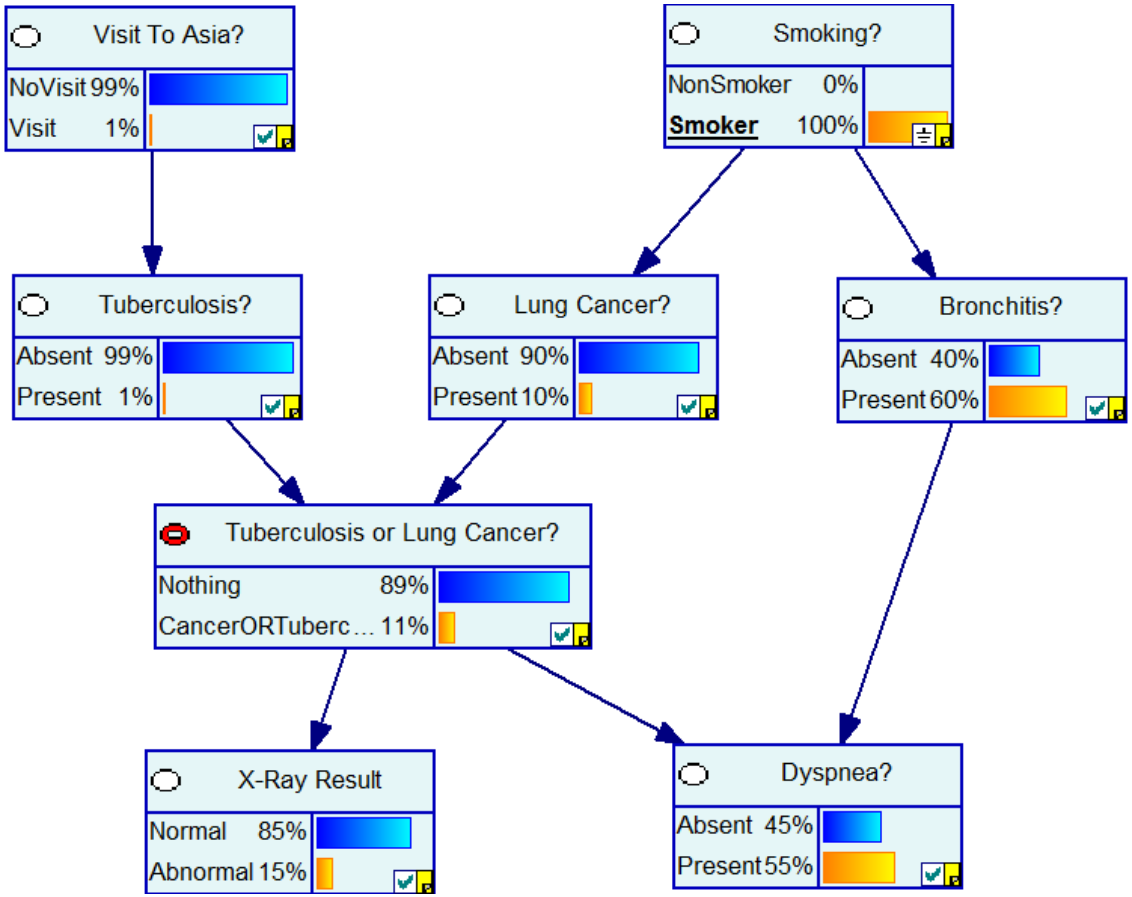
Marginal MAP

$$q^* = \arg\max_Q \sum_Z \mathrm{P}(Q, Z|e),$$

a more general query than MAP, as it allows for evidence.

A common use of MAP and marginal MAP is to fill in the values of some missing observations. E.g., to predict the values of diseases given some symptons, or, given a noisy acousting signal, we want to find the most likely utterance that may have produced it.

Main types of queries



The MAP is not necessarily $(a^0, t^0, l^0, e^0, b^1, d^1, x^0)$, i.e., the list of most likely values in each conditional marginal distribution: the marginals are not independent in our model.

Complexity

- Exact inference is NP-hard in the general case
  - Feasible for discrete BNs and GBNs with relatively sparse structures;
  - NP-hard in the general case even for polytrees;
- Approximate inference
  - Absolute error within $\epsilon < \frac{1}{2}$ of true probability in polynomial time when no evidence ($\mathbf{E} = \emptyset$); yet NP-hard when $\mathbf{E} \neq \emptyset$;
  - Absolute error within $\epsilon < \frac{1}{2}$ of true probability NP-hard with evidence ($\mathbf{E} \neq \emptyset$);
  - Relative error within ϵ of true probability NP-hard with evidence and without evidence.
- MAP:
  - NP-hard; marginal MAP NP-hard even for polytrees

An NP-hard problem: probably (unless P = NP) requires exponential time in the worst case, i.e., most likely cannot be solved in polynomial time.

Methods

- Exact inference
  - Variable elimination
  - Clique tree message passing
- Particle-based approximate inference
  - Forward sampling
  - Likelihood weighting
  - Markov chain Monte Carlo

Variable elimination

Variable elimination

Brute force (i.e., using the joint) requires a number of summations exponential in the number of unobserved variables. In our example, $2^7 = 128$ summations.

$$P(D) = \sum_{a,s,t,l,b,e,x} P(a)P(s)P(t \mid a)P(e \mid l,t)P(l \mid s)P(b \mid s)P(x \mid e)P(D \mid b,e).$$

However, the summations have many common factors which we can cache and reuse. Using the distributive property we can push summations in. For example, the summation over A:

$$P(D) = \sum_{s,l,t,b,e,x} \left(\sum_{a} P(a)P(t \mid a) \right) P(s)P(e \mid l,t)P(l \mid s)P(b \mid s)P(x \mid e)P(D \mid b,e).$$

Variable elimination

Pushing in the summation over A means multiplying all factors that involve A and then summing over all values of A. We have two factors involving A:

A	P
yes	0.010
no	0.990

Table 2: $P(A)$

T	A	P
yes	yes	0.050
no	yes	0.950
yes	no	0.010
no	no	0.990

Table 3: $P(T \mid A)$

We multiply them to obtain a new factor, $\psi_1(T, A)$, over T and A.

T	A	P
yes	yes	0.001
no	yes	0.009
yes	no	0.010
no	no	0.980

Table 4: $\psi_1(T, A)$

Variable elimination

We now sum over A in $\psi_1(T, A)$ to create a new factor, $\tau_1(T)$

T	A	P
yes	yes	0.001
no	yes	0.009
yes	no	0.010
no	no	0.980

Table 5: $\psi_1(T, A)$

T	P
yes	0.010
no	0.990

Table 6: $\tau_1(T)$

$$\tau_1(T) = \sum_a \psi_1(T, a) = \sum_a P(T \mid a)P(a)$$

We have marginalized out A by performing 2 summations in $\psi_1(T, A)$.

Variable elimination

We now use $\tau_1(T)$ in our original summation. We did 2 summations and have at most $2^6 = 64$ left (if we took a brute force approach), compared to the original $2^7 = 128$ summations.

$$P(D) = \sum_{s,l,t,b,e,x} \left(\sum_{a} P(a)P(t \mid a) \right) P(s)P(e \mid l,t)P(l \mid s)P(b \mid s)P(x \mid e)P(d \mid b,e)$$

$$= \sum_{s,l,t,b,e,x} \tau_1(t)P(s)P(e \mid l,t)P(l \mid s)P(b \mid s)P(x \mid e)P(d \mid b,e)$$

We now keep eliminating variables, until we are left with $P(D)$.

Variable elimination

Consider eliminating T next. We multiply the two factors involving A:

$\tau_1(T)$

T	P
yes	0.010
no	0.990

Table 7: $\tau_1(T)$

$P(E \mid L, T)$

E	L	T	P
yes	yes	yes	1.000
no	yes	yes	0.000
yes	no	yes	1.000
no	no	yes	0.000
yes	yes	no	1.000
no	yes	no	0.000
yes	no	no	0.000
no	no	no	1.000

Table 8: $P(E \mid L, T)$

$\psi_2(E, L, T) = \tau_1(T)P(E \mid L, T)$.

E	L	T	P
yes	yes	yes	0.010
no	yes	yes	0.000
yes	no	yes	0.010
no	no	yes	0.000
yes	yes	no	0.990
no	yes	no	0.000
yes	no	no	0.000
no	no	no	0.990

Table 9: $\psi_2(E, L, T)$

Variable elimination

We now marginalize out T.

$$\psi_2(E, L, T) = \tau_1(T)P(E \mid L, T).$$

E	L	T	P
yes	yes	yes	0.010
no	yes	yes	0.000
yes	no	yes	0.010
no	no	yes	0.000
yes	yes	no	0.990
no	yes	no	0.000
yes	no	no	0.000
no	no	no	0.990

Table 10: $\psi_2(E, L, T)$

$$\tau_2(E, L) = \sum_t \psi_2(E, L, t).$$

E	L	P
yes	yes	1.000
no	yes	0.000
yes	no	0.010
no	no	0.990

Table 11: $\tau_2(E, L)$

Variable elimination example run

Consider computing $P(D)$ with the elimination ordering T, S, E, A, L, B, X. Steps 1–3 are:

1 $\mathcal{L} = \{f_A(A), f_T(T,A), f_S(S), f_L(L,S), f_B(B,S), f_E(E,T,L), f_X(X,E), f_D(D,E,B)\}$. **Delete T.**

$$g_1(A,E,L) = \sum_T (f_T(A,T) \times f_E(E,T,L))$$

not necessarily a probability term

size = 16

2 $\mathcal{L} = \{f_A(A), f_S(S), f_L(L,S), f_B(B,S), f_X(X,E), f_D(D,E,B), g_1(A,E,L)\}$. **Delete S.**

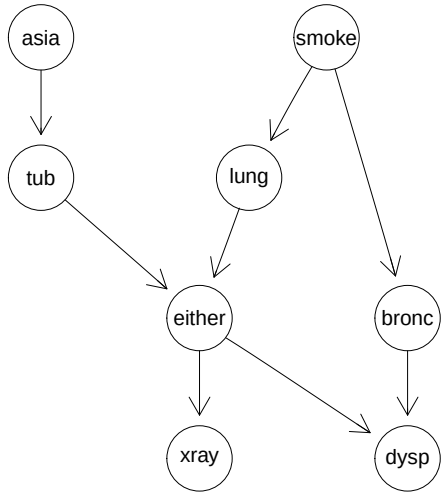
$$g_2(L,B) = \sum_S (f_S(S) \times f_L(L,S) \times f_B(B,S))$$

size = 8

3 $\mathcal{L} = \{f_A(A), f_X(X,E), f_D(D,E,B), g_1(A,E,L), g_2(L,B)\}$. **Del. E**

$$g_3(X,D,B,A,L) = \sum_E (f_X(X,E) \times f_D(D,E,B) \times g_1(A,E,L))$$

size = 64



Variable elimination example run

Steps 4–7:

4 $\mathcal{L} = \{\underbrace{f_A(A)}, g_2(L,B), \underbrace{g_3(X,D,B,A,L)}\}$. **Delete A** size = 32

$$g_4(X,D,B,L) = \sum_A (f_A(A) \times g_3(X,D,B,A,L))$$

5 $\mathcal{L} = \{\underbrace{g_2(L,B), g_4(X,D,B,L)}\}$. **Delete L.** size = 16

$$g_5(X,D,B) = \sum_L g_2(L,B) \times g_4(X,D,B,L)$$

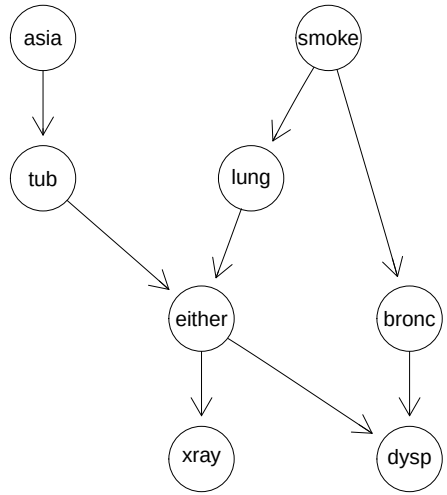
6 $\mathcal{L} = \{\underbrace{g_5(X,D,B)}\}$. **Delete B.** size = 8

$$g_6(X,D) = \sum_B g_5(X,D,B)$$

7 $\mathcal{L} = \{\underbrace{g_6(X,D)}\}$. **Delete X.** size = 4

$$g_7(D) = \sum_X g_6(X,D)$$

8 return normalize($g_7(D)$)



Variable elimination example run

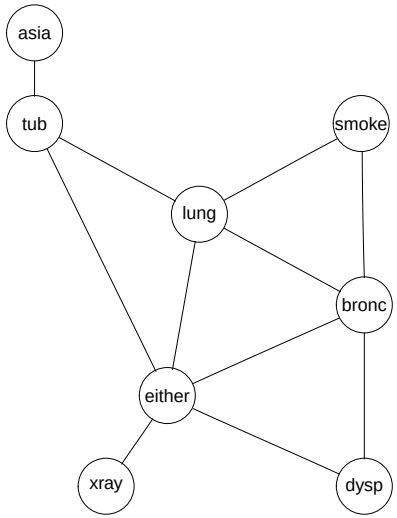
We have generated intermediate factors ψ of sizes: 16, 8, 64, 32, 16, 8, and 4 entries. The largest factor was $\psi(X, D, B, A, L, E)$ with $2^6 = 64$ entries.

We reduced these to the $\tau()$ factors with a total of $16/2 + 8/2 + 64/2 + 32/2 + 16/2 + 8/2 + 4/2 = 72$ summations.

Note that the factor over T was larger than in our example where we eliminated A first. The difference is due to the different elimination ordering.

Variable elimination ordering

The computational cost is dominated by the sizes of the intermediate factors generated. The intermediate factors correspond to maximal cliques in the undirected graph induced by elimination. A maximal clique is a complete subgraph that is not a subgraph of another clique.



Intermediate factor sizes depend on DAG density and an the elimination ordering. Finding the optimal elimination ordering is NP-hard, yet heuristics exist for finding good orderings.

Accounting for evidence

Accounting for evidence is simple. For an observation e of a variable E:

- For every factor ψ involving E:
  - remove entries inconsistent with e;
  - remove E from the scope of the factor;
- Remove **E** from the set of variables to be eliminated.

A lot of evidence variables can **simplify the query** as we have less non-evidence variables **Q** to sum over.

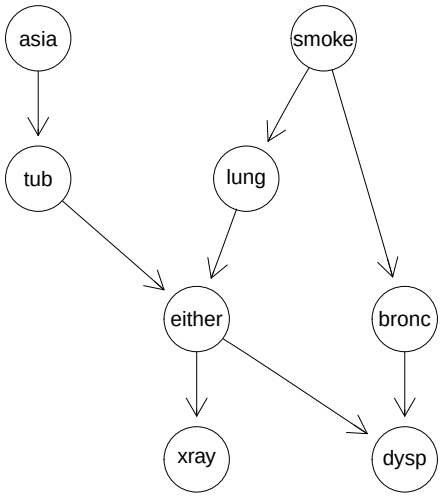
Clique tree message passing

By running VE we obtain the marginal probability for the variables of interest $\mathbf{Q}$. By caching the performed computations, we can obtain marginal probabilities for all maximal cliques in the graph induced by elimination, in twice the running time of VE. The algorithm is called clique tree message propagation.

Particle-based approximate inference

Particle-based approximate inference

Randomly sample M instances (particles) from the BN. Then estimate different probabilities of interest from the sample. For example, with $M = 3$ (toy example), we estimate $P(A = a^1)$ and $P(S = s^1)$:



$\rightarrow$

A	S	T	L	B	E	X	D
no	yes	no	no	yes	no	no	yes
no	yes	no	no	no	no	no	no
no	no	yes	no	no	yes	yes	yes

$$P(A = a^1) = \frac{0}{3} = 0 \qquad P(S = s^1) = \frac{2}{3}$$

Forward sampling

Sampling from a BN without any evidence is straightforward. The following is the algorithm:

1. Let $(X_1, \ldots, X_n)$ be a topological ordering of $\mathbf{X}$
2. for i $= 1, \ldots, n$:
   3. Sample x_i from $P(X_i \mid \mathbf{pa}(x_i))$
3. return $(x_1, \ldots, x_n)$

The cost is $O(Mnp \log d)$, where $p = \max_i |\mathrm{Pa}_{X_i}|$, and $d = \max_i |\Omega_{X_i}|$.

Convergence

We can estimate the probability of an absolute error for a given M:

$$P_{\mathcal{D}}\left(\hat{P}_{\mathcal{D}}(\boldsymbol{e}) \notin [P(\boldsymbol{e}) - \epsilon, P(\boldsymbol{e}) + \epsilon]\right) \leq 2e^{-2M\epsilon^2},$$

which gives the bound

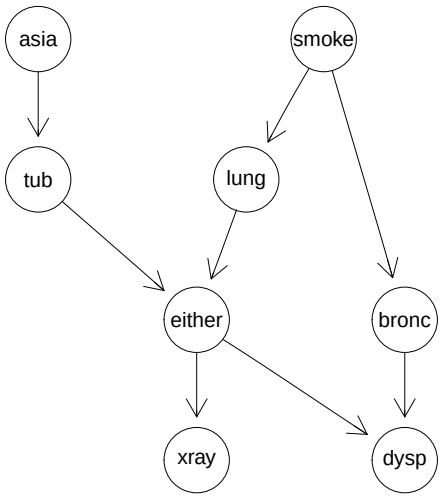
$$M \geq \frac{\ln(2/\delta)}{2\epsilon^2},$$

needed to bound the absolute error with probability $1 - \delta$, where ϵ is the error. Thus, M grows logarithmically with $\frac{1}{\delta}$ and quadratically in $\frac{1}{\epsilon}$.

Absolute error, however, may not be very useful if it is large compared to the probability we are estimating. While the relative error is of interest, we do not know $P(\mathbf{e})$ and thus can never establish an analogous bound for M.

Rejection sampling

For $P(\mathbf{Q} \mid \mathbf{e})$, with evidence $\mathbf{e}$, the straighforward adaptation is to reject particles inconsistent with $\mathbf{e}$. For example, with evidence $T = t^1$, we estimate $P(A = a^1 \mid T = t^1)$ and $P(S = s^1 \mid T = t^1)$ with $M = 3$:



$\rightarrow$

A	S	T	L	B	E	X	D
no	no	yes	no	no	yes	yes	yes

$$P(A = a^1 \mid T = t^1) = \frac{0}{1} = 0 \qquad P(S = s^1 \mid T = t^1) = \frac{0}{1} = 0$$

Inefficient when $P(\mathbf{e})$ is low, as it will reject many particles.

Likelihood weighting

Instead of sampling from evidence nodes $\mathbf{E}$, fix their values as $\mathbf{E} = \mathbf{e}$. Adjust the weight of each obtained instance so as to account for its likelihood given $\mathbf{e}$:

$$w = \prod_i P(E_i = e_i \mid \mathbf{pa}(e_i)).$$

No samples are rejected, though some may have very low weights.

Summary

Exact inference

- Straightforward to use
- Tractable for sparse DAGs, not for densely connected DAGs
- Clique tree message passing: answer multiple queries with twice the cost of VE

Approximate inference by sampling

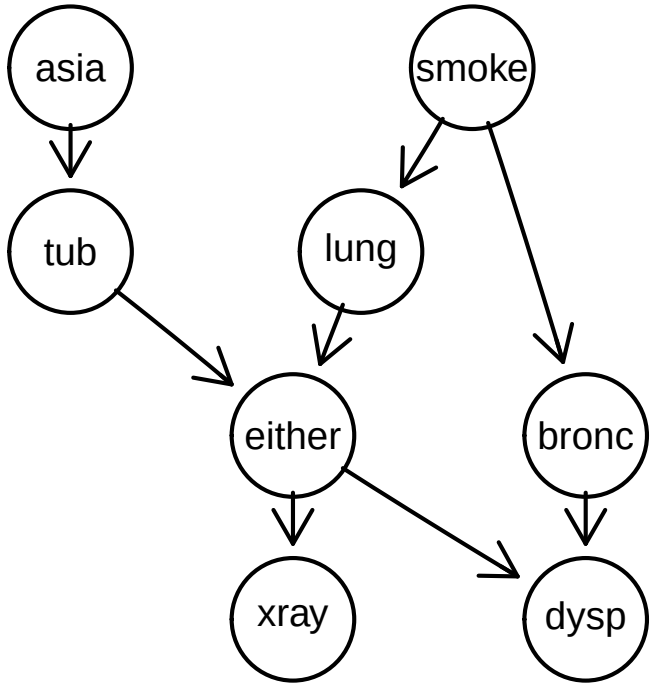
- Approximation improves with the number of samples
- Likelihood weighting faster than rejection sampling
- Easy to parallelize
- Applicable regardless of DAG density
- Applicable to CPDs of arbitrary form; not limited to Gaussian or discrete networks

Hands-on

Inference in R

- The `gRain` package clique implements clique tree message passing for discrete Bayesian networks.
- `bnlearn` provides particle-based approximate inference for discrete, Gaussian and conditional linear Gaussian networks.
- Neither `bnlearn` nor `gRain` implement MAP inference. We can only find the MAP over a small set of variables **A** for which we can compute or approximate the joint.

The chest clinic



All variables are Boolean (yes/no). Load the network.

```
asia.fit <- load_asia_fit()
```

Exact inference complexity

Consider queries of the type $P(\mathbf{Q}|\mathbf{e})$, where $\mathbf{W} = \mathbf{X} \setminus \mathbf{E} \setminus \mathbf{Q}$ may be non-empty.

If all variables are observed, i.e., $\mathbf{E} = \mathbf{X}$, how many summations do we need in order to compute the probability of evidence?

Consider a JPD over n binary variables. We have observed all variables except for A and B, and we want the marginal for B. How many summations do we need to find $P(D)$? What if we observed no evidence?

Clique tree message passing

`gRain` steps when performing inference:

- compile: moralize, triangulate, find RIP ordering, form initial clique potentials
- optional: absorb evidence
- propagate: transform clique potentials to clique marginals

Thus, compile chooses an elimination ordering and therefore determines complexity of inference. These steps are taken internally by `querygrain`.

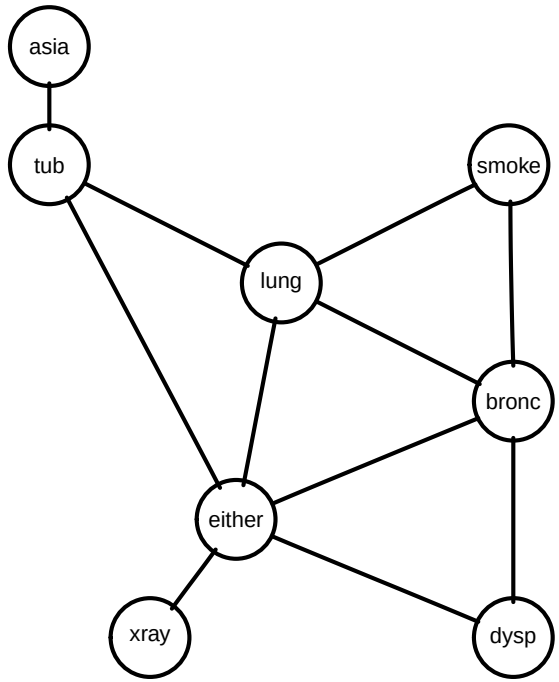
Clique tree message passing

The 'cliques' are the scopes (the domain variables) of the intermediate factors generated. Each corresponds to maximal clique in the graph on the right.

```
asia.gr <- as.grain(asia.fit)
asia.gr <- compile(asia.gr)
asia.gr$rip$cliques
```

```
## [[1]]
## [1] "asia" "tub"
##
## [[2]]
## [1] "either" "lung"   "tub"
##
## [[3]]
## [1] "either" "lung"   "bronc"
##
## [[4]]
## [1] "smoke" "lung"  "bronc"
##
## [[5]]
## [1] "either" "dysp"   "bronc"
##
## [[6]]
## [1] "either" "xray"
```

```
plot(asia.gr$ug)
```



Clique tree message passing

After running inference, each calibrated factor $\psi(\mathbf{X})$ corresponds to $P(\mathbf{X})$. For example, the the first factor corresponds to $P(A, T)$:

```
asia.gr <- propagate(asia.gr)
asia.gr$equipot[[1]]
```

```
##       asia
## tub       yes     no
##   yes 0.0005 0.0099
##   no  0.0095 0.9801
```

Original CPTs are perserved with a single factor.

Clique tree message passing

As long as the variables of interest are within the same ψ, we can efficiently query `asia.gr`, as all the relevant information is stored in ψ.

Compare using `asia.gr` versus a running message passing for each query.

```
asia.gr <- compile(asia.gr)
system.time(replicate(1e2, querygrain(asia.gr, 'dysp')))
not_compiled  <- as.grain(asia.fit)
system.time(replicate(1e2, querygrain(not_compiled, 'dysp')))
```

```
##    user  system elapsed
##   0.130   0.000   0.169
##    user  system elapsed
##   1.857   0.004   2.008
```

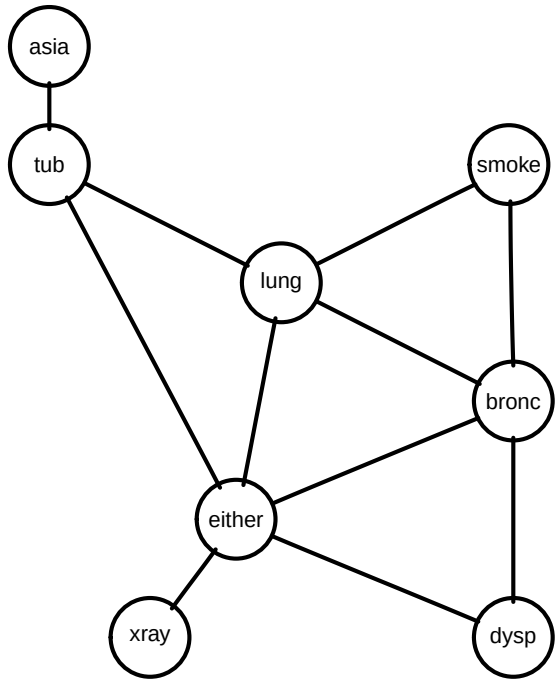
Clique tree message passing

Note that the largest factors created by clique tree message passing have a scope of three variables, whereas are example when computing $P(D)$ had a factor with six variables. Therefore, a much more efficient ordering exists than the one we used in our example.

```
asia.gr <- as.grain(asia.fit)
asia.gr <- compile(asia.gr)
asia.gr$rip$cliques
```

```
## [[1]]
## [1] "asia" "tub"
##
## [[2]]
## [1] "either" "lung"   "tub"
##
## [[3]]
## [1] "either" "lung"   "bronc"
##
## [[4]]
## [1] "smoke" "lung"  "bronc"
##
## [[5]]
## [1] "either" "dysp"   "bronc"
##
## [[6]]
## [1] "either" "xray"
```

```
plot(asia.gr$ug)
```



Forward sampling with base R

Sample smoke, lung, and bronc with the base function `sample()`. Begin with smoke:

```
yn <- c("yes", "no")
set.seed(0)
s <- sample(yn, 1, prob = asia.fit$smoke$prob)
s
```

```
## [1] "yes"
```

Sample L and B by chosing the appropriate conditional distribution.

```
set.seed(0)
l <- sample(yn, 1, prob = asia.fit$lung$prob[ ])
b <- sample(yn, 1, prob = asia.fit$bronc$prob[])
c(s, l, b)
```

What values do you get? Repeat. Should we remove `set.seed()`?

Forward sampling with base R

We now have a complete particle:

```
a <- sample(yn, 1, prob = asia.fit$asia$prob)
t <- sample(yn, 1, prob = asia.fit$tub$prob[, a])
e <- sample(yn, 1, prob = asia.fit$either$prob[, l, t])
x <- sample(yn, 1, prob = asia.fit$xray$prob[, e])
d <- sample(yn, 1, prob = asia.fit$dysp$prob[, b, e])
instance <- c(a, s, t, l, e, x, b, d)
instance
```

```
## [1] "no"  "yes" "no"  "no"  "no"  "no"  "yes" "no"
```

How could we get 15 particles? Hint: second argument of `sample()`.

Forward sampling with bnlearn

`bnlearn` provides two functions for particle-based inference:

- `cpdist` : create a sample $\mathcal{D}$ of n [weighted] instances. Then estimate any probability of interest from the sample.
- `cpquery`: compute the probability of an event in $\mathcal{D}$

Let us draw 15 samples. `evidence = TRUE` means that we are not conditioning on any evidence.

```
set.seed(0)
samples.asia <- cpdist(asia.fit, nodes = nodes(asia.fit),
                       evidence = TRUE, n = 15)
summary(samples.asia)
```

```
##   asia      tub       smoke     lung      bronc     either    xray      c
##  yes: 1   yes: 0   yes:8   yes: 0   yes:10   yes: 0   yes: 1   ye
##  no :14   no :15   no :7   no :15   no : 5   no :15   no :14   no
```

Forward sampling with bnlearn

```
summary(samples.asia)
```

```
##   asia       tub       smoke      lung       bronc      either     xray       
##  yes: 1   yes: 0   yes:8   yes: 0   yes:10   yes: 0   yes: 1   ye
##  no :14   no :15   no :7   no :15   no : 5   no :15   no :14   no
```

- What is $P_{\mathcal{D}}(D)$?
- What is $P_{\mathcal{D}}(S = s^0, D = d^1)$?

```
t <- table(samples.asia[, c('smoke', 'dysp')])
prop.table(t)
```

```
##       dysp
## smoke       yes       no
##   yes 0.3333333 0.2000000
##   no  0.2666667 0.2000000
```

Forward sampling with bnlearn

What is the most likely assignment (MAP) to A and S?

```
t <- table(samples.asia[, c('smoke', 'asia')])
t
```

```
##       asia
## smoke yes no
##   yes   1  7
##   no    0  7
```

If we wanted the MAP over all variables, what size of table would we need?

```
prod(dim(table(samples.asia)))
```

To obtain the MAP with particle-based sampling, we would need an accurate inference of the joint probability. This gets harder as the number of variables increases.

Forward sampling with bnlearn

We can also use `cpquery` to get the probability of an event directly (avoid the `prop.table(table())` step). The arguments are:

- event is Q; evidence is E; n is M;
- method is `ls` for logic sampling.

```
set.seed(0)
ep <- cpquery(asia.fit,
     event = (smoke == "no" & dysp == "yes"),
     evidence = TRUE, n = 15)
ep
```

```
## [1] 0.1333333
```

Forward sampling with bnlearn

```
set.seed(0)
ep <- cpquery(asia.fit,
     event = (smoke == "no" & dysp == "yes"),
     evidence = TRUE, n = 15)
ep
```

```
## [1] 0.1333333
```

`event` end `evidence` are expressions that are evaluated on the generated particles, and need to evaluate to a logical vector of length `n`. We can thus use Boolean operators. Above, we used `&` to specify the event of a patient not being a smoker and suffering from dyspnea.

Since we can perform exact inference on this network, let us measure the absolute error ($|P(S = s^0, D = d^1) - P_{\mathcal{D}}(S = s^0, D = d^1)|$).

Forward sampling with bnlearn

We need `grain` for exact inference.

```
gr.asia <- as.grain(asia.fit)
q <- querygrain(gr.asia,
       nodes = c("smoke", "dysp"), type = "joint")
q
```

What is the absolute error? Use `q` to get the true probability.

Now reduce absolute error below 0.001.

```
set.seed(0)
ep <- cpquery(asia.fit,  )
```

Forward sampling with bnlearn

To bound the absolute error with probability $1 - \delta$, we need $M \geq \frac{\ln(2/\delta)}{2\epsilon^2}$, where ϵ is the error. For $\epsilon = \frac{1}{10000}$ and $\epsilon = \frac{1}{100000}$ with 99% probability, we need M in the order of 2e8 and 2e10, respectively:

```
compute_nparticles_abserr <- function(delta, epsilon) {
  log(2 / delta) /  (2 * epsilon ^ 2)
}
mbound <- compute_nparticles_abserr(1e-2, 1e-4)
mbound2 <- compute_nparticles_abserr(1e-2, 1e-5)
formatC(c(mbound, mbound2))
mbound2 > .Machine$integer.max
```

```
## [1] "2.649e+08" "2.649e+10"
## [1] TRUE
```

Note that the lower bound on M is **independent of network complexity**, and only depends on ϵ and δ.

Forward sampling with bnlearn

Consider the not very likely event that `(asia == "yes") & (tub == "yes")`, with probability 0.0005.

```
gr.asia <- as.grain(asia.fit)
q <- querygrain(gr.asia,
       nodes = c("asia", "tub"), type = "joint")
tp <- q['yes', 'yes']
tp
```

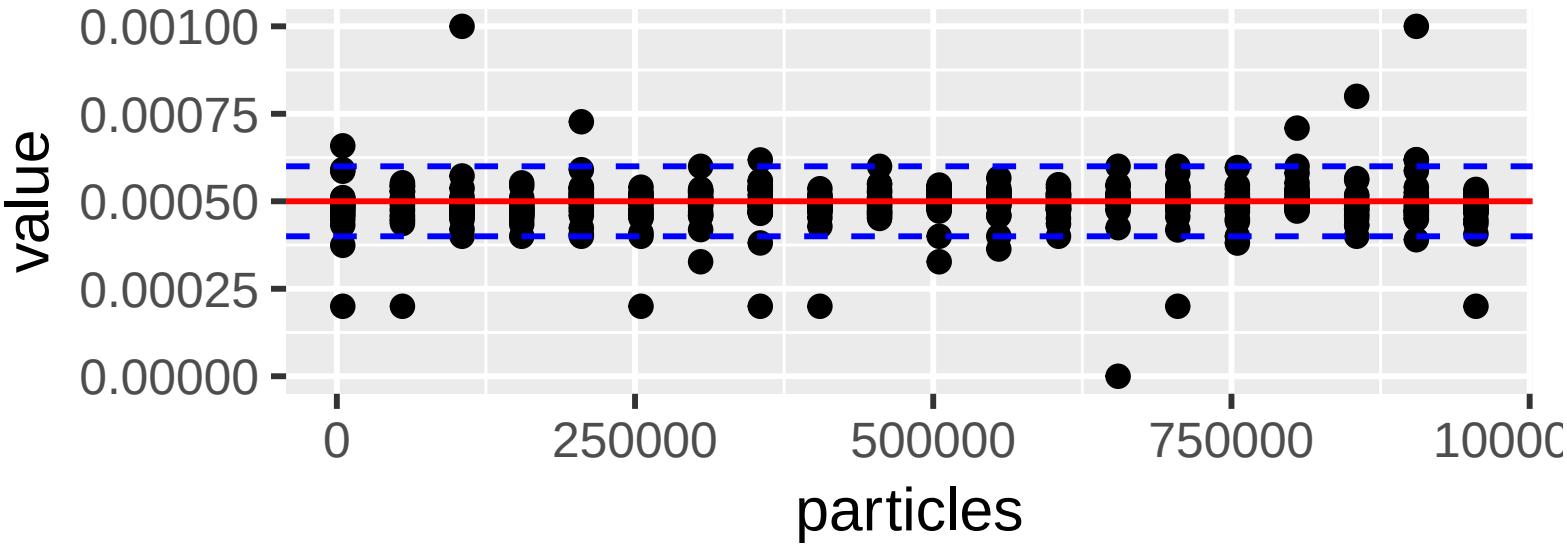
```
## [1] 5e-04
```

Plot our estimates of this probability with M increasing from 5000 to 10^{6}.

```
nparticles = seq(from = 5 * 10^3, to = 10^6, by = 5 * 10^4)
prob = matrix(0, nrow = length(nparticles), ncol = 20)
for (i in seq_along(nparticles))
   for (j in 1:20)
      prob[i, j] = cpquery(asia.fit, event
         = (asia == "yes") & (tub == "yes"),
         evidence = TRUE, method = "ls", n = nparticles[i])
```

Forward sampling with bnlearn

Even though we need $M \geq 2e8$ to get within $\epsilon = \frac{1}{10000}$ with high probability, we are already within ϵ at much lower Ms. The horizontal lines show the true probability $\pm\epsilon$. Variance of the estimates barely seems to be decreasing however at these lower Ms.



Forward sampling with bnlearn

On my machine, bnlearn is relatively fast to sample `2.6e8` particles. By using `cpquery` instead of `cpdist` we avoid loading a large data structure into memory. The absolute error is far below $\epsilon = \frac{1}{10000}$.

```
large <- cpquery(asia.fit,
        event = (asia == "yes") & (tub == "yes"),
        evidence = TRUE, method = "ls", n = 2.6e8 )
(large - tp)
min(large, tp) / max(large, tp)
```

```
## [1] -2.315385e-06
## [1] 0.9953692
```

Note that the estimate is close to the true probability also in relative terms. This is true as long as the absolute error is small relative to the (unkown) true probability. In high-dimensional spaces many events have very low probabilities, yes some can be magnitudes more likely that others.

Forward sampling with bnlearn

2e10 exceeds the largest integer representable on my machine and is out of reach for bnlearn. Indeed, 2e10 particles cannot be sampled on my machine. Thus, absolute error within $\epsilon = \frac{1}{100000}$ with 99% probability is not feasible.

```
larger <- cpquery(asia.fit,
          event = (asia == "yes") & (tub == "yes"),
          evidence = TRUE, method = "ls", n = 2.6e10 )
larger
```

Rejection sampling

Particle-based inference gets harder when we have evidence.

Consider we know that: $A = a^1$. If we draw 5000 samples to estimate $P(S = s^0, D = d^1 | A = a^1)$, how many non-rejected samples do you expect? Why?

```
set.seed(0)
samples.asia <- cpdist(asia.fit,
                nodes = c("smoke", "dysp"),
                evidence=(asia == "yes"),
                n=5000)
```

Rejection sampling

What is the absolute error?

```
ep <- prop.table(table(samples.asia))
ep <- ep['no', 'yes']
gray <- setEvidence(gr.asia, nodes = "asia", states = c("yes"))
q <- querygrain(gray, nodes = c("smoke", "dysp"), type = "joint")
tp <- q['no', 'yes']
abs(ep - tp)
```

Reduce the absolute error below 0.001.

How could we estimate the probability of the evidence from the obtained sample? Complete the code below:

```
nrow(samples.asia) /
```

Likelihood weighting

Say that we know $A = a^1$. With likelihood weighting we fix $A = a^1$ and sample from the rest of the network. We need to set `method = "lw"`.

```
set.seed(0)
samples.asia <- cpdist(asia.fit, nodes = nodes(asia.fit),
                       evidence=list(asia = "yes"),
                       n=5000, method = "lw")
```

How many samples were generated? With rejection sampling from the prior distribution, without fixing A, how many samples did we obtain with the same evidence?

Likelihood weighting

What are the weights of the samples? Note that we are sampling from the posterior distribution, because asia is a root of the network.

```
w <- attr(samples.asia, 'weights')
summary(w)
```

We need weights when some $E \in \mathbf{E}$ has unobserved parents. So, consider instead that the sole evidence is $L = l^1$. Draw 5000 samples in S and D.

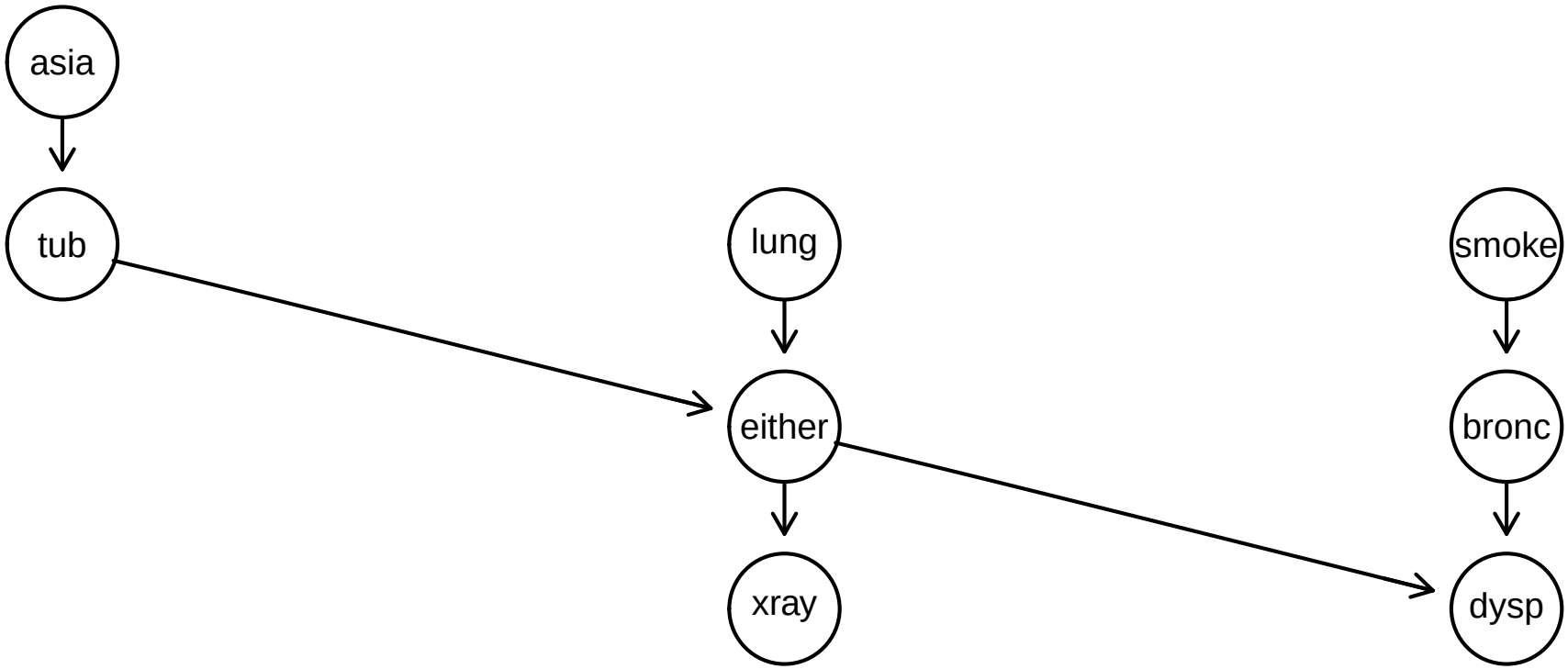
```
set.seed(0)
samples.asia <- cpdist(, nodes = ,
                       evidence=list(lung = 'yes'),
                       n=5000, method = "lw")
w <- attr(samples.asia, 'weights')
```

What are weights of the first five samples? Do they depend on D?

Likelihood weighting

The actual network that we are sampling from is called the mutilated network. For `lung = 'yes'`, lung has no parents in the mutilated network (it is independent of its parents as it takes a fixed value) whereas its distribution is not probabilistic.

```
mutbn = mutilated(asia.fit, list(lung = "yes"))
plot(bn.net(mutbn))
```



Likelihood weighting

Thus, we need weights to compensate for the difference between the mutilated and posterior distribution. Indeed, in our sample, the distribution of S in $\mathcal{D}$ corresponds to that in the prior distribution, due to the missing arc from S to L.

```
prop.table(table(samples.asia$smoke))
asia.fit$smoke
```

Likelihood weighting

Because asia is a root node, evidence on asia was taken into account when sampling from subsequent nodes. Smoker, however, was sampled from the prior distribution with 50% for each outcome. In the posterior distribution, conditioned to `lung = 'yes'`, the probability of not smoking decreases tenfold.

```
asia.fit$lung$prob['yes', ]
```

```
##  yes   no
## 0.10 0.01
```

Thus, cases with `smoke = 'yes'` are 10 times less likely when given `lung = 'yes'`, that those with `smoke = 'no'`. The weights, whose ratio is $\frac{P(L{=}l^1|S{=}s^1)}{P(L{=}l^1|S{=}s^0)} = 10$, correct for this. bnlearn returns normalized weights, so that the largest weight equals 1.

Likelihood weighting

Now that we have our sample and the weights we can ask queries about events or distributions. For example, $P_D(S, D \mid L = l^1)$ is:

```
prop.table(xtabs(w ~ smoke + dysp,
                 data = cbind(samples.asia, w = w)))
```

```
##       dysp
## smoke          yes           no
##   yes 0.74493777 0.15976221
##   no  0.07271038 0.02258963
```

True $P(S, D \mid L = l^1)$ is:

```
##       dysp
## smoke          yes           no
##   yes 0.74545455 0.16363636
##   no  0.06909091 0.02181818
## attr(,"class")
## [1] "parray" "array"
```

Likelihood weighting

If we use `cpquery` instead of `cpdist` it will take care of the weights for us, but we need to restrict our query to a specific event. Complete the code below to ask for the probability of $P_D(S = s^0, D = d^1 \mid L = l^1)$? (that is, `smoke == "no" & dysp == "yes"` given `lung = 'yes'`):

```
set.seed(0)
epl <- cpquery(,  event = (smoke == "no" & dysp == ),
               evidence = list(lung = 'yes'), n = 5000, method = 'lw
epl
```

What is the absolute error? (see two slides back)

```
tpl <- q['no', 'yes']
abs(tpl - )
```

The marks data set

Consider the marks dataset. It contains students' scores, from 0 to 100, in mechanics, vectors, algebra, analysis and statistics:

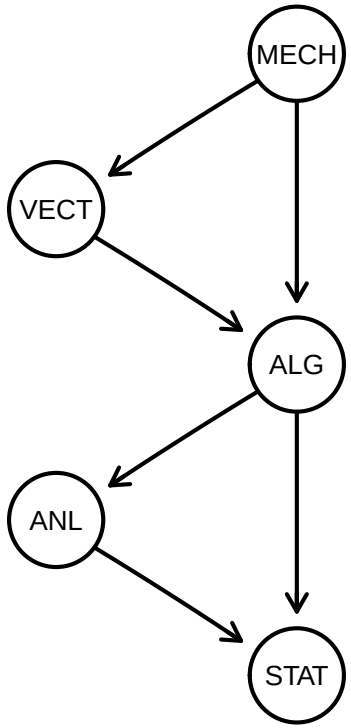
```
head(marks)
```

```
##   MECH VECT ALG ANL STAT
## 1   77   82  67  67   81
## 2   63   78  80  70   81
## 3   75   73  71  66   81
## 4   55   72  63  70   68
## 5   63   63  65  70   63
## 6   53   61  72  64   73
```

Gaussian networks

Only particle-based inference (`bnlearn`) is available for Gaussian networks. Let us first learn a network from the data `hc`.

```
bn.marks <- hc(marks)
plot(bn.marks)
```



Note that the Gaussian is not necessarily a good model for these data.

Gaussian networks

- What is the prob of ALG > 60?
- What is the probability of both STAT and MECH > 60?
- What is the probability of both STAT and MECH > 60 given ALG > 60?

```
bn.marks.fit <- bn.fit(bn.marks, marks)
cpquery(bn.marks.fit, event = ((ALG > 60)), evidence = TRUE)
cpquery(bn.marks.fit, event = ((STAT > 60) & (MECH > 60)),
        evidence = TRUE)
cpquery(bn.marks.fit, event = ((STAT > 60) & (MECH > 60)),
        evidence = (ALG > 60))
```

```
## [1] 0.1955
## [1] 0.035375
## [1] 0.1472149
```

Gaussian networks

Is STAT independent of MECH given ALG? Using `cpdist` we can their compute correlation.

```
samples.marks <- cpdist(bn.marks.fit, nodes = c('STAT', 'MECH'),
                        evidence = TRUE)
cor(samples.marks$STAT, samples.marks$MECH)
```

```
## [1] 0.3560979
```

```
samples.marks <- cpdist(bn.marks.fit, nodes = c('STAT', 'MECH'),
                        evidence = (ALG > 60))
cor(samples.marks$STAT, samples.marks$MECH)
```

```
## [1] 0.120473
```

Gaussian networks

Indeed, when we know ALG ≥ 60, MECH ≥ 60 does not tell us anything about the probability of STAT ≥ 60.

```
cpquery(bn.marks.fit, event = ((STAT > 60)),
        evidence = ((MECH > 60) & ALG > 60))
```

```
## [1] 0.4579646
```

```
cpquery(bn.marks.fit, event = ((STAT > 60)),
        evidence = (ALG > 60))
```

```
## [1] 0.4561518
```

We are not specifying the number particles. Look at the documentation of cpquery to find out how bnlearn determines the default number. The number of parameters of a network is given by `nparams()`.

Gaussian networks

What was the probability of the evidence? That is, what fraction of samples was kept with logic sampling?

```
samples.marks <- cpdist(bn.marks.fit, nodes = c('STAT', 'MECH'), ev
samples.marks.noevidence <- cpdist(bn.marks.fit, nodes = c('STAT',
nrow(samples.marks )
```

```
## [1] 1481
```

```
nrow(samples.marks.noevidence)
```

```
## [1] 8000
```

What is the marginal distribution of STAT given ALG ≥ 60? Which plot could we use?

Gaussian networks

With likelihood weighting, we need either specify exact values for evidence or an interval:

```
cpquery(bn.marks.fit, event = ((STAT > 60)),
        evidence = list(ALG = c(60, 100)), method = "lw")
```

```
## [1] 0.4758392
```

```
cpquery(bn.marks.fit, event = ((STAT > 60)),
        evidence = list(ALG = c(60)), method = "lw")
```

```
## [1] 0.2875819
```

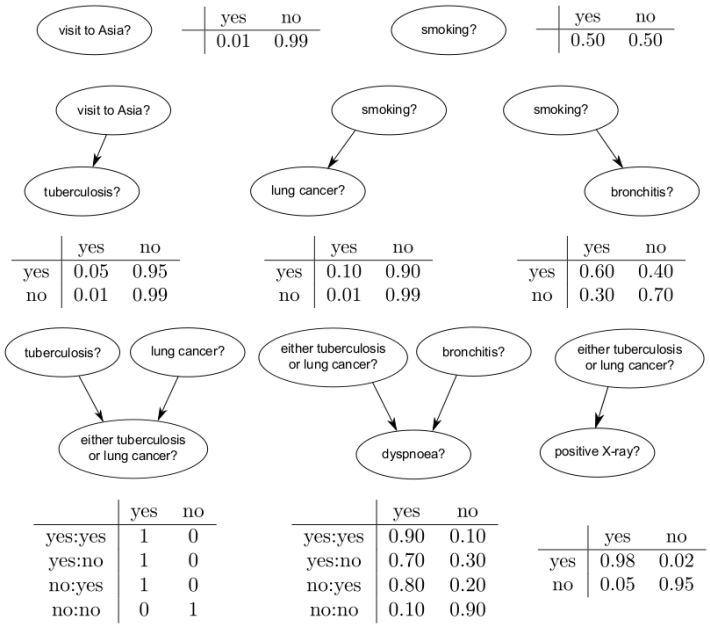
Compute the marginal of VECT given ALG > 60 an MECH > 60 with likelihood weighting.

Table of Contents

Overview

- Learn structure $\mathcal{G}$ and parameters (CPDs) $\boldsymbol{\theta}$
- Data set $\mathcal{D}$ with N samples

A	S	T	L	B	E	X	D
no	yes	no	no	yes	no	no	yes
no	yes	no	no	no	no	no	no
no	no	yes	no	no	yes	yes	yes
no	no	no	no	yes	no	no	yes
no	no	no	no	no	no	no	yes
no	yes	no	no	no	no	no	yes



Overview

Learn $\mathcal{G}$

- NP-hard in the general case
- Too many possible DAGs
- Good solutions with heuristic algorithms

Learn $\boldsymbol{\theta}$ given $\mathcal{G}$

- Straightforward with complete data
- Requires inference with incomplete data

Other

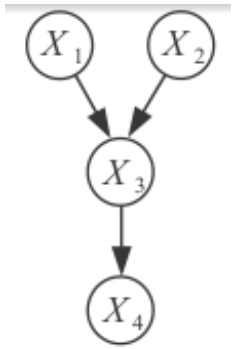
- We can incorporate prior knowledge: e.g., presence/absence of arcs, type of structure, prior probability of an arc
- Data size N: optimal results only in the large-sample limit

Learning parameters

Learning parameters

Empirical frequencies

- N_{ijk}: # cases in $\mathcal{D}$ with $X_i = k$ and $\mathbf{Pa}(X_i) = \mathbf{pa}_i^j$
- N_{ij}: # cases in $\mathcal{D}$ with $\mathbf{Pa}(X_i) = \mathbf{pa}_i^j$ ($N_{ij} = \sum_{k=1}^{R_i} N_{ijk}$)



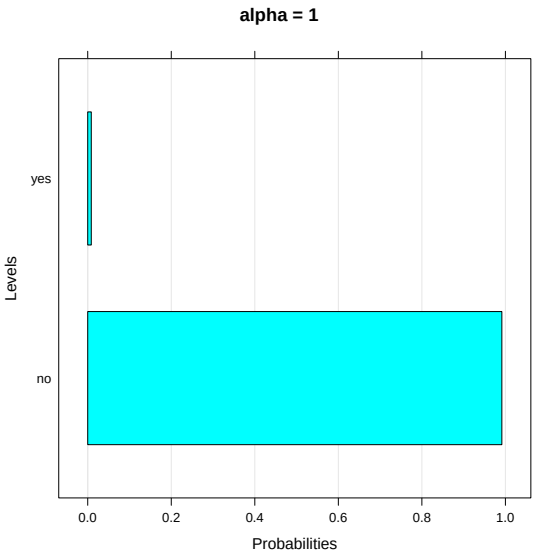
X_1	X_2	X_3	X_4
1	2	1	2
1	1	1	2
2	1	2	1
2	1	2	2
1	2	1	2
1	3	1	1

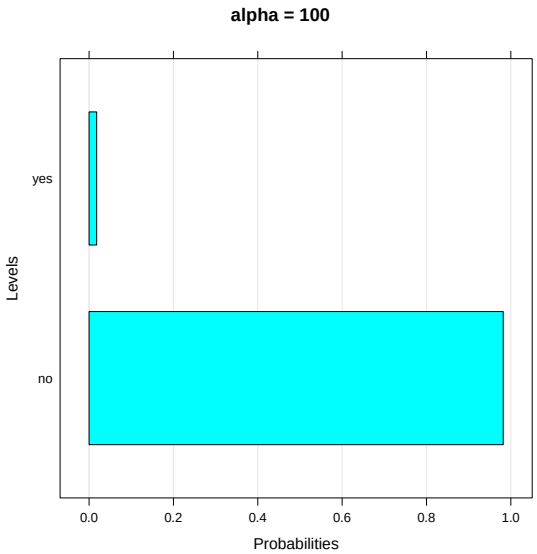
Maximum likelihood

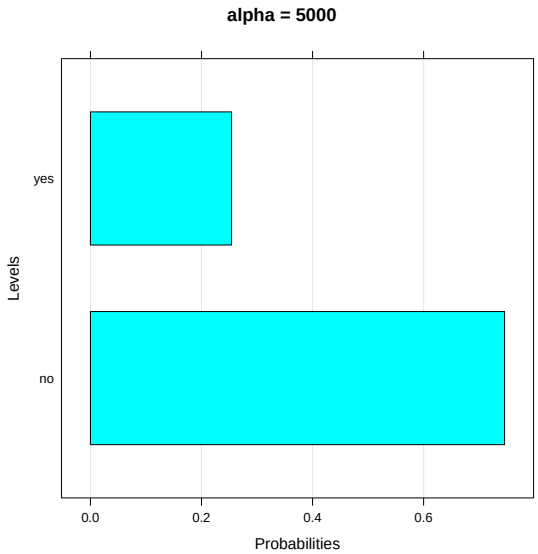
- $\widehat{\theta}_{ijk}^{\mathrm{ML}} = \frac{N_{ijk}}{N_{ij}}$
- $\theta_{1-1} = p(X_1 = 1) = \frac{N_{ijk}}{N_{ij}} = \frac{4}{6} = 2/3$
- $\theta_{322} = p(X_3 = 2 | X_1 = 1, X_2 = 2) = \frac{N_{ijk}}{N_{ij}} = \frac{0}{2} = 0$
- $\theta_{361} = p(X_3 = 1 | X_1 = 2, X_2 = 3) = \frac{N_{ijk}}{N_{ij}} = \frac{0}{0} =$ undefined

Learning parameters: Bayesian estimation

- Smoother estimates: no 0 probabilities
- A prior distribution over the parameters
- A Dirichlet prior gives the posterior in closed form
  - Parameter α imaginary sample size
  - Weight of the prior compared to the observed sample
- Learning the parameters for A from 5000 samples







Learning structure

Learning structure

Score-based

- Generate many candidate DAGs, select one with goodness-of-fit score
- Optimization problem

Hybrid

1. Statistical tests to reduce search space
2. Score-based search

Constraint-based

Basic idea

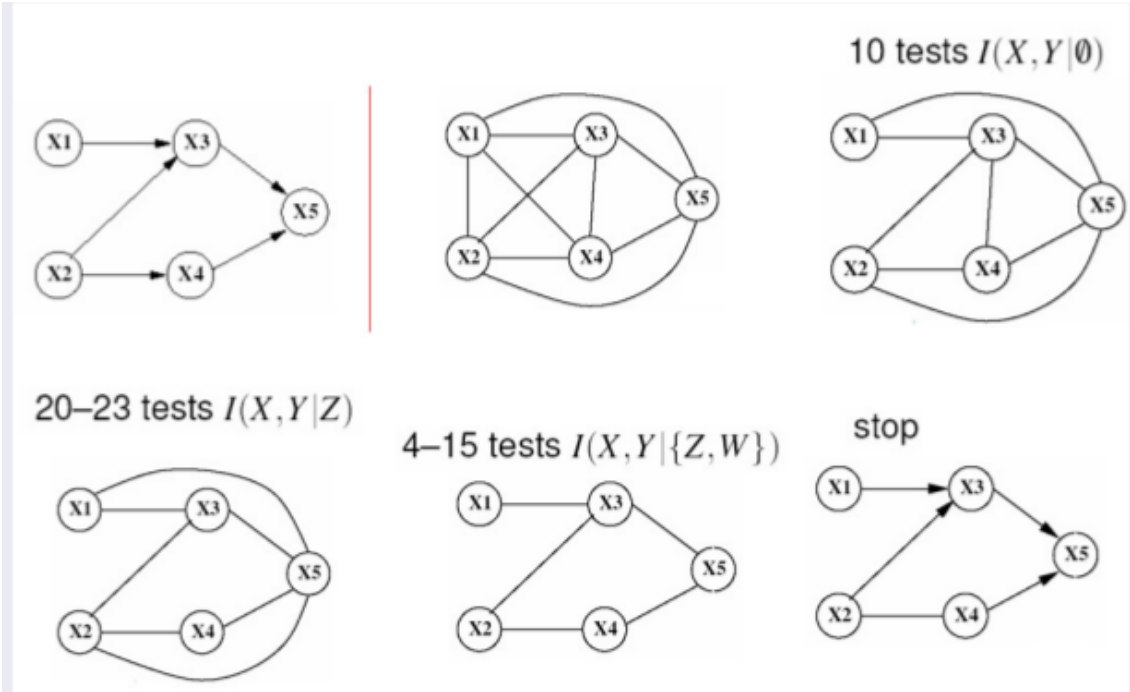
- Identify undirected edges:
  - $X - Y \in \mathcal{G}$ if $\nexists\, \mathbf{Z}$ s.t. $p_{X \perp\!\!\!\perp Y|\mathbf{Z}} < \alpha$
- Orient edges with simple rules
- Recover equivalence class
- Assumes the BN is a P-map, which does not hold for some distributions

Complex search

- 2^{n-2} possible $\mathbf{Z}$
- Test unreliable with large $\mathbf{Z}$
  - $2^{|\mathbf{Z}|}$ conditioning configurations with binary variables
- Heuristics: bound number of parents, backward search, learning Markov blankets and joining

Backward constraint-based overview

- Start from full undirected graph
- Remove $X - Y$ if $\exists\ \mathbf{Z}$ such that $p_{X \perp\!\!\!\perp Y|\mathbf{Z}} > \alpha$
  - (not rejecting independence hypothesis)
- Consider bigger $\mathbf{Z}$ at each iteration
- Stop at some predefined $|\mathbf{Z}|$
  - Hopefully find a good DAG with small $|\mathbf{Z}|$



Constraint-based algorithms in bnlearn

- Grow-Shrink and Incremental Association variants: learn the Markov blanket of each node with different forward and step-wise approaches; the initial network is assumed without edges;
- Max-Min Parents & Children: minimax approach to avoid CI tests known a priori to accept the null hypothesis of independence;
- Hiton-PC: the most scalable choice, uses a first pass based on marginal tests followed by a backward selection.

Independence tests

A common statistic is conditional mutual information

$$CMI(X, Y \mid \mathbf{Z}) = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} \frac{N_{ijk}}{N} \log \frac{N_{ijk} N_k}{N_{ik} N_{jk}}$$

Asymptotic tests

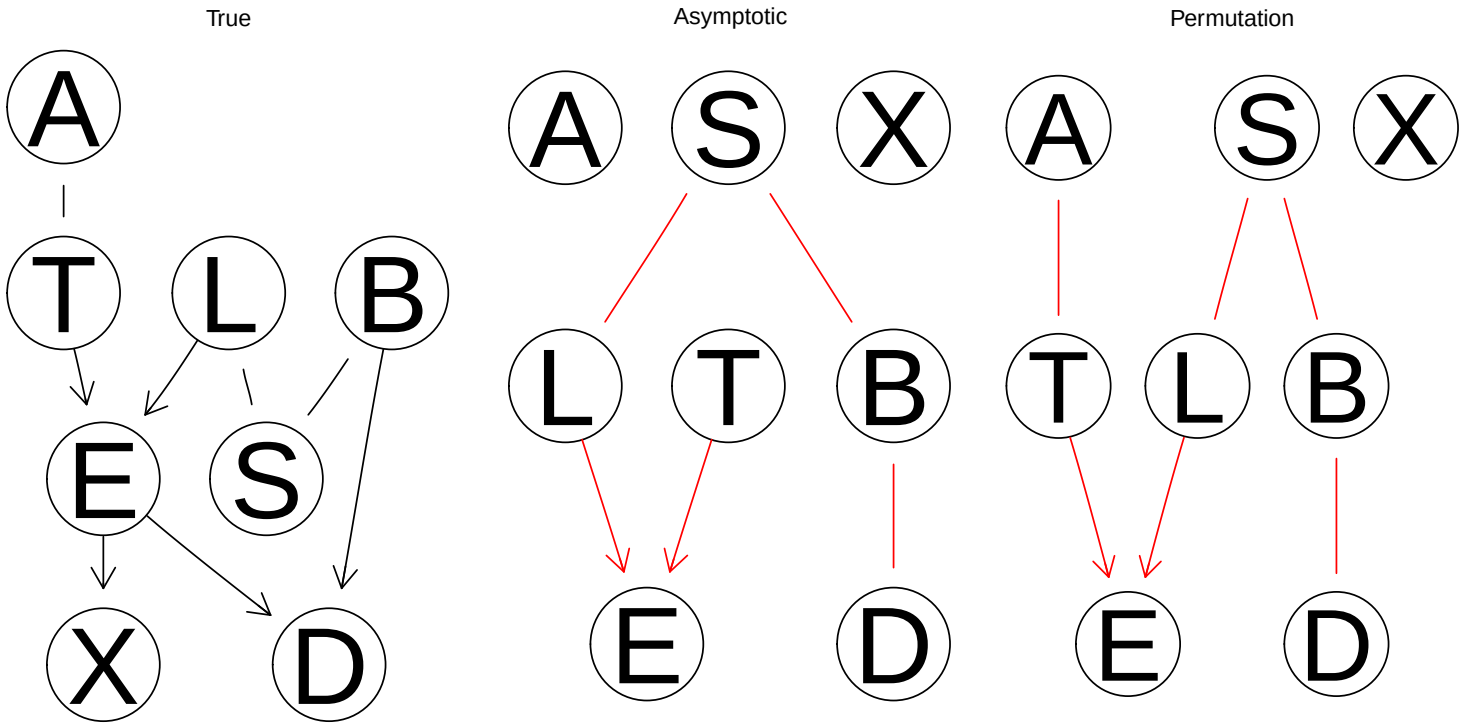
- Closed form for discrete and Gaussian variables
  - CMI has an asymptotic $\chi^2(I-1)(J-1)(K)$ null distribution
- Require a sufficient sample size for asymptotic behavior

Permutation tests

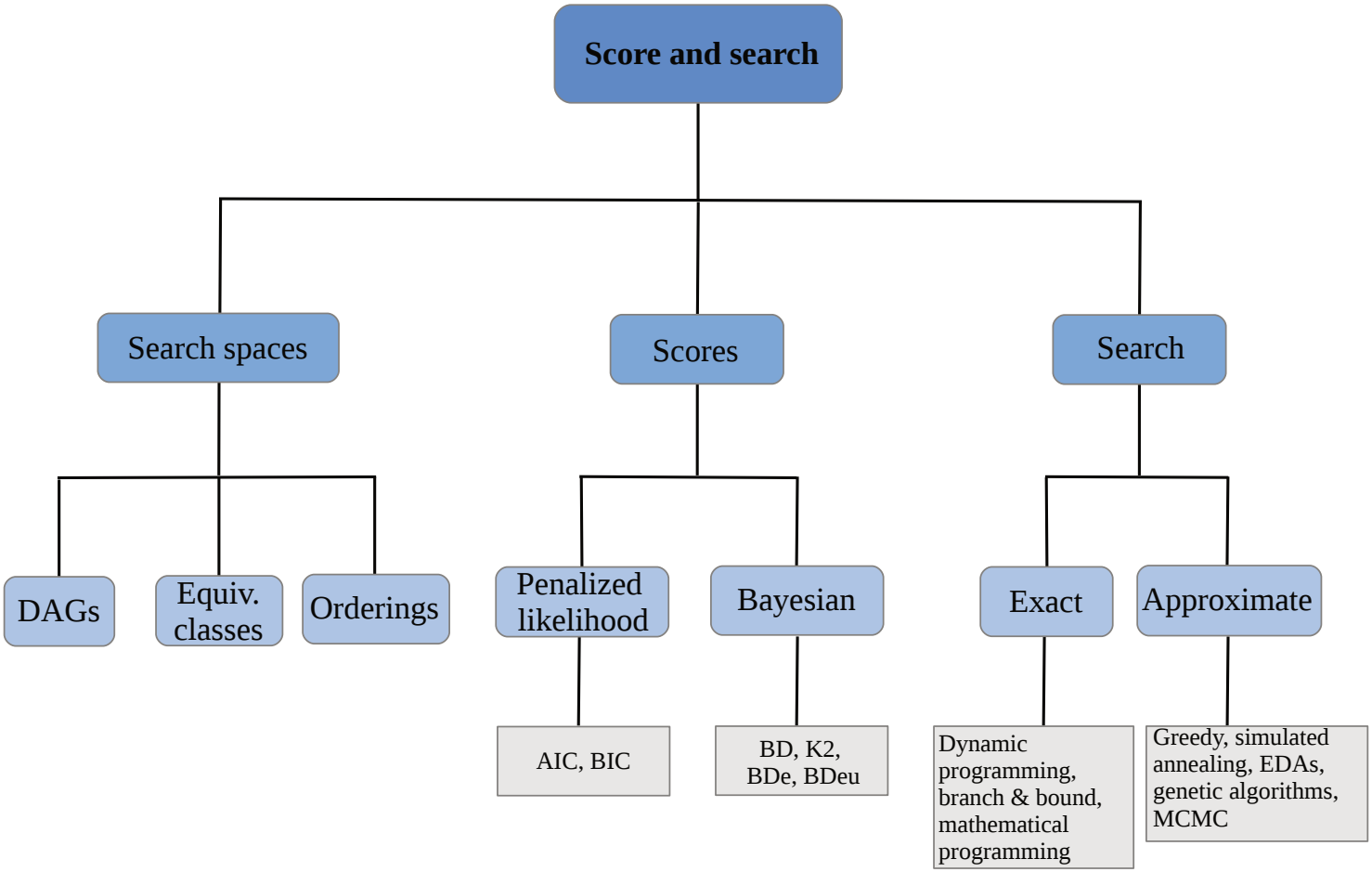
- Computationally intensive
- More reliable with small samples

Independence tests

- Same algorithm, two tests
- Asymptotic testing misses the $A - T$ edge
- Permutation testing only miss one arc

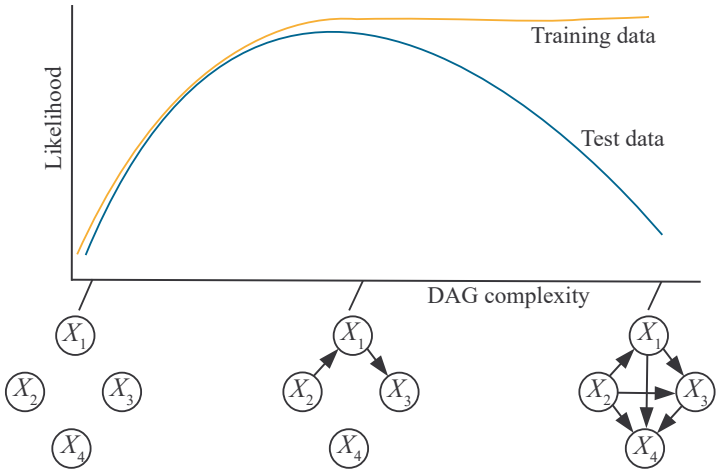


Score-based


Score and search
Search spaces
Scores
Search
DAGs
Equiv. classes
Orderings
Penalized likelihood
Bayesian
Exact
Approximate
AIC, BIC
BD, K2, BDe, BDeu
Dynamic programming, branch & bound, mathematical programming
Greedy, simulated annealing, EDAs, genetic algorithms, MCMC

Penalized likelihood scores

- We want to fit, not overfit, the data



- Commonly used approximations to $P(\mathcal{D}|\mathcal{G})$

$$AIC(\mathcal{G}) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{R_i} N_{ijk} \log \frac{N_{ijk}}{N_{ij}} - |\theta|$$

Tends to overfit

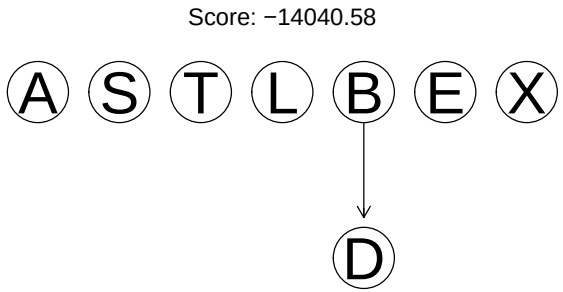
$$BIC(\mathcal{G}) \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{R_i} N_{ijk} \log \frac{N_{ijk}}{N_{ij}} - |\theta| \frac{1}{2} \log N$$

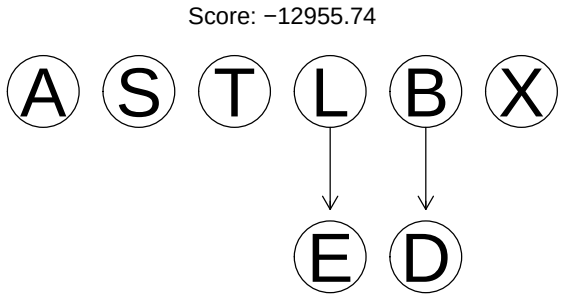
Tengs to underfit

- Also for Gaussian and conditional linear Gaussian

Search algorithms: local search

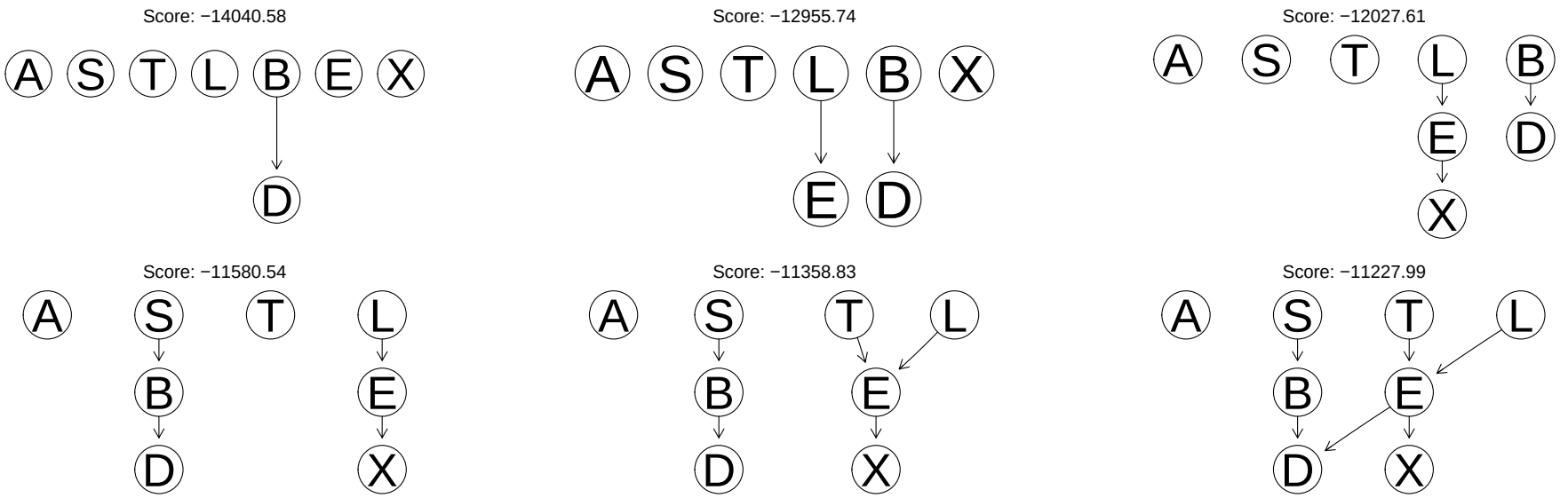
- Initial structure $\mathcal{G}$
- Local operators:
  - Edge addition
  - Edge deletion
  - Edge reversal
- Apply operators until stopping criterion





Greedy hill-climbing search on chest clinic

- Start with empty structure
- At each step apply best operator
- Stop when no improvement
- Here HC added six arcs in six steps



Search algorithms

Local optima with hill-climbing

- Random restarts from different structures
- Tabu search: intelligently consider non-improving solutions

Exact search

- Only for small n
- Tree-like DAG in time quadratic in n with a decomposable score
  - Tree-like: at most one parent per node

Non-local search heuristics

- Genetic search
- Simulated annealing
- Hard to tune parameters

Summary

- No single "best" algorithm or approach

Constraint-based

- Depend on reliability of test
- Asymptotic tests may be unreliable; permutation tests slow
- Easy to parallelize

Score-based

- Possible local optimum with limited data
- More commonly used
- Generally slower

Hybrid

- Considered state-of-the-art

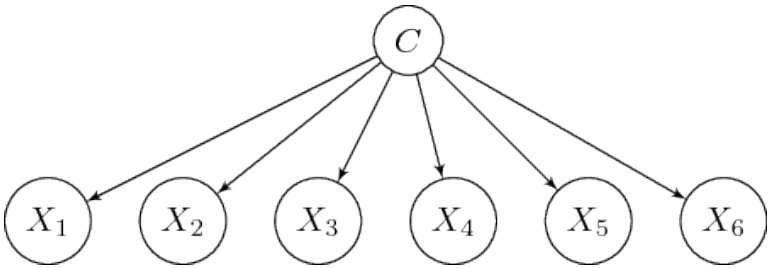
Parameters

- Bayesian estimates preferable over maximum likelihood

Classification

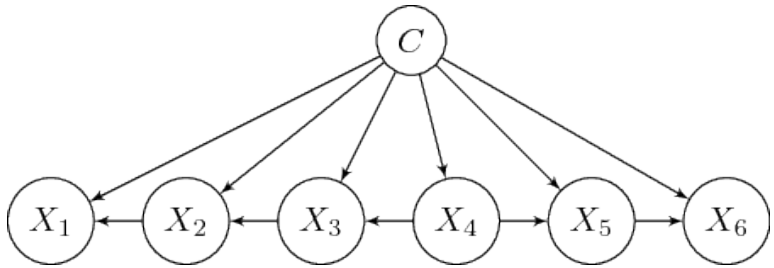
Bayesian network classifiers

- A Bayesian network used to obtain $P(c \mid \mathbf{x})$
- Interested in learning $P(c \mid \mathbf{x})$ rather than $P(c, \mathbf{x})$
- Differences with respect to learning $P(c, \mathbf{x})$



$$p(c, \mathbf{x}) = p(c)p(x_1|c)p(x_2|c)p(x_3|c)$$
$$p(x_4|c)p(x_5|c)p(x_6|c)$$

Naive Bayes



$$p(c, \mathbf{x}) = p(c)p(x_1|c, x_2)p(x_2|c, x_3)$$
$$p(x_3|c, x_4)p(x_4|c)p(x_5|c, x_4)p(x_6|c, x_5)$$

Tree-augmented naive Bayes

- Augmented naive Bayes DAG subspace
- Feature selection, ensembles of DAGs
- Discriminative scores, based on $P(c \mid \mathbf{x})$

Hands-on

Breast cancer data

Data on recurrence of breast cancer within five years of surgery.

- 286 patients; 85 recurring whereas 201 not (we have removed 9 patients with incomplete information). 9 measured variables:
  - Patient age (`age`)
  - Menopause (non, pre, post) (`menopause`)
  - Tumour malignancy degree (`deg_malig`)
  - Tumour location quadrant (`breast-quad`)
  - Perforated lymph node capsule (`node\_caps`)
  - Involved lymph nodes (`inv-nodes`)
  - Max. tumour diameter (`size`)
  - Patient irradiated (`irradiated`)
  - Left or right breast (`breast`)
  - Recurring or not (`Class`)

We have discretized `size` into groups 0–19, 20–39, and 40–54; and `inv-nodes` into groups 0–8, 9–17, and 8–26. This is basically a classification problem where we are interested in recurrence.

Breast cancer data

Load the data and have a look at it:

```
library(bnlearn)
breast <- foreign::read.arff('data/dbreast-cancer.arff')
head(breast[, 1:5])
```

```
##     age menopause tumor_size inv_nodes node_caps
## 1 40-49   premeno      0-19      0-8       yes
## 2 50-59      ge40      0-19      0-8        no
## 3 50-59      ge40     20-39      0-8        no
## 4 40-49   premeno     20-39      0-8       yes
## 5 40-49   premeno     20-39      0-8       yes
## 6 50-59   premeno     20-39      0-8        no
```

No missing values, so we can apply `bnlearn` directly.

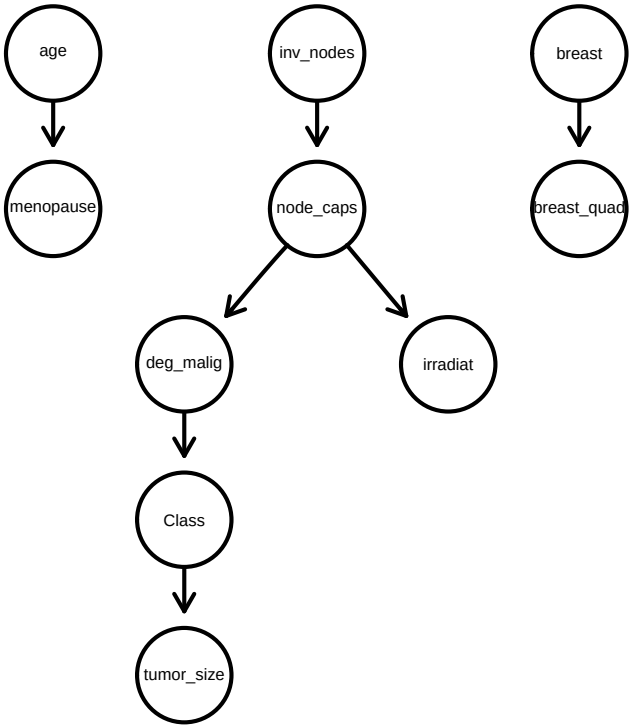
```
anyNA(breast)
```

```
## [1] FALSE
```

Score + search algorithms

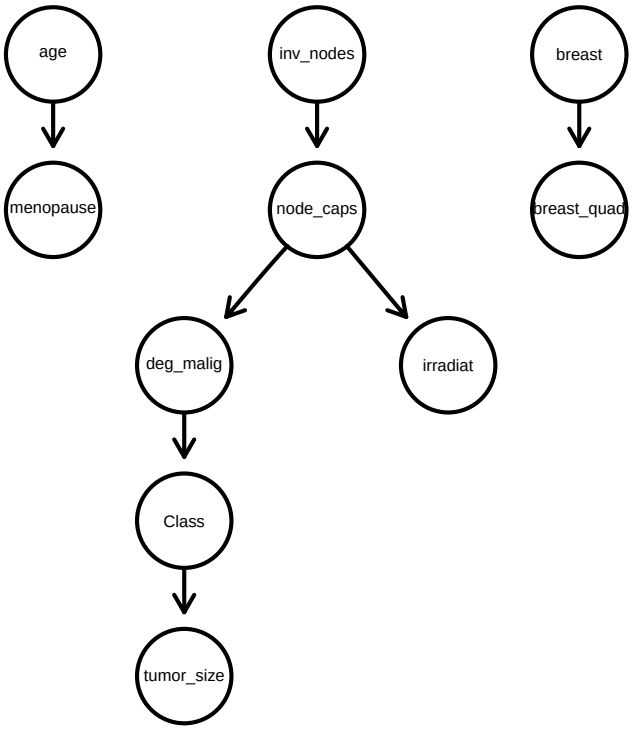
Let us use the hill-climbing (HC) algorithm with the BIC score to learn network structure

```
hc.breast <- hc(breast, score = "bic")
plot(hc.breast)
```



We can see that some of the relationships make sense, such as the arc between age and menopause.

Score + search algorithms



- Name two CIs in $\mathcal{G}$;
- Which nodes suffice to know $P(C)$;
- Any v-structures?
- Revertible arcs?

Useful functions: `dsep()`, `mb()`, `vstruct()`, `cpdag()`.

Scores

`hc` provides a `bn` object, without parameters. We can compute a DAGs score with `score()`. We need to provide data to `score()` in order to learn parameters for the `bn` object.

Compute the log-likelihood on the full data set. Note that the parameters are learned from the second argument, `breast`.

```
score(hc.breast, breast, type="loglik")
```

```
## [1] -2044.865
```

Use `help("bnlearn-package")` to see which scores are available.

Will the BIC score be lower, equal, or larger than the log-likelihood?

And AIC with respect to BIC?

Scores

Which node contributes most to the BIC score? Use the `debug = TRUE` option for `hc()` (in general, `debug = TRUE` gives useful information on `bnlearn` function runs).

```
score(hc.breast, breast, type="bic", debug = TRUE)
```

Take a DAG equivalent to `hc.breast`. Compare its BIC and K2 to those of `hc.breast`.

```
rev.hc.breast <- reverse.arc(hc.breast, from='age', 'menopause')
score(rev.hc.breast, breast, type="bic")
score(rev.hc.breast, breast, type="k2")
```

Scores

Now use the log-likelihood score with hill-climbing. How many free parameters in the network? How many did we have with BIC?

```
loglik.hc.breast <- hc(breast, score='loglik')
nparams(loglik.hc.breast, breast)
```

Find out the penalization coefficient for BIC by print out `hc.breast`.

```
hc.breast
```

If AIC coefficient is 1, which should yield a sparser network with `hc()`? Learn the network with AIC.

Does the AIC model differ from the BIC model? We can bnlearn to compare the structures.

bnlearn comparing structures

- `compare()` takes two graphs (DAGs, CPDAGs, UGs) and returns a list containing tp (true positives), fp (false positives) and fn (false negatives); directed and undirected arcs are considered different.
- `hamming()` computes the Hamming distance between the skeletons of the graphs (zero means a perfect match).
- `shd()` computes the Structural Hamming distance between two CPDAGs, which is similar to the Hamming distance but with a penalty of $\frac{1}{2}$ for directed-undirected arc differences.
- `graphviz.compare()`: compares by plotting the graphs side by side. Wrong (wrt. first argument) directions shown in red and missing arcs in blue.

```
compare(hc.breast, aic.hc.breast, arcs = TRUE)
graphviz.compare(hc.breast, aic.hc.breast)
```

Search

- Identify the first three steps of HC (hint: use `plot()` the `max.iter` argument to `hc()`).
- How did the BIC evolve?

```
## [1] -2226.295 -2211.158 -2199.784
```

Search

Which search operators does HC use? Why did it stop? (hint: run `hc()` with `debug = TRUE`)

We can also see that, in this case, random restarts do not seem to improve the BIC. Thus, gives us some confidence that we are not being trapped in a local optimum.

```
set.seed(100)
hc.breast.restart <- hc(breast, restart = 1000, perturb = 10)
BIC(hc.breast.restart, breast)
```

```
## [1] -2168.594
```

```
all.equal(cpdag(hc.breast.restart), cpdag(hc.breast))
```

```
## [1] TRUE
```

What is the `maxp` argument to `hc`. How can it be useful?
Which other search + score algorithms are available in `bnlearn`?

Constraint-based structure learning

Learn a network with the Grow-shrink algorithm. Is it a DAG? Can we compute its log-likelihood?

```
gs.breast <- gs(x = breast)
```

```
score(gs.breast, breast, type = "loglik")
```

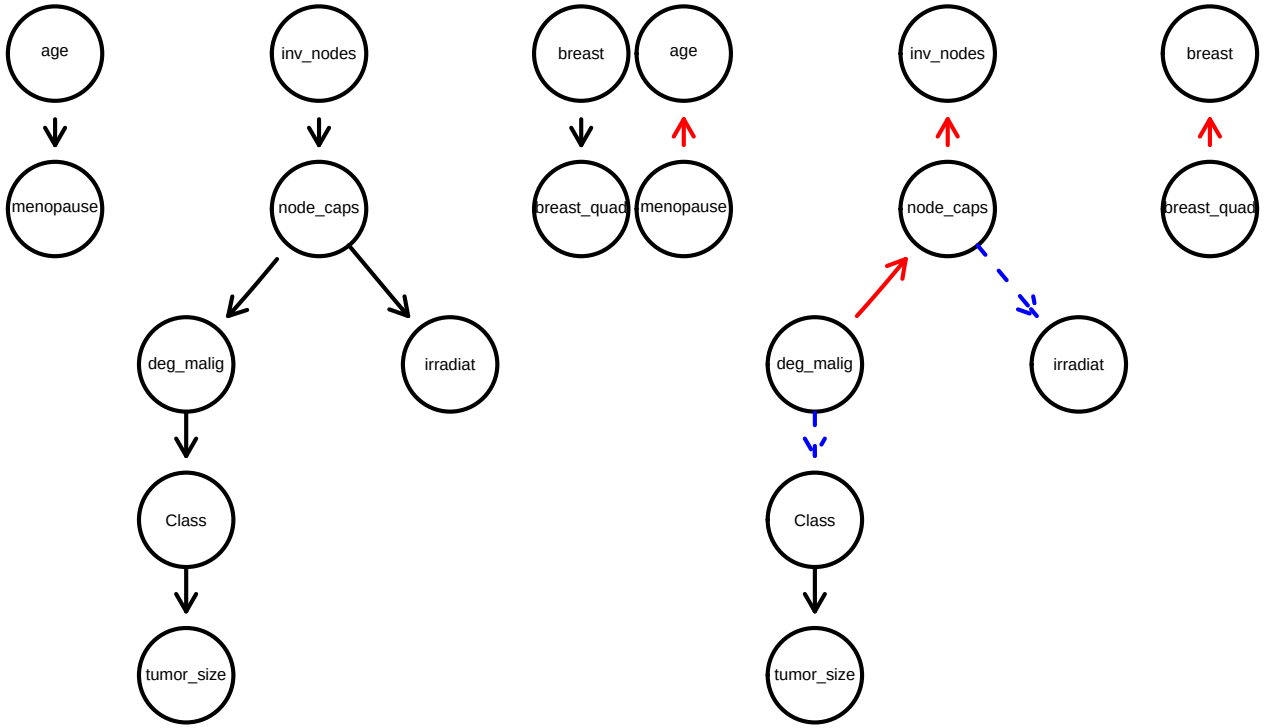
We need to get a consistent extension first.

```
gs.breast <- cextend(gs.breast)
score(gs.breast, breast)
```

```
## [1] -2190.743
```

Constraint-based structure learning

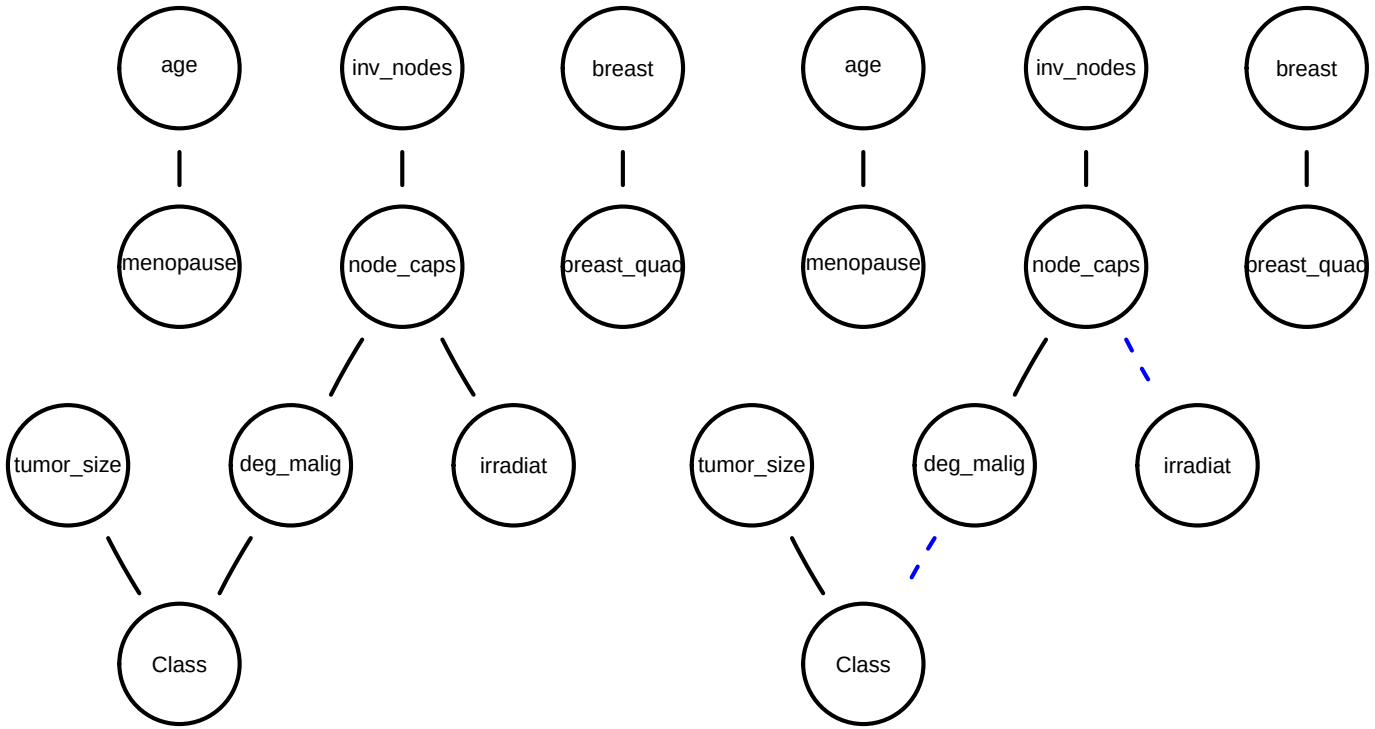
Compare the DAG to those obtained with HC. How many in common?



Constraint-based structure learning

However, it is only meaningfull to compare the CPDAGs. GS has two arcs less, and hence SHD $= 2$.

```
graphviz.compare(cpdag(hc.breast), cpdag(gs.breast))
```




```
shd(gs.breast, hc.breast)
```

```
## [1] 2
```

Constraint-based structure learning

The BIC model has more parameters than the GS one. Compare the BIC and log-lik scores of the two models. Which one has higher BIC? Would this model be a better choice?

By printing the object we can find how many conditional independence (CI) tests were performed to obtain the structure.

```
gs.breast
```

Change the alpha argument so that we only get 5 arcs:

```
gs.breast.string <- gs(breast, alpha=)
narcs(gs.breast.string)
```

Constraint-based structure learning

Use a different test of conditional independence. Use `help("bnlearn-package")` to see the available tests. Does it affect runtime? Is the model very different?

```
gs.breast.2 <- gs(breast, test = )
gs.breast.2 <- cextend(gs.breast.2)
plot(gs.breast.2)
```

Unlike the PC algorithm, the GS algorithm learns local Markov blankets for all variables and then complete the full network. Consider running gs with the debug option.

```
gr.breast <- gs(breast, debug = TRUE)
```

Constraint-based structure learning

Consider another constraint-based algorithm from `bnlearn`. See `?gs` Does loglik improve with respect to GS?

Note that constraint-based algorithms in `bnlearn` are parallelized. On a data set this small parallelization only adds overhead, but is useful for data set with more nodes.

```
library(parallel)
cl <- makeCluster(5)
system.time(gs.breast <- gs(breast, cluster = cl))
system.time(gs.breast <- gs(breast))
stopCluster(cl)
```

```
##    user  system elapsed
##   0.019   0.000   0.398
##    user  system elapsed
##   0.007   0.000   0.007
```

Constraint-based structure learning

Do hill-climbing or iamb generalize better with respect to log-likelihood?

```
bn.cv(breast, 'iamb', loss = "logl")
bn.cv(breast, 'hc', loss = "logl")
```

CLGs

Structure learning functions for Gaussian and conditional linear Gaussian models are the same as for discrete ones. Thus, we can use `hc` as until now. Consider a synthetic data set with discrete and continuous variables:

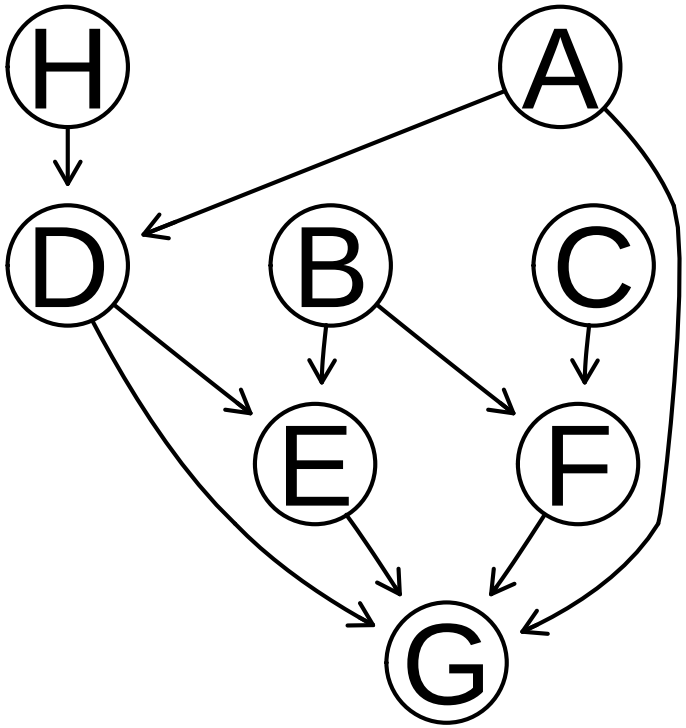
```
data("clgaussian.test")
head(clgaussian.test)
```

```
##   A B C         D        E F          G        H
## 1 a b d  6.460721 11.98657 b  34.84246 2.334846
## 2 b a a 12.758389 30.43674 b 106.63596 2.359112
## 3 b c c 12.175140 17.21532 a  68.92951 2.319435
## 4 b c d 12.006609 14.41646 b  86.17521 2.417494
## 5 b a a 12.328071 30.39631 b 103.58541 2.268150
## 6 b c c 12.613419 15.19344 b  90.84664 2.308369
```

CLGs

Let us learn a conditional linear Gaussian network with `hc()`. Note that continuous variables cannot be parents of discrete ones.

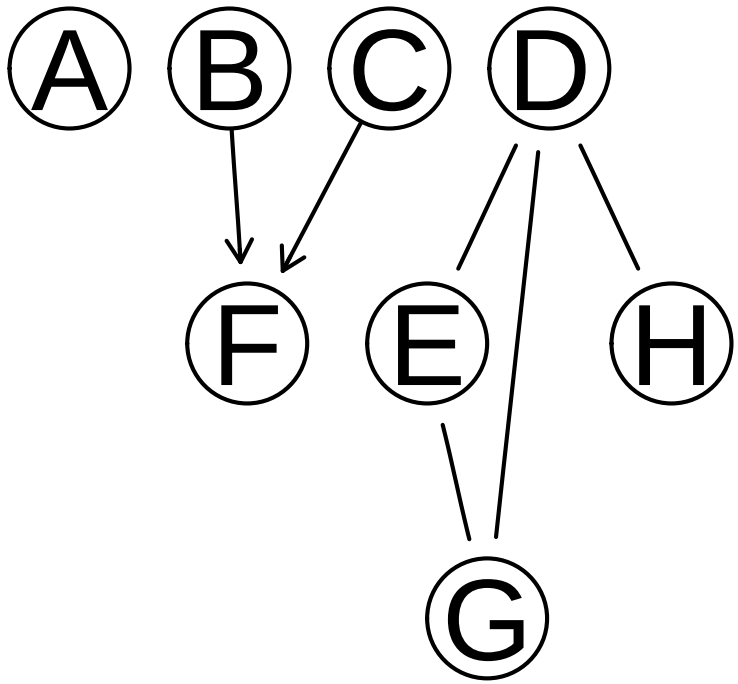
```
plot(hc(clgaussian.test))
```



CLGs

Likewise, we can apply constraint-based algorithms. What changes are the tests that are used.

```
plot(gs(clgaussian.test))
```

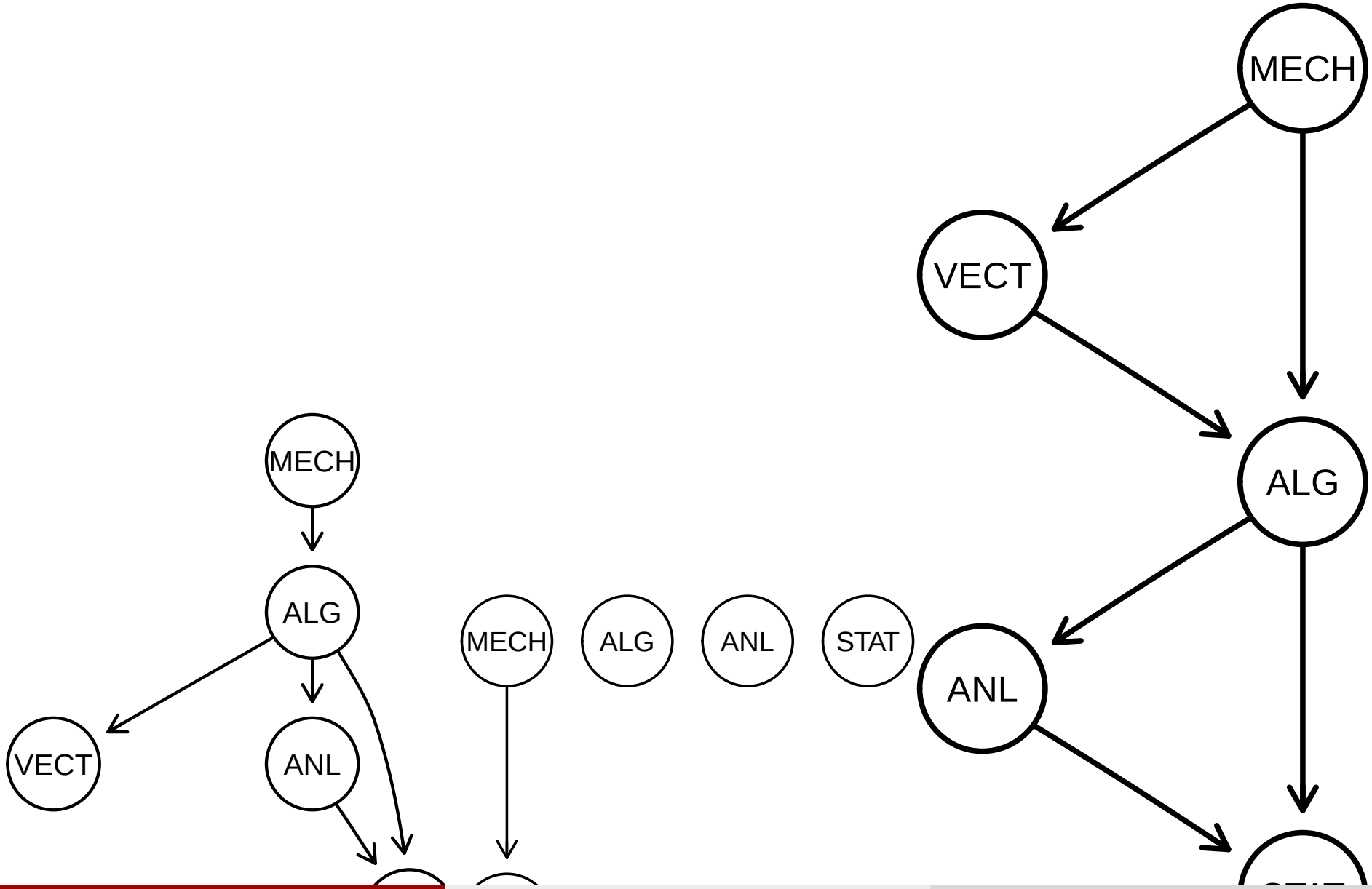


Latent variables

It may be useful to consider the presence of latent variables. Edwards ("Introduction to Graphical Modelling") assigned the students in the marks data set into two groups using an EM algortihm. The networks for the two groups are completely different, and both differ from the overall network.

```
data("marks")
latent <- factor(c(rep("A", 44), "B", rep("A", 7), rep("B", 36)))
plot(hc(marks[latent == "A", ]))
plot(hc(marks[latent == "B", ]))
plot(hc(marks))
```

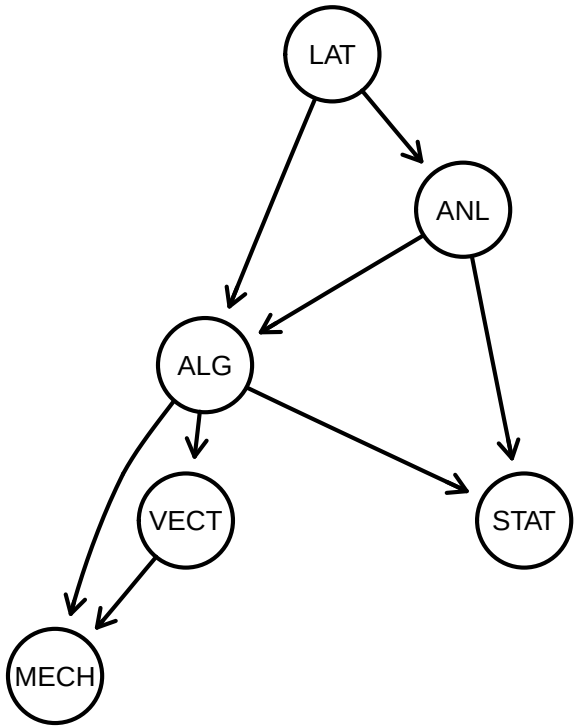
Latent variables



Latent variables

The latent variable it will necessarily be a root of the network, because it is the only discrete variable in our model.

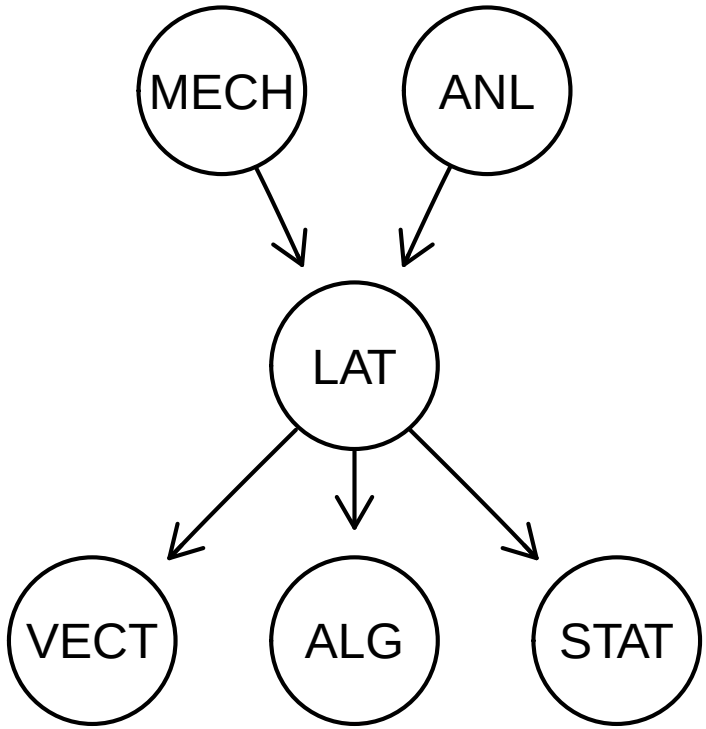
```
lmarks <- data.frame(marks, LAT = latent)
plot(hc(lmarks))
```



Latent variables

If we discretize the network we get yet a different model.

```
dmarks = discretize(marks, breaks = 2, method = "interval")
plot(hc(data.frame(dmarks, LAT = latent)))
```

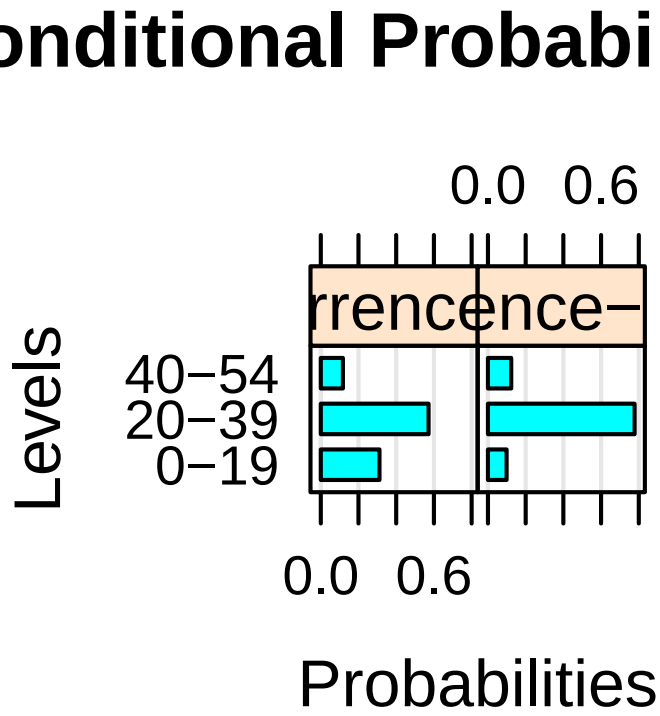


`bnlearn` uses interval, quantile and Hartemink's discretization. While the first two are univariate, the latter tries to maximize mutual information

Parameter estimation

We use `bn.fit` to learn parameters with `bnlearn`

```
breast.bn.fit <- bn.fit(hc.breast, breast)
bn.fit.barchart(breast.bn.fit$tumor_size)
```



Parameter estimation

We see that some probabilities are set to 0.

```
breast.bn.fit$menopause$prob[ ,1:3]
```

```
##           age
## menopause       20-29       30-39       40-49
##    ge40    0.00000000 0.00000000 0.10112360
##    lt40    0.00000000 0.02777778 0.00000000
##    premeno 1.00000000 0.97222222 0.89887640
```

0 probabilities can produce undesirable results. In the Pathfinder study, 10% percent of cases were incorrectly diagnosed due to 0 probabilities —the correct disease was ruled out by a finding that had been given 0 probability (Koller and Friedman, 2006, pp. 67)

Parameter estimation

Avoid 0 probabilities by using Bayesian parameter estimation. `bnlearn` uses a common `iss` parameters for all CPTs in the model. The `iss` is: "the imaginary sample size used by the bayes method to estimate the conditional probability tables (CPTs) associated with discrete nodes".

```
breast.bpe <- bn.fit(hc.breast, breast, method = "bayes", iss = 1)
breast.bpe$menopause$prob[ ,1:3]
```

```
##           age
## menopause       20-29       30-39       40-49
##    ge40    0.047619048 0.001536098 0.101557632
##    lt40    0.047619048 0.029185868 0.000623053
##    premeno 0.904761905 0.969278034 0.897819315
```

Parameter estimation

- Which of the two models has a higher likelihood score?

```
logLik(breast.bn.fit, breast)
logLik(breast.bpe , breast)
```

How could we make all local probability distributions uniform (just as an exercise; otherwise it does not make sense)?

Gaussian networks

bnlearn implements Gaussian networks. Each variable is normally distributed with its mean a linear function of its parents and a standard deviation σ. Learn a structure with hc and then learn the parameters.

```
data(marks)
bn.marks <- hc(marks)
bn.marks <- bn.fit(bn.marks, marks)
coef(bn.marks)$ALG
```

```
## (Intercept)        MECH        VECT
##  25.3619809   0.1833755   0.3577122
```

Gaussian networks

`bnlearn` does not directly implement Bayesian parameter estimation for Gaussian networks. One can use an external package to fit regularized coefficients , instead of ordinary least squares ones, and replace the parameters in the network. LASSO and ridge regression correspond to different priors on the coefficients. Note that excessive regularization, e.g., with LASSO, might lead to zero coefficients which would make a node independent of its parents.

We will use lasso regularization, with $\lambda = 0.01$ to fit the parameters for ALG. Thus, what we would get is lower coefficients and more independence in the network.

Gaussian networks

```
bn.marks.reg <- bn.marks
m <- as.matrix(marks)
# You will need the glmnet package for this
library(glmnet)
gnet <- glmnet(m[ , c('VECT', 'MECH')], m[, 'ALG'], alpha =0, family
coefs <- coef(gnet, s = 0.1)[, 1]
bn.marks.reg$ALG = list(coef = coefs, sd = sigma(bn.marks$ALG))
# cpquery(bn.marks.reg, event = ALG > 50, evidence = TRUE)
bn.marks.reg$ALG$coefficients
```

```
## (Intercept)        MECH        VECT
##  26.4143039   0.1799663   0.3395365
```

```
bn.marks$ALG $coefficients
```

```
## (Intercept)        MECH        VECT
##  25.3619809   0.1833755   0.3577122
```

bnclassify

While `bnlearn` implements discrete naive Bayes and TAN, we will use the `bnclassify` package for discrete classifiers.

The car evaluation data set

- PRICE
  - buying: buying price
  - maint: price of the maintenance
- TECH
  - COMFORT
    - doors: number of doors
    - persons: capacity in terms of persons to carry
    - lug_boot: the size of luggage boot
  - safety: estimated safety of the car
- class: car acceptability

We are interested in car acceptability. It is a function of price and tech, which in turn are functions of the observed variables.

The car evaluation data set

Load the data. All variables discrete.

```
load('data/car.rda')
summary(car)
```

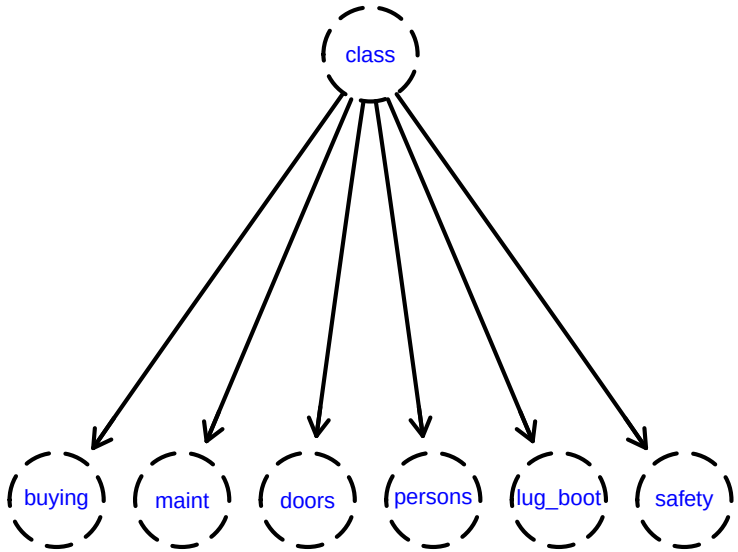
```
##    buying       maint         doors      persons      lug_boot     safe
##  low  :432   high :432   2    :432   2    :576   big  :576   high
##  med  :432   low  :432   3    :432   4    :576   med  :576   low
##  high :432   med  :432   4    :432   more:576   small:576   med
##  vhigh:432   vhigh:432   5more:432
##    class
##  unacc:1210
##  acc  : 384
##  good :  69
##  vgood:  65
```

Naive Bayes

```
library(bnclassify)
```

- Learn a naive Bayes structure from car data. The target variable is 'class'. Plot it.

```
nb.car <- nb(class = 'class', dataset =  )
```



Naive Bayes

- Learn network parameters. What is the class prior probability?

```
nb.car <- lp(nb.car, car, smooth = 0)
params(nb.car)[['class']]
```

```
## class
##      unacc        acc       good      vgood
## 0.70023148 0.22222222 0.03993056 0.03761574
```

Predicting

When only one variable is unobserved, inference amounts to multiplying the entries for the observed ones.

- How could we compute class posterior for first instance using just the CPTs? The features values for the first instance are:

```
car[1, -7]
```

```
##   buying maint doors persons lug_boot safety
## 1  vhigh vhigh     2       2    small    low
```

Predicting

We need to multiply CPT entries. Complete the code below to get the class-conditional probabilities for `lug_boot` and `safety`.

```
b <- params(nb.car)[['buying']]['vhigh', ]
m <- params(nb.car)[['maint']]['vhigh', ]
d <- params(nb.car)[['doors']]['2', ]
p <- params(nb.car)[['persons']]['2', ]
l <- params(nb.car)[['lug_boot']][['', ]
s <- params(nb.car)[['safety']][['', ]
cp <- params(nb.car)[['class']]
cpost <- cp * b * m *  * p *  *
```

Predicting

- Now, multiply the CPT entries accordingly. You should get the following:

```
## class
##       unacc         acc        good       vgood
## 0.001407374 0.000000000 0.000000000 0.000000000
```

- Is this the class posterior distribution? How do we get the class posterior.

Predicting

- Why are there 0 probabilities? Look at the CPT of buying.
- Avoid zero's by using Bayesian parameter estimation (use a Dirichlet prior with hyperparameter `alpha != 0`)

```
nb.car <-
```

Predicting

Get the class posterior for the last five instances:

- What would the predicted class labels be?

```
p <- predict(nb.car, car, prob = TRUE)
tail(p)
```

```
##              unacc         acc        good       vgood
## [1723,] 0.94459821 0.004153620 0.02865175 0.022596424
## [1724,] 0.23381820 0.299936481 0.45723037 0.009014958
## [1725,] 0.12346412 0.230996424 0.24095583 0.404583624
## [1726,] 0.92809628 0.004634127 0.02998218 0.037287418
## [1727,] 0.21719983 0.316377610 0.45235815 0.014064411
## [1728,] 0.09339996 0.198429598 0.19413755 0.514032894
```

Predicting

Get them by dropping the `prob` the argument (or setting it to `FALSE`).

```
p <- predict(nb.car, car)
tail(p)
```

Predicting

How well does naive Bayes learn the training data? Compute accuracy from a confusion matrix.

```
cm <- table(predicted=p, true=car$class)
cm
```

```
##           true
## predicted unacc  acc good vgood
##     unacc  1162   87    0     0
##     acc      46  287   46    30
##     good      2   10   21     0
##     vgood     0    0    2    35
```

```
sum(cm * diag(1, nrow(cm), ncol(cm))) / sum(cm)
```

```
## [1] 0.8709491
```

or use accuracy from bnclassify.

```
bnclassify:::accuracy(p, car$class)
```

Irrelevant features

Is the `doors` feature independent of the class? Use `bnlearn::ci.test` to test.

```
ci.test(...)
```

If we remove it, do you expect naive Bayes to perform better or worse on the training data?

```
car_wo_doors <- car[, -3]
nb.car.wod <- bnc('nb', 'class', dataset = car, smooth = 1)
p <- predict(nb.car.wod, car )
p <- predict(nb.car, car )
bnclassify::accuracy(p, car$class)
```

Correlated features

Is any of the class-conditional independece assumptions violated? E.g., check for `buying` and `maint` given class (use bnlearn):

```
ci.test('maint', 'buying', 'class', car)
```

How does this violation affect classification performance? Let us look at an extreme example: perfect correlation among variables; let us add a copy of the `safety` feature to the data set.

- Should accuracy stay the same/degrade/improve? Run the code below.

```
car2 <- cbind(car, safety2=car$safety)
nb.car2 <- bnc('nb', 'class', car2, smooth = 1)
p <- predict(nb.car2, car2)
bnclassify:::accuracy(p, car2$class)
```

Correlated features

Such assumptions are often violated and degrade the probability estimates, yet that does not necessarily lead to poor classification, as long as the true class has the highest probability.

- What happens if we add 10 copies of `safety` to the data set?

```
sft <- car[, rep(6,10)]
colnames(sft) <- paste0('safety', 1:10)
car2 <- cbind(car, sft)
nb.car2 <- bnc('nb', 'class', car2, smooth = 1)
p <- predict(nb.car2, car2)
bnclassify:::accuracy(p, car2$class)
```

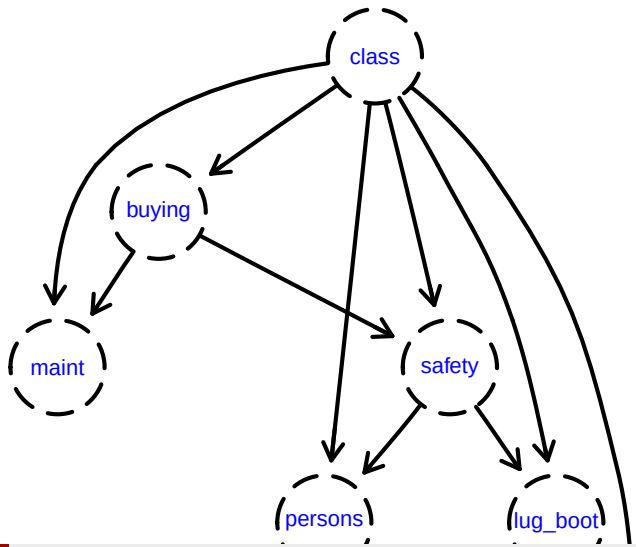
What are the possible approaches for handling feature correlation?

One-dependence augmented naive Bayes

The `tan_cl` function produces a tree augmented naive Bayes (TAN). It uses the Chow-Liu algorithm to maximize (penalized) likelihood in time quadratic in the number of features.

Let us call `tan_cl` without a score argument. The yields the maximum likelihood TAN.

```
tn <- tan_cl('class', car)
tn <- lp(tn, car, smooth = 1)
plot(tn)
```



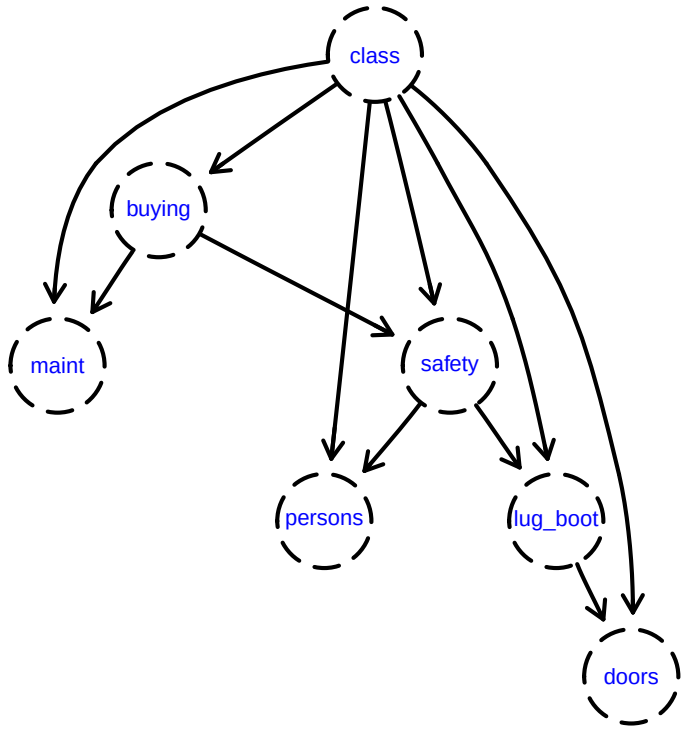
One-dependence augmented naive Bayes

- doors is independent of lug_boot given the class. Why is there an arc between these nodes? How many arcs added?
- Which feature is the root of the predictors tree?

One-dependence augmented naive Bayes

Changing the root produces an equivalent DAG.

```
tns <- tan_cl('class', car, root = 'safety')
plot(tns)
```



One-dependence augmented naive Bayes

```
tns <- lp(tns, car, smooth = 1)
p <- predict(tn, car, prob = TRUE)
ps <- predict(tns, car, prob = TRUE)
identical(p, ps)
```

```
## [1] TRUE
```

One-dependence augmented naive Bayes

- What is the accuracy on the training data?

```
p <- predict(tn, car )
bnclassify::accuracy(p, car$class)
```

Does this model have more free parameters than the naive Bayes? It its likelihood score higher?

```
nparams(...)
nparams(...)
logLik(, car)
logLik(, car)
```

One-dependence augmented naive Bayes

Bayesian information criterion (BIC) and Akaike information criterion (AIC) scores penalize log-likelihood by a factor of model size

$$\text{AIC} = LL(D) - |\theta|$$

$$\text{BIC} = LL(D) - \frac{\log(N)}{2}|\theta|$$

- Which is more restrictive?

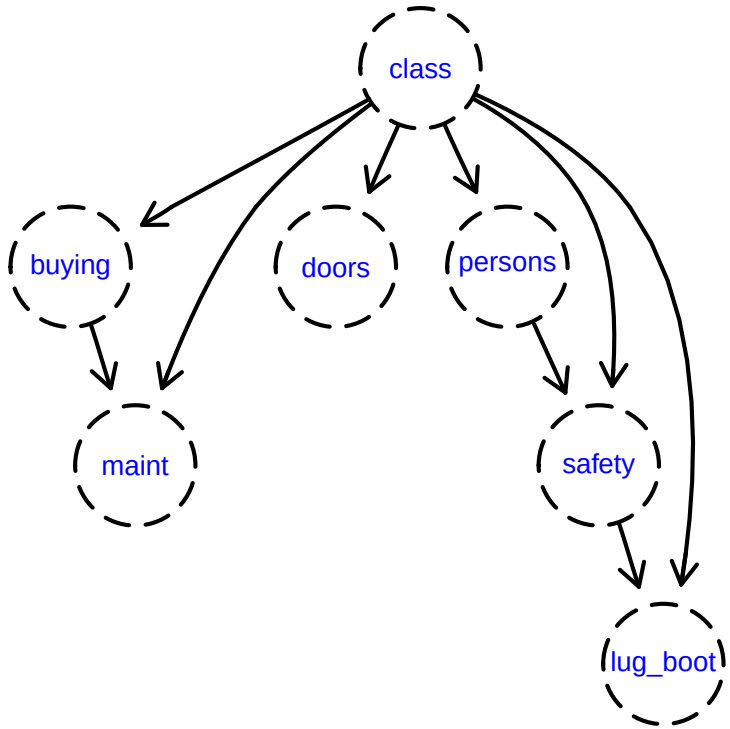
An arc between X_i and X_j contributes $NI(X_i; X_j \mid C) - (r_i - 1)(r_j - 1)r_c$ to AIC and $NI(X_i; X_j \mid C) - \frac{\log(N)}{2}(r_i - 1)(r_j - 1)r_c$ to BIC.

- Can BIC and AIC omit spurious arcs from structure?

One-dependence augmented naive Bayes

Let us call `tan_cl` with the aic score.

```
tn.aic <- tan_cl('class', car, score = 'aic')
tn.aic <- lp(tn.aic, car, smooth = 1)
plot(tn.aic)
```



One-dependence augmented naive Bayes

- Is the predictor subgraph a tree?
- Which arcs are added?
- Is the AIC score of this network higher than that of the maximum likelihood TAN?

```
AIC(object = tn, car)
```

```
## [1] -13461.59
```

```
AIC(object = tn.aic, car)
```

```
## [1] -13438.59
```

One-dependence augmented naive Bayes

- What quantity does the arc between lug_boot and safety contribute to AIC? Compute it using conditional mutual information and the number of added parameters.

```
sl.cmi <- bnclassify:::cmi(, , car, 'class')
# The number of parameters added is:
ap <- (3 - 1) * (3 - 1) * 4
N <-
N * sl.cmi - ap
```

One-dependence augmented naive Bayes

- Would BIC include this arc (between lug_boot and safety)?

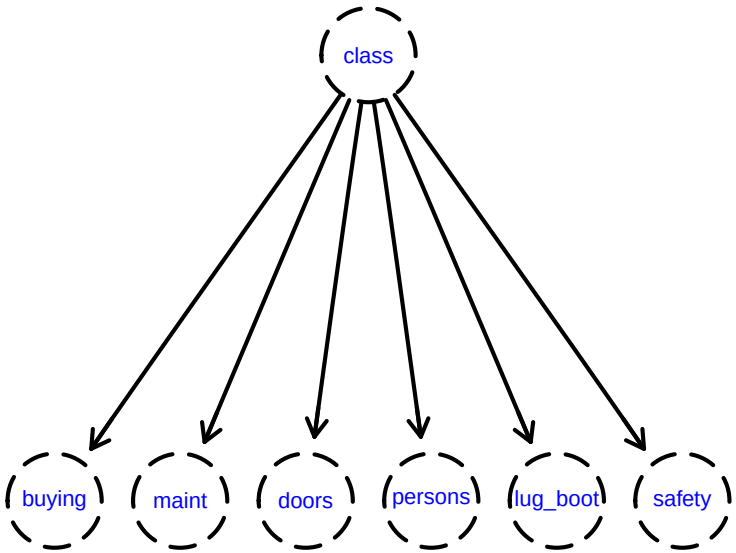
We can get the score contribution of each arc with `local_ode_score`:

```
bnclassify:::local_ode_score_contrib('safety',
                    'lug_boot', 'class', car)
```

One-dependence augmented naive Bayes

On the car data set, BIC adds no arcs.

```
tn.bic <- tan_cl('class', car, score = 'bic')
plot(tn.bic)
```



Comparing classifiers

Which of the above models would you chose? If TAN is more accurate on the training data, is it the better model?

- We can perform cross-validation with `cv`. Before we can use `tn.bic` we need to...?

```
tn.bic <- ...
bnclassify::cv(list(nb.car, tn, tn.aic, tn.bic),
               car, k = 10)
```

Passing multiple models to `cv` uses the same CV partitions to learn and test the classifiers. This allows for paired tests when comparing performance.

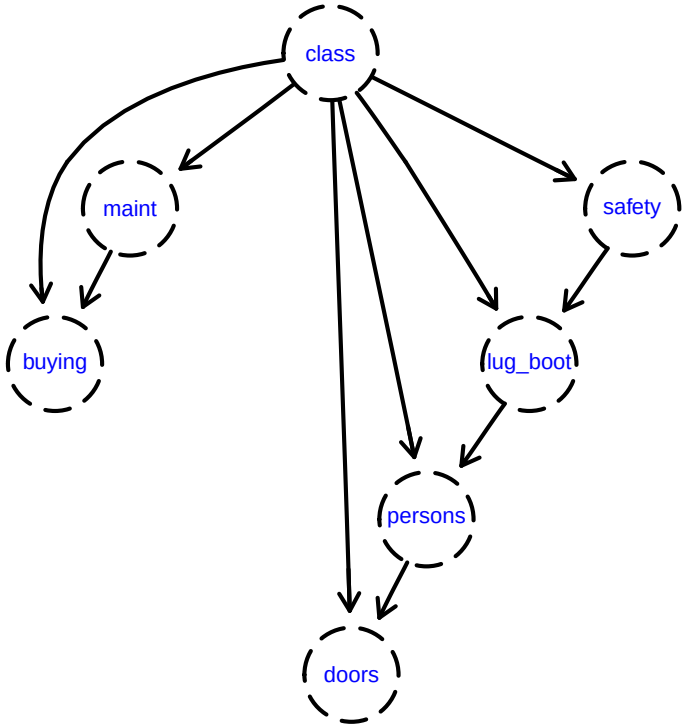
Wrapper structure learning

We can learn one-dependence estimators by maximizing structure accuracy. `tan_hc` greedily adds arcs starting from the naive Bayes until no arc improves accuracy by more than `epsilon = 0`.

```
set.seed(0)
tn.hc <- tan_hc('class', car, k = 10, epsilon = 0, smooth = 1)
```

Wrapper structure learning

- Plot the structure. Similar to Chow-Liu ODE structures?



This is how structure score evolved.

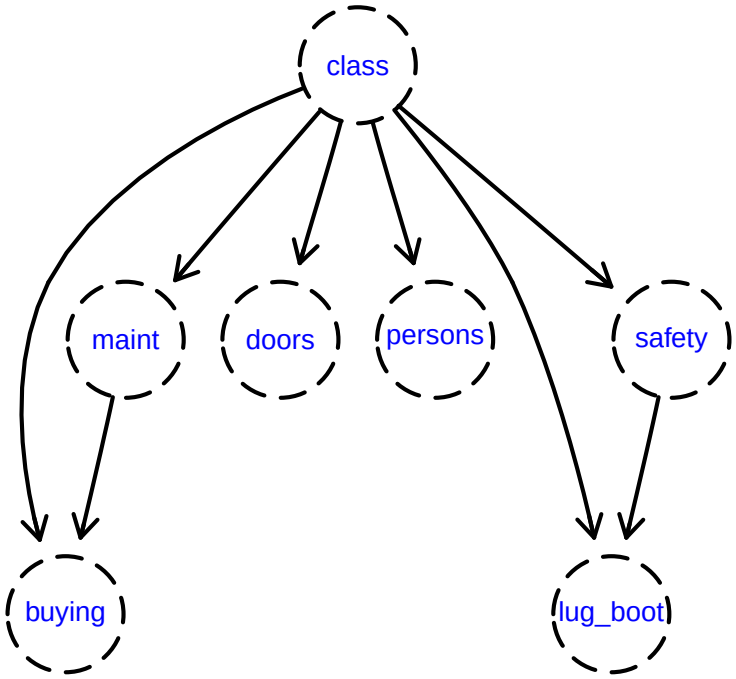
```
tn.hc$.greedy_scores_log
```

```
## [1] 0.8582183 0.9050902 0.9357734 0.9398197 0.9450322
```

Wrapper structure learning

- What if we increased `epsilon`?

```
set.seed(0)
tn.hc2 <- tan_hc('class', car, k = 10, epsilon = , smooth = 1)
```




```
tn.hc2$.greedy_scores_log
```

```
## [1] 0.8582183 0.9050902 0.9357734
```

Wrapper structure learning

- What is the k argument for?

Wrappes are slower than Chow-Liu.

```
system.time(tan_hc('class', car, k = 10, epsilon = 0, smooth = 1))
```

```
##    user  system elapsed
##   0.337   0.004   0.341
```

```
system.time(tan_cl('class', car))
```

```
##    user  system elapsed
##   0.021   0.000   0.021
```

- How could get lower runtimes?

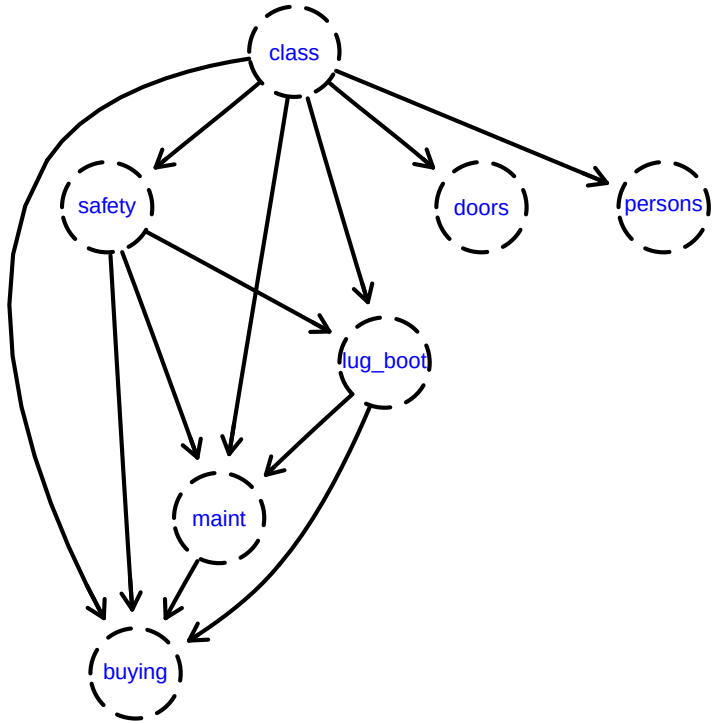
Wrapper structure learning

The semi-naive Bayes `bsej` starts from a naive Bayes and at each step it tries removing features or conditioning them on other features.

```
set.seed(0)
bs.car <- bsej('class', car, k = 10, epsilon = 0, smooth = 1)
```

Wrapper structure learning

- Plot the structure. How many arcs?



Wrapper structure learning

- What are the supernodes? Do they correspond to the hidden PRICE and TECH variables?

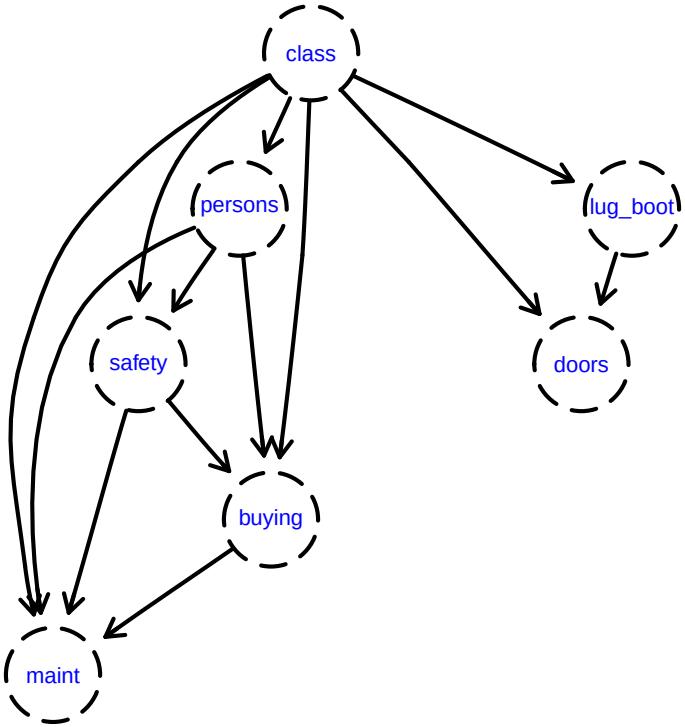
```
bnclassify:::is_supernode(c('maint', 'buying'), bs.car)
```

`fssj` to learns a semi-naive Bayes starting from an empty feature set.

- Call `fssj` (analogously to `bsej`).

Wrapper structure learning

```
set.seed(0)
fs.car <- fssj()
```



- Were any variables omitted?

Table of Contents

Software and resources

Bayesian networks in R

- bnlearn:
  - constraint-based and score + search learning, approximate inference
  - utilities: moralize, make random dag
  - efficient
- gRain:
  - exact inference
  - not very efficient
- gRbase
  - graph utilities: moralize, make random dag
  - largely replaced by bnlearn
- bnclassify: classifiers
- others: pcalg, ggm
- graphical models task view:
  https://cran.r-project.org/web/views/gR.html

Some useful functions

For analysing network structures

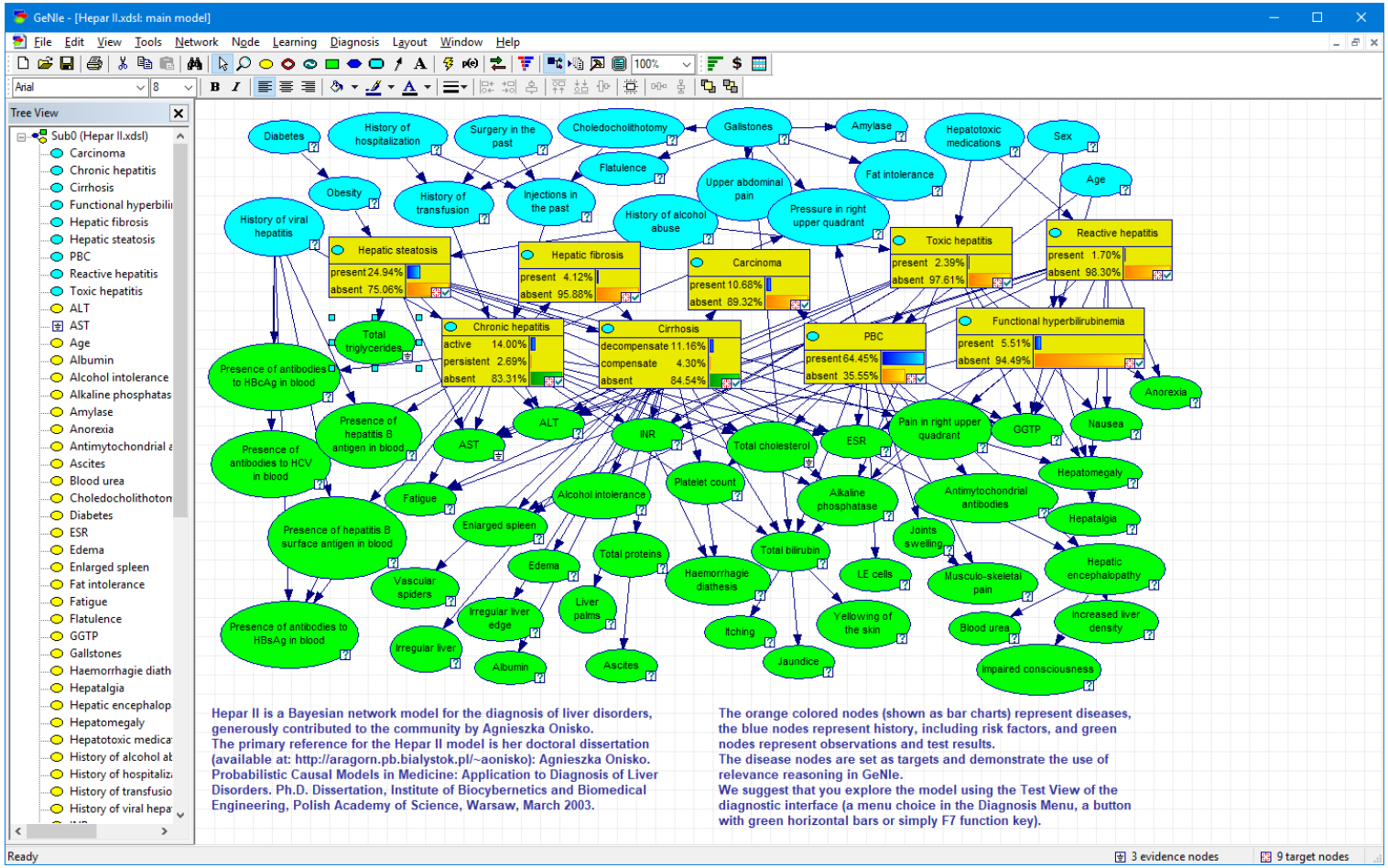
```
bnlearn::mb
bnlearn::ancestors
bnlearn::descendants
bnlearn::children
bnlearn::nparams
bnlearn::model2network
bnlearn::skeleton
bnlearn::compare
bnlearn::cpdag
bnlearn::node.ordering
bnlearn::nodes
bnlearn::dsep
```

Some useful functions

For manipulating network structures

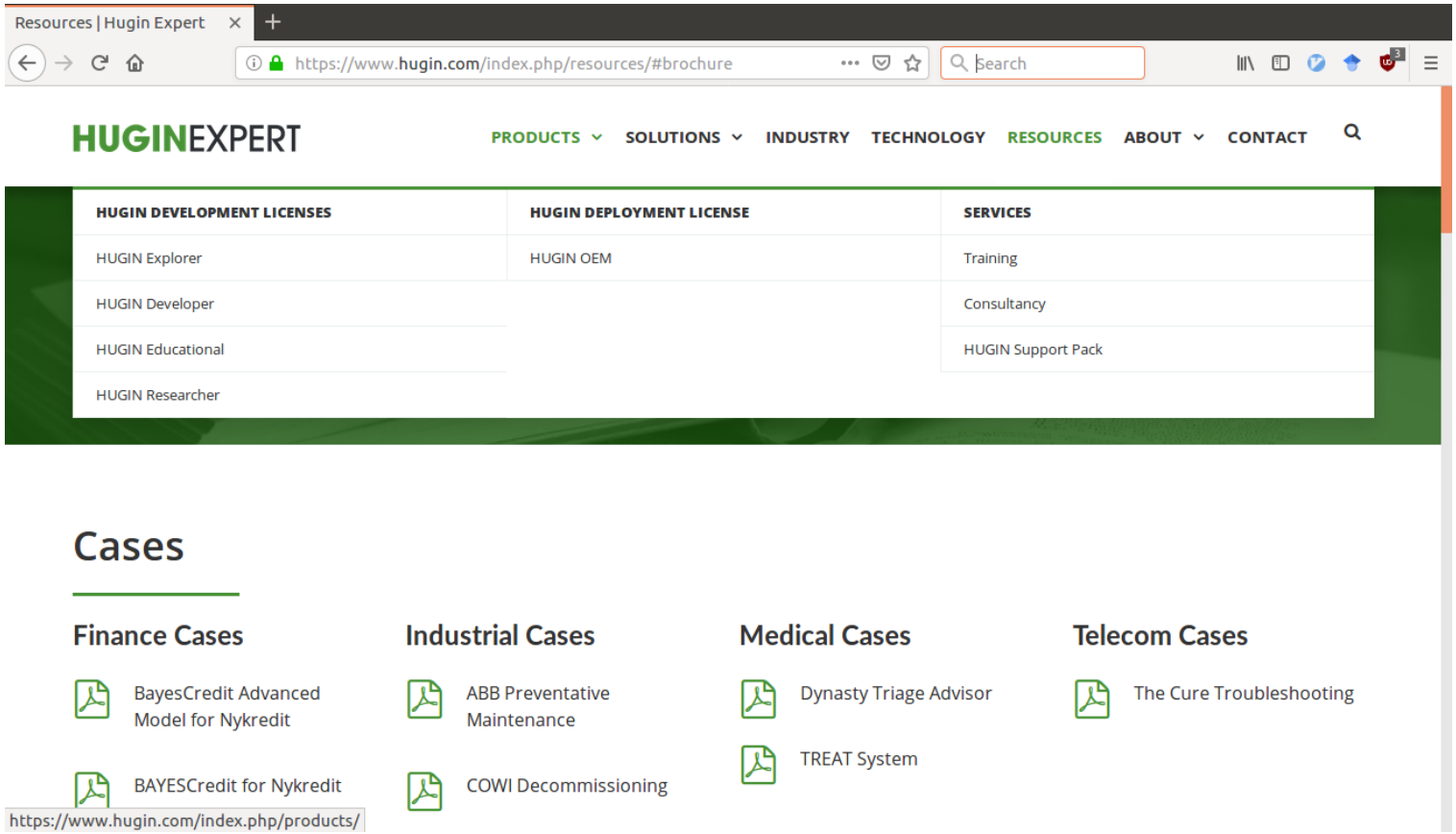
```
gRbase::random_dag
bnlearn::random.graph
bnlearn::empty.graph

bnlearn::set.arc
bnlearn::cextend
gRbase::moralize

gRbase::triangulate
gRbase::getCliques
```

Software



- https://www.bayesfusion.com/genie/
- Free for academic use

Software



- https://www.hugin.com/

Software

Graphical user interface

- GUI: Elvira, Open Markov, ...

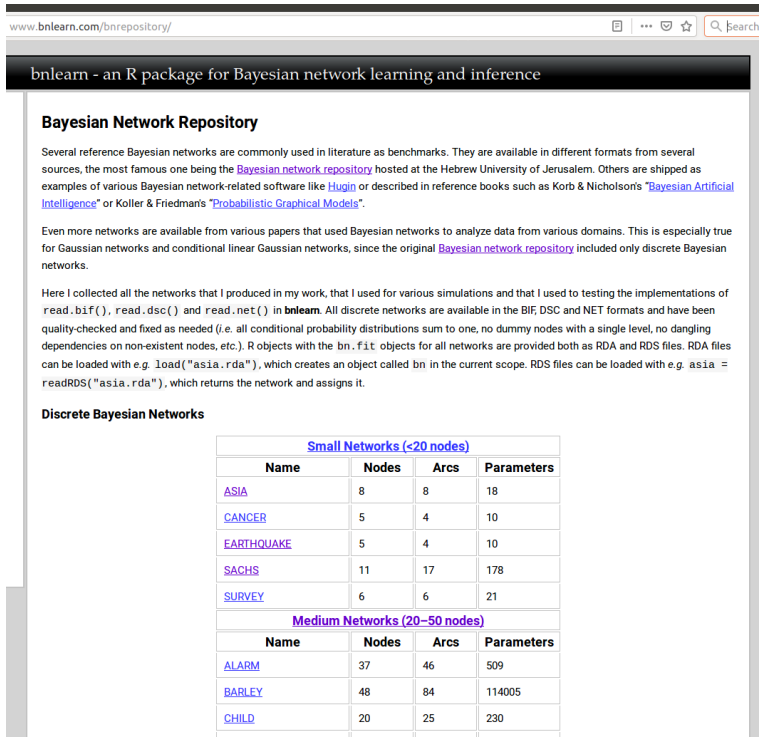
Interpreted languages

- R packages: bnlearn, deal, pcalg, catnet, mugnet, bnclassify, gRbase, gRain
- Matlab: BNT https://github.com/bayesnet/bnt (by Kevin Murphy @ Google)
- Python: http://pgmpy.org/, aGrUM/pyAgrum http://agrum.gitlab.io/

Lists

- https://github.com/topics/bayesian-networks
- http://www.cs.ubc.ca/~murphyk/Software/bnsoft.html (last updated 2014)

bnlearn Networks repository



www.bnlearn.com/bnrepository/

bnlearn - an R package for Bayesian network learning and inference

Bayesian Network Repository

Several reference Bayesian networks are commonly used in literature as benchmarks. They are available in different formats from several sources, the most famous one being the Bayesian network repository hosted at the Hebrew University of Jerusalem. Others are shipped as examples of various Bayesian network-related software like Hugin or described in reference books such as Korb & Nicholson's "Bayesian Artificial Intelligence" or Koller & Friedman's "Probabilistic Graphical Models".

Even more networks are available from various papers that used Bayesian networks to analyze data from various domains. This is especially true for Gaussian networks and conditional linear Gaussian networks, since the original Bayesian network repository included only discrete Bayesian networks.

Here I collected all the networks that I produced in my work, that I used for various simulations and that I used to testing the implementations of `read.bif()`, `read.dsc()` and `read.net()` in **bnlearn**. All discrete networks are available in the BIF, DSC and NET formats and have been quality-checked and fixed as needed (*i.e.* all conditional probability distributions sum to one, no dummy nodes with a single level, no dangling dependencies on non-existent nodes, *etc.*). R objects with the `bn.fit` objects for all networks are provided both as RDA and RDS files. RDA files can be loaded with *e.g.* `load("asia.rda")`, which creates an object called `bn` in the current scope. RDS files can be loaded with *e.g.* `asia = readRDS("asia.rda")`, which returns the network and assigns it.

Discrete Bayesian Networks

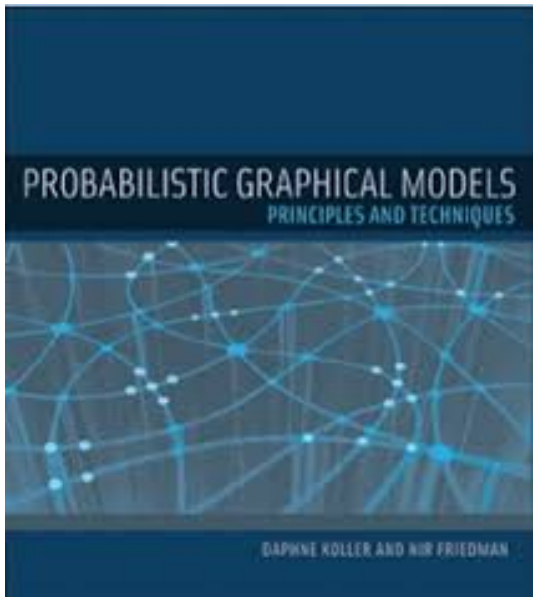
Small Networks (<20 nodes)			
Name	**Nodes**	**Arcs**	**Parameters**
ASIA	8	8	18
CANCER	5	4	10
EARTHQUAKE	5	4	10
SACHS	11	17	178
SURVEY	6	6	21
Medium Networks (20–50 nodes)			
Name	**Nodes**	**Arcs**	**Parameters**
ALARM	37	46	509
BARLEY	48	84	114005
CHILD	20	25	230

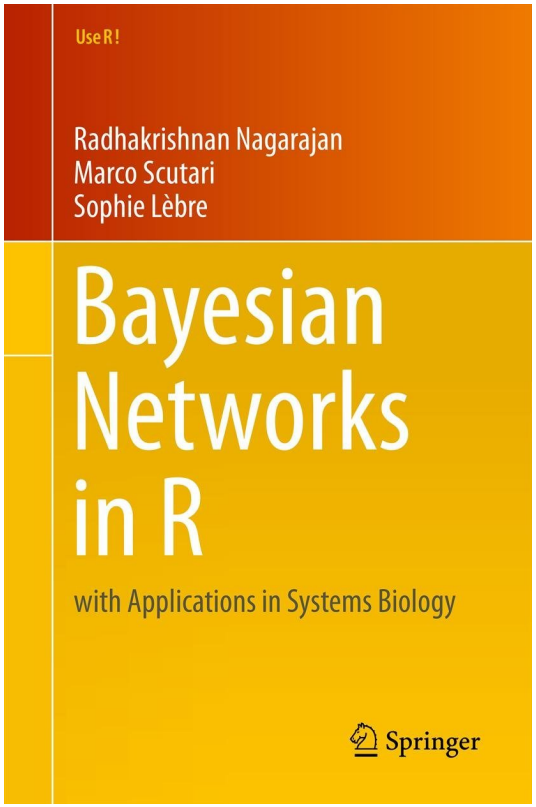
http://www.bnlearn.com/bnrepository/ All networks can be loaded directly into R

```
load(url("http://www.bnlearn.com/bnrepository/sachs/sachs.rda"))
```

Learning Resources

- "Probabilistic graphical models" MOOC (D. Koller @ Stanford): https://class.coursera.org/course/pgm
- Excellent slides: http://www.bnlearn.com/about/teaching/slides-bnshort.pdf
- Network repository: http://www.bnlearn.com/bnrepository/





Summary

Summary

Compact JPD **representation**

- Networks with millions of variables

Algorithms and methods for **inference**

- NP-hard in general
- Efficient for sparse networks
- Exact inference often feasible in practice
- Approximate inference

Algorithms and methods for **learning** from data

- NP-hard in general
- Efficient for sparse networks
- Often reasonable in practice
- Include prior knowledge
- Trade-off between faithfulness and reliability

A workshop on Bayesian networks

Bojan Mihaljevic

Universidad Politécnica de Madrid

Analyx
Poznań, Poland

Table of Contents

Reproducing Sachs, et al.

Reproducing Sachs, et al.

We will reproduce the analysis of Sachs, et al. (2005)

```
sachs = read.table("data/sachs.data.txt", header = TRUE)
head(sachs, n = 5)
```

```
##    Raf  Mek  Plcg  PIP2  PIP3   Erk  Akt PKA   PKC  P38  Jnk
## 1 26.4 13.2  8.82 18.30 58.80  6.61 17.0 414 17.00 44.9 40.0
## 2 35.9 16.5 12.30 16.80  8.13 18.60 32.5 352  3.37 16.5 61.5
## 3 59.4 44.1 14.60 10.20 13.00 14.90 32.5 403 11.40 31.9 19.5
## 4 73.0 82.8 23.10 13.50  1.29  5.83 11.8 528 13.70 28.6 23.1
## 5 33.7 19.8  5.19  9.73 24.80 21.10 46.1 305  4.66 25.7 81.3
```

Reproducing Sachs, et al.

- Apply the Semi-Interleaved HITON-PC algorithm.
- How close it is to the network with previously-known connections? We specify this network below:

```
library(bnlearn)
sachs.modelstring =
paste("[PKC][PKA|PKC][Raf|PKC:PKA][Mek|PKC:PKA:Raf][Erk|Mek:PKA]",
"[Akt|Erk:PKA][P38|PKC:PKA][Jnk|PKC:PKA][Plcg][PIP3|Plcg]",
"[PIP2|Plcg:PIP3]", sep = "")
dag.sachs = model2network(sachs.modelstring)
```

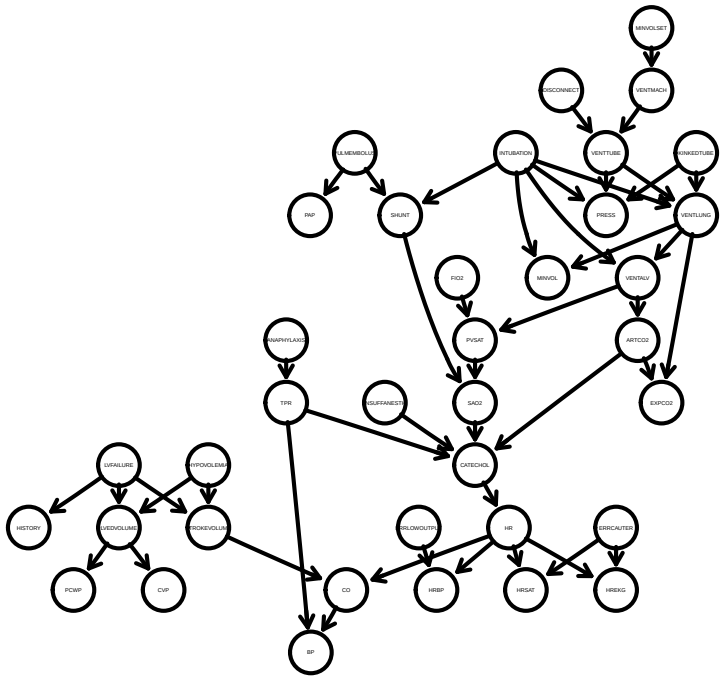
Reproducing Sachs, et al.

- What are the distributional assumptions of our model? What could we do?
- Apply the appropriate data transformation
- Consider a score-based algorithm
- Now consider bootstrapping the algorithm and keeping arcs with 85% confidence
- Compare to the previously-known connections. Note that some of the dependencies discovered with the model were later validated with experiments

An existing network

The ALARM network

```
load('data/alarm.rda')
alarm.fit <- bn
bn <- as.bn(as.grain(alarm.fit))
plot(as.grain(alarm.fit))
```



The ALARM network

There are diagnostic (e.g., intubation, disconnect), intermediate, and measurement nodes.

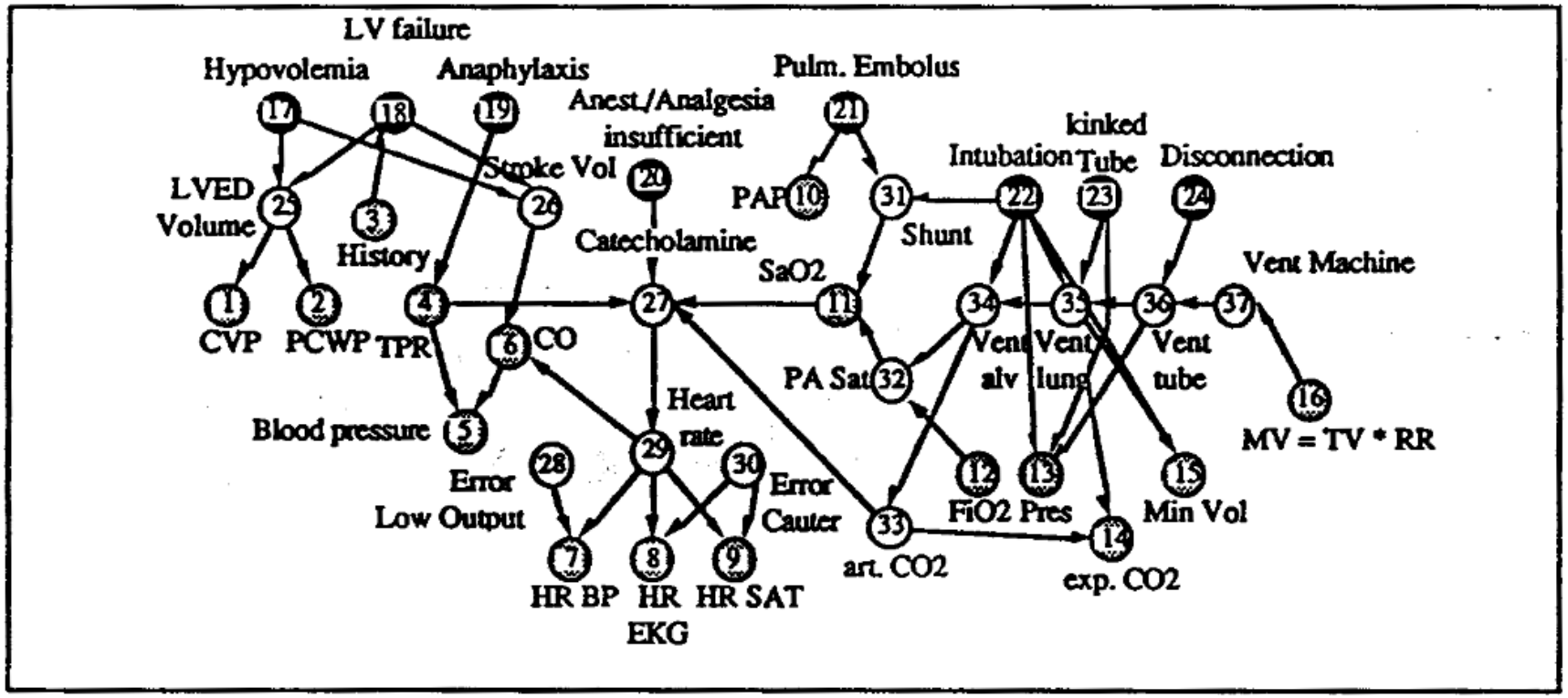


Fig. 1 The ALARM network representing causal relationships is shown with diagnostic (●), intermediate (○) and measurement (◎) nodes. CO: cardiac output, CVP: central venous pressure, LVED volume: left ventricular end-diastolic volume, LV failure: left ventricular failure, MV: minute ventilation, PA Sat: pulmonary artery oxygen saturation, PAP: pulmonary artery pressure, PCWP: pulmonary capillary wedge pressure, Pres: breathing pressure, RR: respiratory rate, TPR: total peripheral resistance, TV: tidal volume

Inspect the network

- How many nodes?
- How many arcs?
- What is the average neighborhood size?
- What are the v-structures in the graph? See `vstruct` in `bnlearn`.
- What are the parents of HISTORY?
- What is the Markov blanket of LVED VOLUME?
- What is the neighborhood of STROKEVOLUME?
- Are the local distributions discrete or Gaussian?

Recover network structure

1. Sample from the original network
2. Run a structure learning algorithm of your choice
3. Compare its structural distance to the original network
4. is it an I-map of the original distribution (i.e., network?)?
5. Is its likelihood score higher than that of the original network?
6. What is the accuracy for predicting INTUBATION?
7. What is the accuracy for predicting INTUBATION with classifier learning?

Improve learning

1. Try different options of the algorithm you just used (e.g,. independence tests)
2. Try other algorithms (score + search, constraint-based, hybrid)
3. Consider parallelizing CB algorithms
4. Which algorithms' models score similarly to the original network?
5. Which has the best cross-validated loglik score?

Inference

Go back to the original network

- What is the marginal probability of HYPOVOLEMIA?
- What is the map assignment to INSUFF ANESTH and CATECHOL?
- What is the joint probability of SHUNT and CATECHOL?
- Which is the largest clique in the junction tree?
- Find the probability of an event over 9 different variables.
- Condition on 3 variables. Find the probability of an event over 9 different variables.
- Find the marginal probability of HYPOVOLEMIA with sampling-based inference. Is it close to the exact probability?
- Find the MAP assignment to INSUFF ANESTH and CATECHOL with sampling-based inference
- Set evidence in a number of leaf nodes, and estimate the marginal over an intermediate node. What is the absolute error of particle-based inference do with 1e5 samples?