Bayesian Networks in Policy and Society

Day 1: Definitions and Fundamentals

Marco Scutari

Dalle Molle Institute for
Artificial Intelligence (IDSIA)

August 8, 2022

Machine Learning: The Grand Vision

Machine learning studies the algorithms and the statistical tools that allow computer systems to perform specific, well-defined tasks without explicit instructions. It is a sub-field of artificial intelligence.

Broadly speaking, in order to do this:

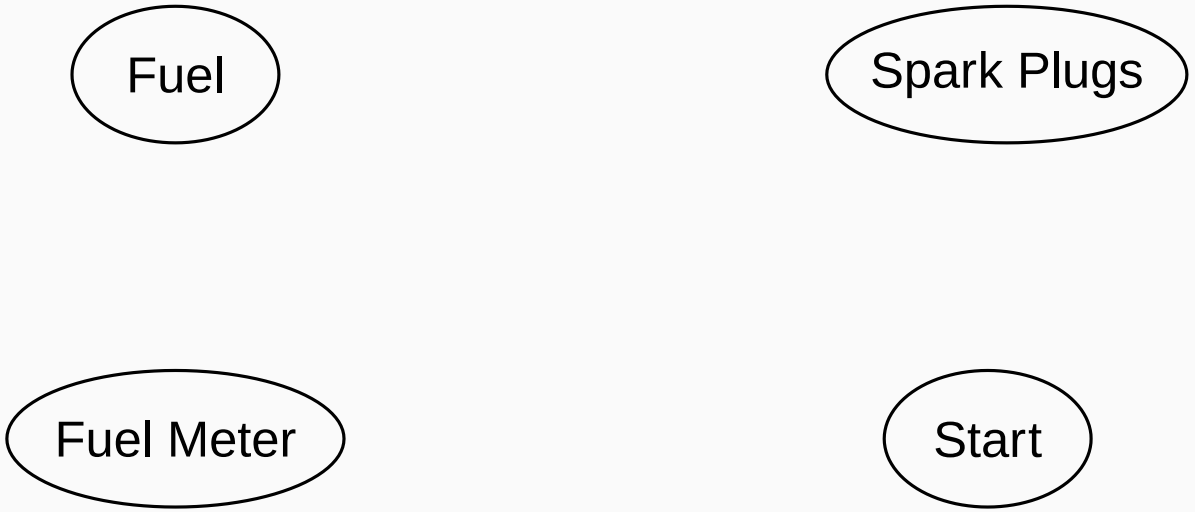
1. We need a working model of the world that describes the task and its context in a way a computer can understand.
2. We need a goal: how do we measure the performance of the model? Because that is what we optimise for! Usually it is the ability to predict new events.
3. We encode our knowledge of the world drawing information from training data, experts or both: this is called learning.
4. The computer system uses the model as a proxy of reality and to perform inference and decide if/how to perform the assigned task as new inputs come in.

Identify the Variables to Include in the Model

The first step in building a machine learning model is to choose which variables to include. Which aspects of/entities in the world do we need the model to represent for the computer to carry out the assigned task? This is known as feature selection.

- Each aspect of the world or entity is modelled with one random variable.
- We should use a small enough number of variables because if we have too many:
  - it is difficult it is to construct the model;
  - it is difficult to interpret and to troubleshoot it;
  - the model requires too much computing power to learn and to run.
- We must choose which are the relevant events that make up the sample space of each variable, again taking care of not having too many.

An example: The Car Start Problem



	Realistic	Pragmatic
Fuel	0%–100%	Yes, No
Spark Plugs	Work, Fault	Work, Fault
Start	Yes, No	Yes, No
Fuel Meter	0%–100%	Empty, Half, Full

Generative versus Discriminative Models

The second step is choosing which class of machine learning models to select from.

- Generative models: we have a set of variables $X_1, \ldots, X_N$ describing various components of a complex phenomenon, and we are interested in building a mechanistic model of that phenomenon to understand it. Therefore, we want to show how the various parts interact with each other. In order to do so we choose to model their joint probability $\mathrm{P}(X_1, \ldots, X_N)$.
- Discriminative models: we have one particular variable (say, X_1) that is closely tied with our model task, and a number of other variables ($X_2, \ldots, X_N$) which we believe can be used to predict it. We do not care about how the X_i are related to each other, so we just model $\mathrm{P}(X_1 \mid X_2, \ldots, X_N)$.

Model Relationships Between Variables

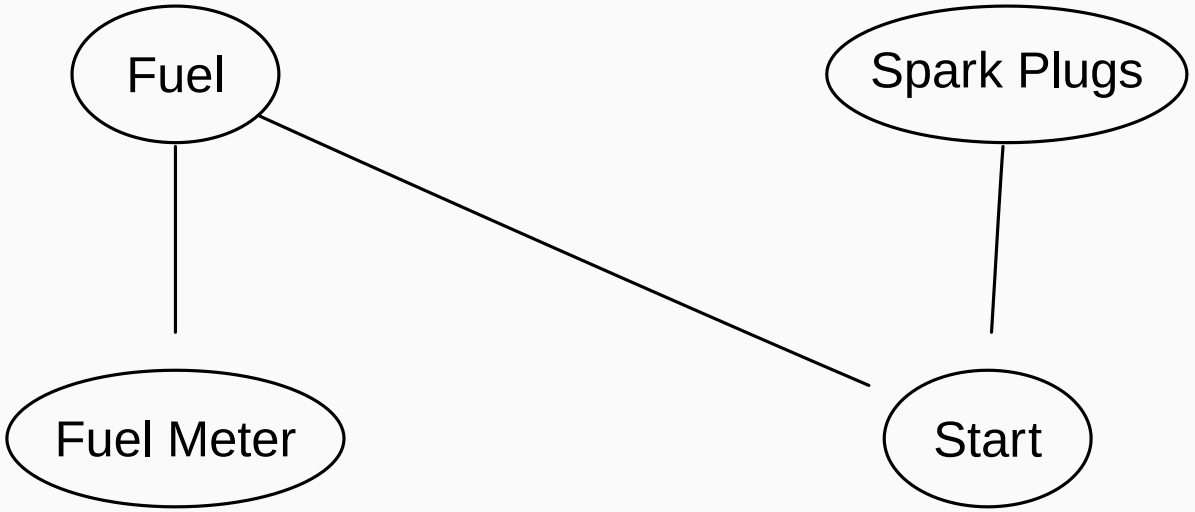
How do we decide whether there is a relationships between variables?

We never have perfect knowledge of what we are modelling: hence we use the language of probability, and we say that two variables are associated if the occurrence of an event in one variable affects the probability of an event occurring in another variable, possibly conditional on other variables.

How can we acquire information on what we are modelling:

- consulting domain experts;
- using probability and statistics to extract it from data;
- a combination of both.

The Car Start Problem, with Edges



- The `Fuel Meter` measures the amount of `Fuel`?
- The `Spark Plugs` ignite the `Fuel` to `Start` the car?
- If there is `Fuel` in the car, it can start even if the `Fuel Meter` is wrong and displays 0%?

Car Start: Probabilistic versus Causal Construction

In probability associations are symmetric: the derivation of Bayes' theorem makes it really clear that

$$\mathrm{P}(X_1 \mid X_2)\,\mathrm{P}(X_2) = \mathrm{P}(X_1, X_2) = \mathrm{P}(X_2 \mid X_1)\,\mathrm{P}(X_1).$$

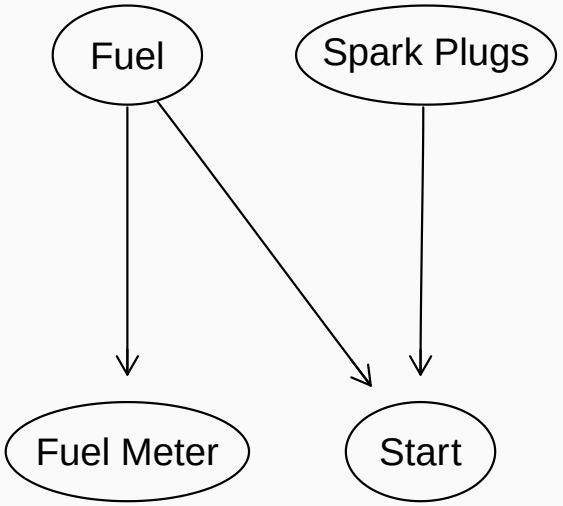
However, it feels more natural to choose the conditioning variables such that they affect the conditioned variables instead of the other way round.

But what does that mean from a modelling point of view? It means that we are giving arcs a causal interpretation and that we choose arc directions to go from cause (nodes) to effect (nodes).

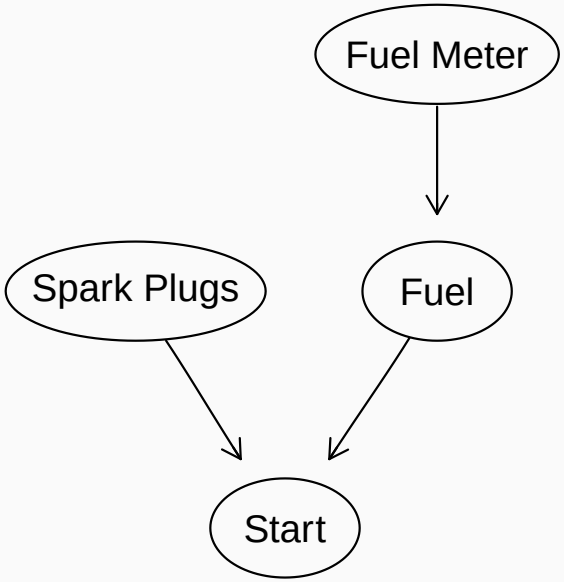
How do we do that?

The Car Start Problem, with Arcs

$$\begin{aligned}
&\mathrm{P}(\texttt{Start}, \texttt{Fuel Meter}, \texttt{Fuel}, \texttt{Spark Plugs}) \\
&\quad = \mathrm{P}(\texttt{Start} \mid \texttt{Fuel}, \texttt{Spark Plugs}) \times \\
&\qquad \mathrm{P}(\texttt{Fuel Meter} \mid \texttt{Fuel})\, \mathrm{P}(\texttt{Fuel}) \times \\
&\qquad \mathrm{P}(\texttt{Spark Plugs})
\end{aligned}$$



$$\begin{aligned}
&\mathrm{P}(\texttt{Start}, \texttt{Fuel Meter}, \texttt{Fuel}, \texttt{Spark Plugs}) \\
&= \mathrm{P}(\texttt{Start} \mid \texttt{Fuel}, \texttt{Spark Plugs}) \times \\
&\quad \mathrm{P}(\texttt{Fuel Meter})\, \mathrm{P}(\texttt{Fuel} \mid \texttt{Fuel Meter}) \times \\
&\quad \mathrm{P}(\texttt{Spark Plugs})
\end{aligned}$$



Car Start: Playing with Arc Directions

The criterion to identify causes and effect is intervention. Consider:

- If we fill the tank with fuel, the fuel meter goes up.
- If we tamper with the fuel meter to make is say `Full`, the fuel tank does not magically refill itself.

Hence, `Fuel` is the cause and `Fuel Meter` is the effect and the most intuitive arc direction is `Fuel` $\rightarrow$ `Fuel Meter`.

What the probability $\mathrm{P}(\texttt{Fuel Meter} \mid \texttt{Fuel})$ tells us is just that if the fuel meter says `Full` there probably is fuel in the tank, whereas if the fuel meter says `Empty` there may be no fuel in the tank (assuming the fuel meter works reliably).

Car Start: The Conditional Probabilities

Spark Plugs

Work	Fault
?	?

Fuel

Yes	No
?	?

Fuel Meter

	Fuel = Yes	Fuel = No
Empty	?	?
Half	?	?
Full	?	?

Start

	Spark Plugs = Work	
	Fuel = Yes	Fuel = No
Yes	?	?
No	?	?

	Spark Plugs = Fault	
	Fuel = Yes	Fuel = No
Yes	?	?
No	?	?

After we decide that the first model is good to go, we need to:

1. choose which distribution to use for each node;
2. fill in the values of its parameters by asking domain experts, estimating them from data or a combination of the two.

The number of parameters gives the complexity of the model, rather than the number of nodes or the number of arcs.

Car Start: Interrogating the Model

A more general way of using a model is to interrogate it: we have some evidence on some of the variables (that is, we assume we know their values), and we would like to know the the probability of some event.

For instance: say that $\texttt{Fuel Meter} = \texttt{Half}$. How does $\mathrm{P}(\texttt{Start} = \texttt{Yes})$ change after we introduce this evidence in the model?

Predicting `Start` from all the other variables is a particular case in which we have evidence on all the other variables.

Car Start: The Exhaustive (Dumb) Way

Armed with patience, we start by writing

$$\mathrm{P}(\texttt{Start} = \texttt{Yes}) = \\ \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}) + \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{No})$$

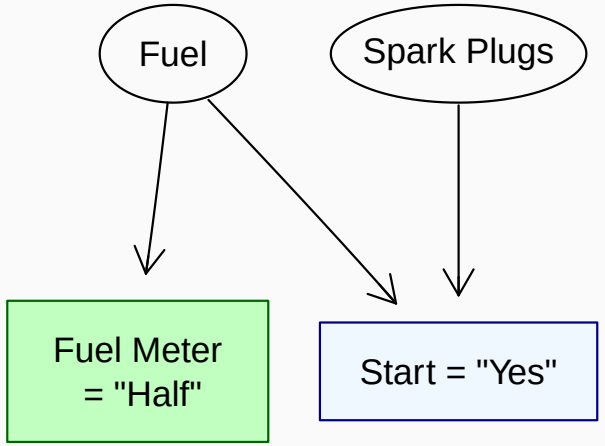
and then, recursively,

$$\mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}) = \\ \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}, \texttt{Spark.Plugs} = \texttt{Work}) + \\ \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}, \texttt{Spark.Plugs} = \texttt{Fault})$$

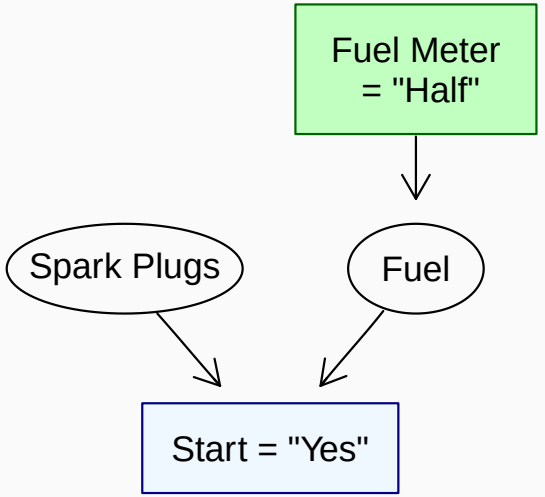
$$\mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}, \texttt{Spark.Plugs} = \texttt{Work}) \\ = \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}, \texttt{Spark.Plugs} = \texttt{Work}, \texttt{Fuel.Meter} = \texttt{Full}) + \\ \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}, \texttt{Spark.Plugs} = \texttt{Work}, \texttt{Fuel.Meter} = \texttt{Half}) + \\ \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}, \texttt{Spark.Plugs} = \texttt{Work}, \texttt{Fuel.Meter} = \texttt{Empty})$$

Car Start: The Principled (Probabilistic) Way

$$\begin{aligned}
&\mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel Meter} = \texttt{Half},\\
&\qquad \texttt{Fuel}, \texttt{Spark Plugs}) =\\
&= \mathrm{P}(\texttt{Start} = \texttt{Yes} \mid \texttt{Fuel}, \texttt{Spark Plugs}) \times\\
&\quad \mathrm{P}(\texttt{Fuel Meter} = \texttt{Half} \mid \texttt{Fuel}) \times\\
&\quad \mathrm{P}(\texttt{Fuel})\,\mathrm{P}(\texttt{Spark Plugs})
\end{aligned}$$

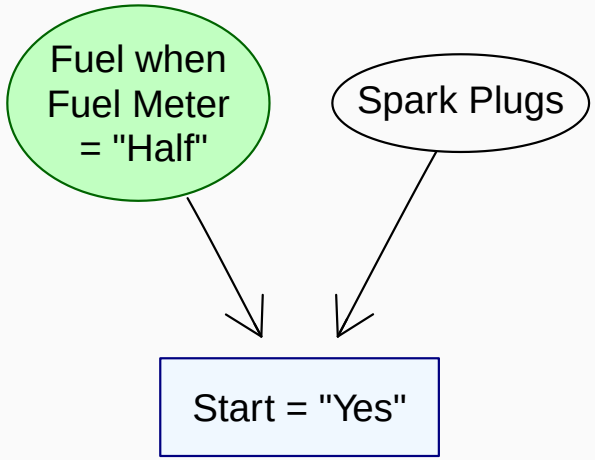


$$\begin{aligned}
&\mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel Meter} = \texttt{Half},\\
&\qquad \texttt{Fuel}, \texttt{Spark Plugs})\\
&= \mathrm{P}(\texttt{Start} \mid \texttt{Fuel}, \texttt{Spark Plugs}) \times\\
&\quad \mathrm{P}(\texttt{Fuel} \mid \texttt{Fuel Meter} = \texttt{Half}) \times\\
&\quad \frac{\mathrm{P}(\texttt{Fuel.Meter} = \texttt{Half})}{\cancel{\mathrm{P}(\texttt{Fuel})}} \times\\
&\quad \cancel{\mathrm{P}(\texttt{Fuel})}\,\mathrm{P}(\texttt{Spark Plugs})
\end{aligned}$$



Car Start: The Principled (Probabilistic) Way

$$\begin{aligned}
&\mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel}, \texttt{Spark Plugs} \mid \\
&\qquad \texttt{Fuel Meter} = \texttt{Half}) \\
&= \mathrm{P}(\texttt{Start} \mid \texttt{Fuel}, \texttt{Spark Plugs}) \times \\
&\qquad \mathrm{P}(\texttt{Fuel} \mid \texttt{Fuel Meter} = \texttt{Half}) \times \\
&\qquad \frac{\mathrm{P}(\texttt{Fuel.Meter} = \cancel{\texttt{Half}})}{\mathrm{P}(\cancel{\texttt{Fuel}}\texttt{.Meter} = \texttt{Half})} \times \\
&\qquad \mathrm{P}(\texttt{Spark Plugs})
\end{aligned}$$



This leaves three variables, of which Start is fixed to Yes: hence we have to consider $\mathrm{P}(\texttt{Start} = \texttt{Yes})$ under four scenarios:

$$\begin{aligned}
&\texttt{Fuel} = \texttt{Yes} \mid \texttt{Fuel Meter} = \texttt{Half}, \texttt{Spark Plugs} = \texttt{Work} \\
&\texttt{Fuel} = \texttt{Yes} \mid \texttt{Fuel Meter} = \texttt{Half}, \texttt{Spark Plugs} = \texttt{Fault} \\
&\texttt{Fuel} = \texttt{No} \mid \texttt{Fuel Meter} = \texttt{Half}, \texttt{Spark Plugs} = \texttt{Work} \\
&\texttt{Fuel} = \texttt{No} \mid \texttt{Fuel Meter} = \texttt{Half}, \texttt{Spark Plugs} = \texttt{Fault}
\end{aligned}$$

and sum the corresponding $\mathrm{P}(\texttt{Start} = \texttt{Yes} \mid \text{scenario})\,\mathrm{P}(\text{scenario})$.

Car Start: The Principled (Algorithmic) Way

Exhaustive enumeration obviously:

1. does not scale (try that with 20 variables!);
2. is only feasible in the first place if all variables are discrete.

Each of the steps in the previous slide corresponds to both

- an (probabilistic) application of Bayes theorem
- a (graphical) manipulation of arcs and nodes.

We can use the fact that arcs represent probabilistic associations to perform symbolic computations though graphical operations!

We can also threat this model like a hierarchical model and adapt the literature on Monte Carlo simulations.

Automating Modelling and Reasoning

Now we want to automate the whole process, so that the computer system itself will (ideally) do all the work.

A model that promises to do this is Bayesian networks (BNs):

- They combine graphs and probability as we did earlier, but in a rigorous fashion.
- There are automated reasoning algorithms for that use the graphical part of the model to guide a computer system in manipulating probability distributions, computing probabilities of and predicting events of interest.
- It is possible to learn them automatically from data.
- They can be used as causal models.
- As far as models, go they are very green: they recycle large amounts of results from classical statistics.

Bayesian Networks in R: The **bnlearn** Package

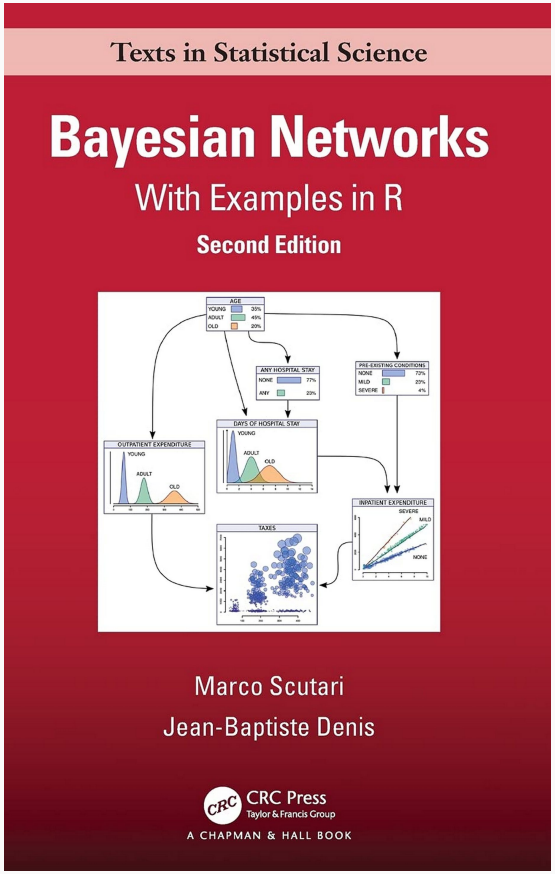
The most comprehensive R package for working with Bayesian networks is **bnlearn**, which you should install by

```
install.packages("bnlearn")
```

The reference website for **bnlearn** is:

http://www.bnlearn.com

And there is a reference book too!



A Graph and a Probability Distribution

Bayesian networks (BNs) are defined by:

- a network structure, a directed acyclic graph $\mathcal{G}$, in which each node corresponds to a random variable X_i;
- a global probability distribution over $\mathbf{X} = \{X_1, \dots, X_N\}$ which can be factorised into smaller local probability distributions according to the arcs present in the graph.

The main role of the network structure is to express the conditional independence relationships among the variables in the model through graphical separation, thus specifying the factorisation of the global distribution:

$$\mathrm{P}(\mathbf{X}; \boldsymbol{\Theta}) = \prod_{i=1}^{N} \mathrm{P}(X_i \mid \Pi_{X_i}; \Theta_{X_i}) \quad \text{where} \quad \Pi_{X_i} = \{\text{parents of } X_i\}.$$

Graphs

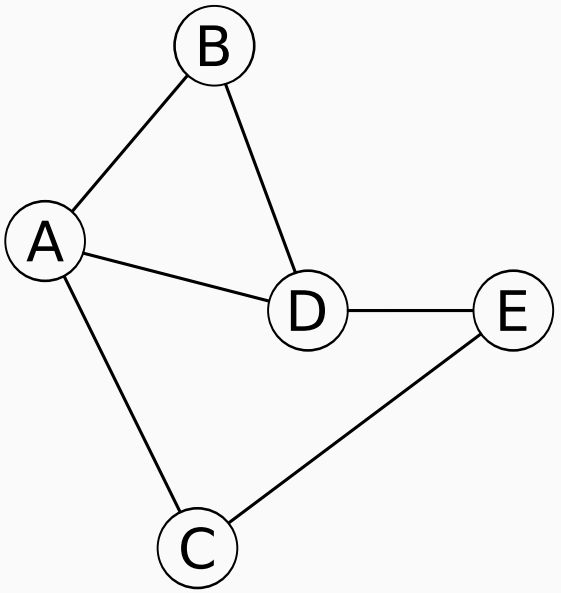
The first component of a BN is a graph. A graph $\mathcal{G}$ is a mathematical object with:

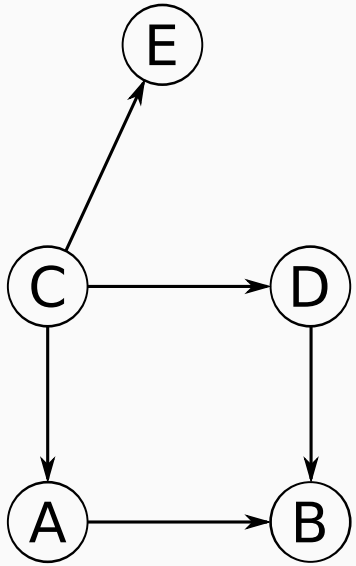
- a set of nodes;
- a set of arcs A which are identified by pairs for nodes.

Given the nodes, a graph is uniquely identified by the arc set. An arc can be:

- undirected if the arc has no direction, for instance $A - B$;
- directed if the arc has a specific direction, for instance $A \rightarrow B$.

The assumption is that there is at most one arc between each pair of nodes.

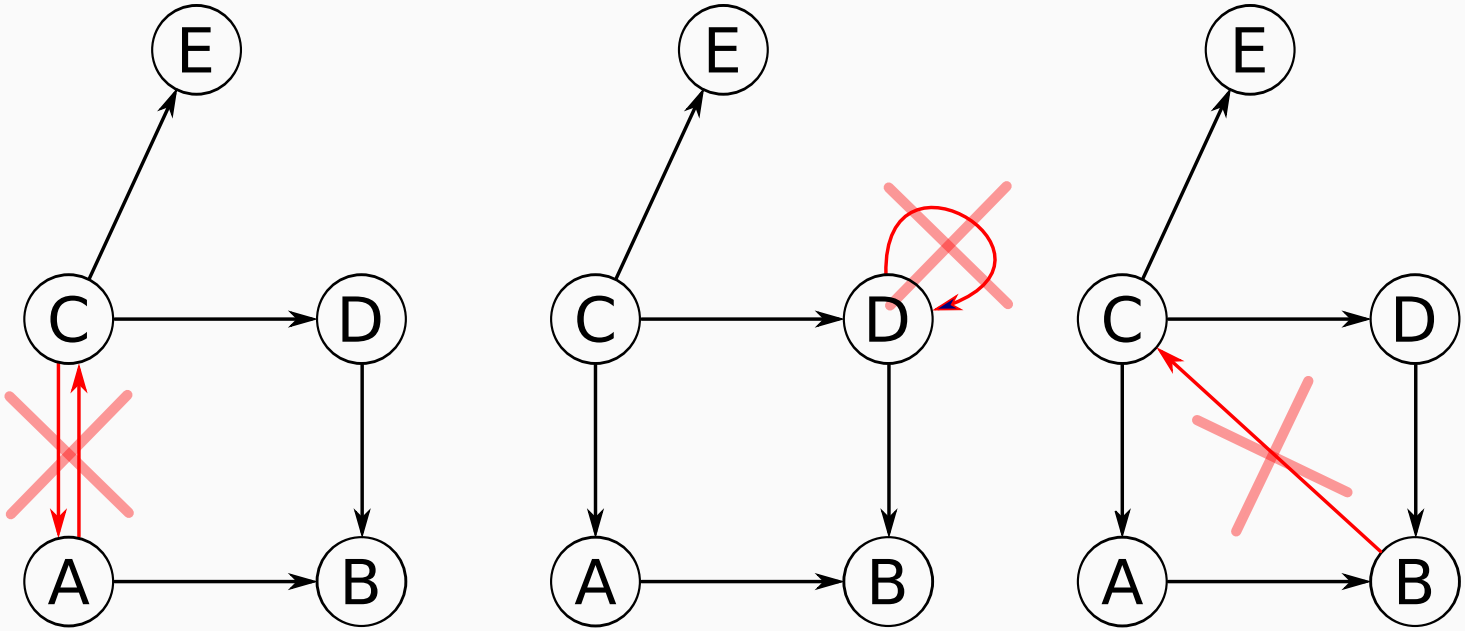




Directed Acyclic Graphs

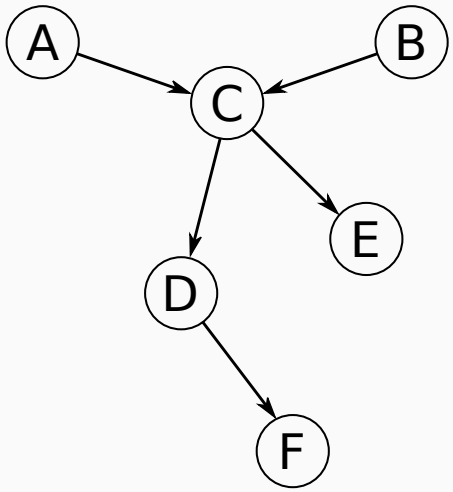
BNs use a specific kind of graph called a directed acyclic graph (DAG), that:

- contains only directed arcs;
- does not contain any loop (an arc $D \rightarrow D$ from a node to itself);
- does not contain any cycle (a sequence of arcs like $B \rightarrow C \rightarrow D \rightarrow B$ that starts and ends in the same node).



How the DAG Maps to the Probability Distribution

DAG



$\Longrightarrow$

Graphical separation

$$
\begin{aligned}
&A \perp\!\!\!\perp_G B \\
&A \perp\!\!\!\perp_G D \mid C \\
&B \perp\!\!\!\perp_G D \mid C \\
&A \perp\!\!\!\perp_G E \mid C \\
&B \perp\!\!\!\perp_G E \mid C \\
&D \perp\!\!\!\perp_G E \mid C \\
&C \perp\!\!\!\perp_G F \mid D \\
&\ldots
\end{aligned}
$$

$\Longrightarrow$

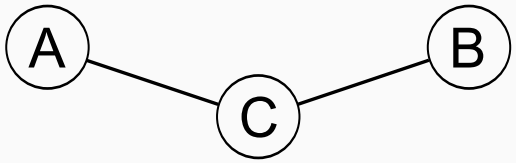
Probabilistic independence

$$
\begin{aligned}
&A \perp\!\!\!\perp_P B \\
&A \perp\!\!\!\perp_P D \mid C \\
&B \perp\!\!\!\perp_P D \mid C \\
&A \perp\!\!\!\perp_P E \mid C \\
&B \perp\!\!\!\perp_P E \mid C \\
&D \perp\!\!\!\perp_P E \mid C \\
&C \perp\!\!\!\perp_P F \mid D \\
&\ldots
\end{aligned}
$$

Formally, the DAG is an independence map of the probability distribution of $\mathbf{X}$, with graphical separation ($\perp\!\!\!\perp_G$) implying probabilistic independence ($\perp\!\!\!\perp_P$).

Graphical Separation in DAGs: Fundamental Connections

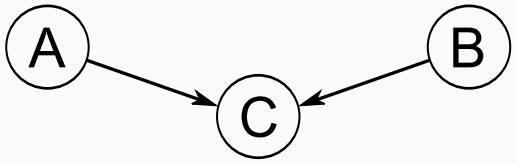
separation (undirected graphs)



$$\mathbf{A} \perp\!\!\!\perp \mathbf{B} \,|\, \mathbf{C}$$

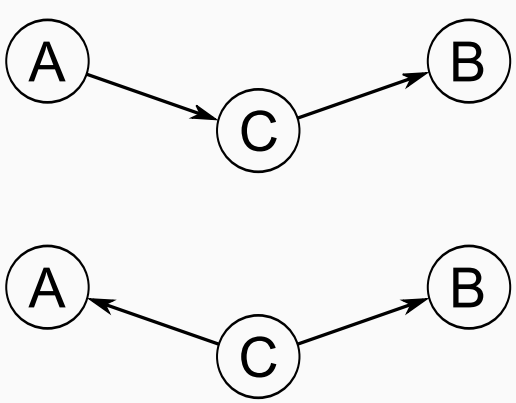
$$\mathrm{P}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \mathrm{P}(\mathbf{A} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{B} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{C})$$

d-separation (directed acyclic graphs)



$$\mathbf{A} \not\perp\!\!\!\perp \mathbf{B} \,|\, \mathbf{C}$$

$$\mathrm{P}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \mathrm{P}(\mathbf{C} \,|\, \mathbf{A}, \mathbf{B})\, \mathrm{P}(\mathbf{A})\, \mathrm{P}(\mathbf{B})$$



$$\mathbf{A} \perp\!\!\!\perp \mathbf{B} \,|\, \mathbf{C}$$

$$\begin{aligned} \mathrm{P}(\mathbf{A}, \mathbf{B}, \mathbf{C}) &= \\ &= \mathrm{P}(\mathbf{B} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{C} \,|\, \mathbf{A})\, \mathrm{P}(\mathbf{A}) \\ &= \mathrm{P}(\mathbf{A} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{B} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{C}) \end{aligned}$$

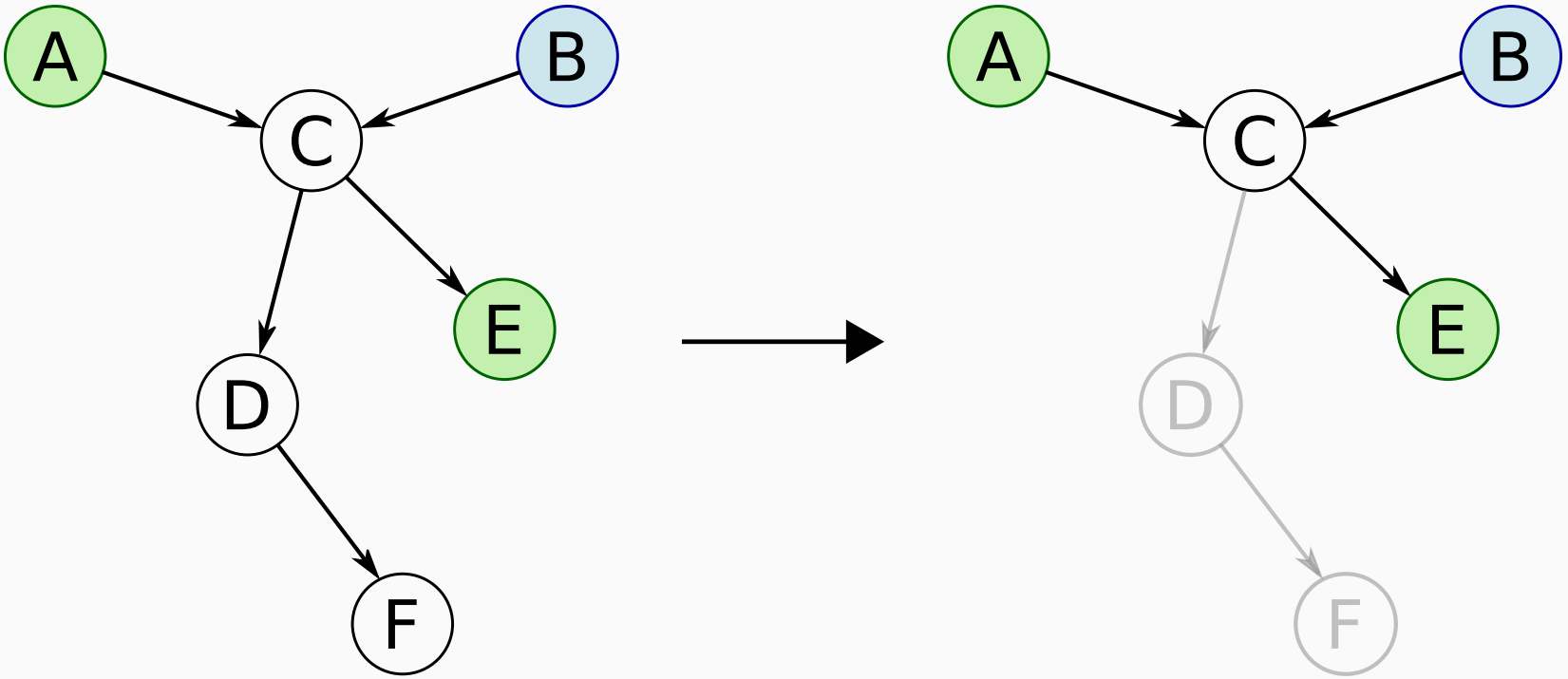
Graphical Separation in DAGs: General Case

Now, in the general case we can extend the patterns from the fundamental connections and apply them to every possible path between $\mathbf{A}$ and $\mathbf{B}$ for a given $\mathbf{C}$; this is how d-separation is defined.

> *If $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are three disjoint subsets of nodes in a directed acyclic graph $\mathcal{G}$, then $\mathbf{C}$ is said to d-separate $\mathbf{A}$ from $\mathbf{B}$, denoted $\mathbf{A} \perp\!\!\!\perp_G \mathbf{B} \mid \mathbf{C}$, if along every path between a node in $\mathbf{A}$ and a node in $\mathbf{B}$ there is a node v satisfying one of the following two conditions:*
>
> 1. *v has converging edges (that is, there are two edges pointing to v from the adjacent nodes in the path) and none of v or its descendants (that is, the nodes that can be reached from v) are in $\mathbf{C}$.*
> 2. *v is in $\mathbf{C}$ and does not have converging edges.*

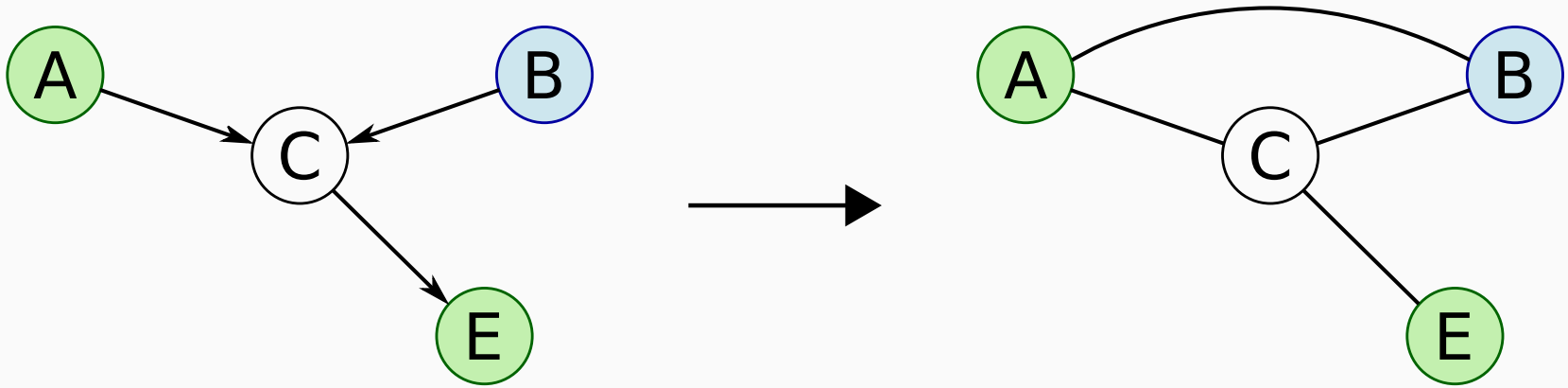
This definition clearly does not provide a computationally feasible approach to assess d-separation; but there are other ways.

A Simple Algorithm to Check D-Separation



Say that we want to check whether A and E are d-separated by B. First, we can drop all the nodes that are not ancestors (that is, parents, parents' parents, etc.) of A, E and B since each node only depends on its parents.

A Simple Algorithm to Check D-Separation

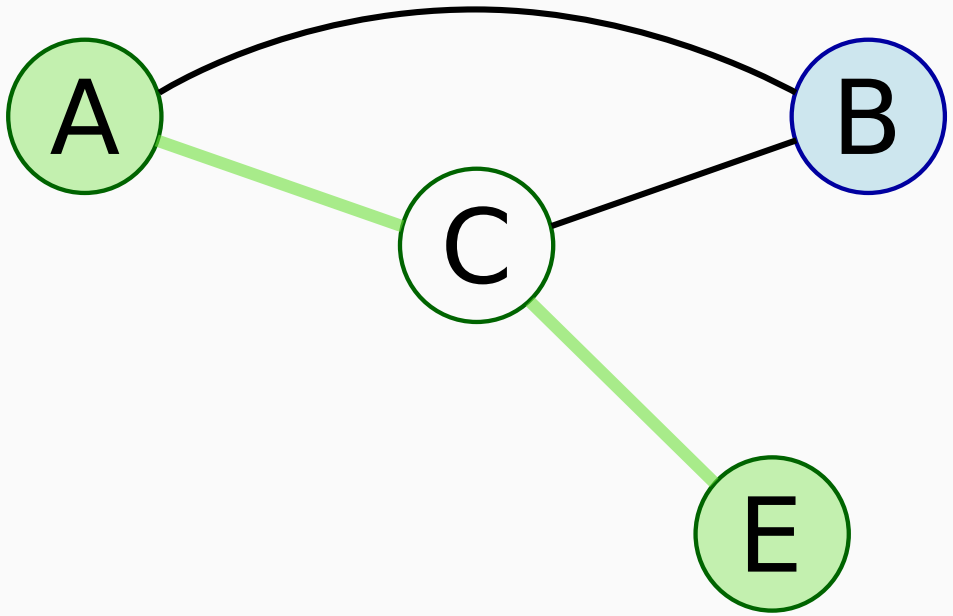


We then transform the subgraph into its moral graph by

1. connecting all the nodes that have one child in common; and
2. removing all arc directions to obtain an undirected graph.

This transformation makes the dependence between parents explicit by "marrying" them and of makes it possible for us to use the classic definition of graphical separation.

A Simple Algorithm to Check D-Separation



Finally, we can just perform a depth-first or breadth-first search and see if we can find an open path between A and E, that is, a path that is not blocked by B.

D-Separation Example: The DAG We Created Earlier

The last graph is an undirected graph: if there is a path from A to E there is a path from E to A. This means that d-separation is symmetric:

$$\mathsf{A} \not\perp\!\!\!\perp_G \mathsf{E} \mid \mathsf{B} \iff \mathsf{E} \not\perp\!\!\!\perp_G \mathsf{A} \mid \mathsf{B}$$

Which must be the case because independence is also symmetric,

$$\mathrm{P}(\mathsf{A}, \mathsf{E} \mid \mathsf{B}) = \mathrm{P}(\mathsf{E}, \mathsf{A} \mid \mathsf{B}) \neq \mathrm{P}(\mathsf{A} \mid \mathsf{B})\,\mathrm{P}(\mathsf{E} \mid \mathsf{B}),$$

and d-separation implies probabilistic independence.

NOTE: d-separation does not necessarily require a separating set. Or, to put it in another way, the separating set can be empty. In that case we are checking whether variables are marginally independent because there is no path at all that connects them.

If we use d-separation as our definition of graphical separation, assuming that the DAG is an independence map leads to the general formulation of the decomposition of the global distribution

$$\mathrm{P}(\mathbf{X}) = \prod_{i=1}^{N} \mathrm{P}(X_i \mid \Pi_{X_i})$$

into the local distributions for the X_i given their parents Π_{X_i}. If X_i has two or more parents it depends on their joint distribution, because each pair of parents forms a convergent connection centred on X_i and we cannot establish their independence. This decomposition is preferable to that obtained from the chain rule,

$$\mathrm{P}(\mathbf{X}) = \prod_{i=1}^{N} \mathrm{P}(X_i \mid X_{i+1}, \ldots, X_N),$$

because the conditioning sets are typically smaller.

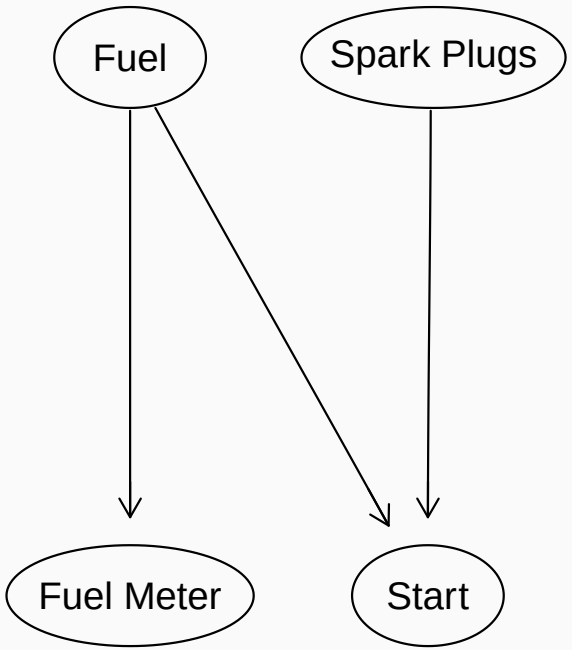
The Local Markov Property

Another result along the same lines is the local Markov property, which can be combined with the chain rule above to get the decomposition into local distributions.

> *Each node X_i is conditionally independent of its non-descendants (the nodes X_j for which there is no path from X_i to X_j) given its parents.*

Compared to the previous decomposition, it highlights the fact that parents are not completely independent from their children in the BN: a trivial application of Bayes' theorem to invert the direction of the conditioning shows how information on a child can change the distribution of the parent.

The Local Markov Property: Car Start



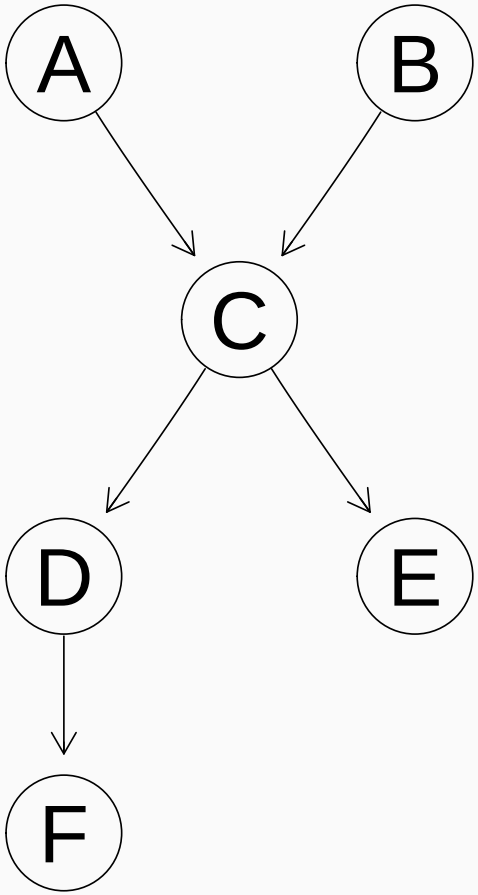
The parent sets:

$$\texttt{Fuel} = \{\}$$
$$\texttt{Fuel Meter} = \{\texttt{Fuel}\}$$
$$\texttt{Spark Plugs} = \{\}$$
$$\texttt{Start} = \{\texttt{Fuel}, \texttt{Spark Plugs}\}$$

The corresponding decomposition:

$$\begin{aligned}\mathrm{P}(&\texttt{Start}, \texttt{Fuel Meter}, \texttt{Fuel},\\ &\texttt{Spark Plugs}) =\\ &\mathrm{P}(\texttt{Start} \mid \texttt{Fuel}, \texttt{Spark Plugs})\\ &\mathrm{P}(\texttt{Fuel Meter} \mid \texttt{Fuel}) \times\\ &\mathrm{P}(\texttt{Fuel})\,\mathrm{P}(\texttt{Spark Plugs})\end{aligned}$$

The Local Markov Property: The DAG We Created Earlier



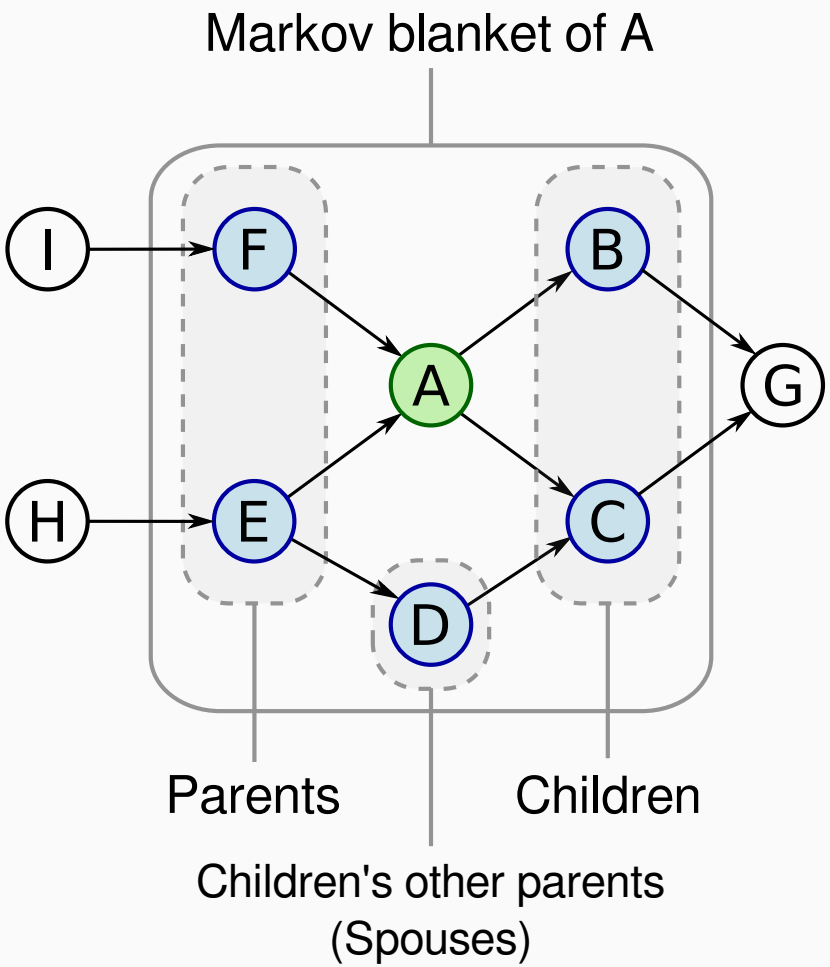
The parent sets:

$$\mathtt{A} = \{\}$$
$$\mathtt{B} = \{\}$$
$$\mathtt{C} = \{\mathtt{A}, \mathtt{B}\}$$
$$\mathtt{D} = \{\mathtt{C}\}$$
$$\mathtt{E} = \{\mathtt{C}\}$$
$$\mathtt{F} = \{\mathtt{D}\}$$

The corresponding decomposition:

$$\mathrm{P}(\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}) = \mathrm{P}(\mathtt{A})\,\mathrm{P}(\mathtt{B})\,\mathrm{P}(\mathtt{C} \mid \mathtt{A}, \mathtt{B})\,\mathrm{P}(\mathtt{D} \mid \mathtt{C})\,\mathrm{P}(\mathtt{E} \mid \mathtt{C})\,\mathrm{P}(\mathtt{F} \mid \mathtt{D})$$

Completely D-Separating: Markov Blankets

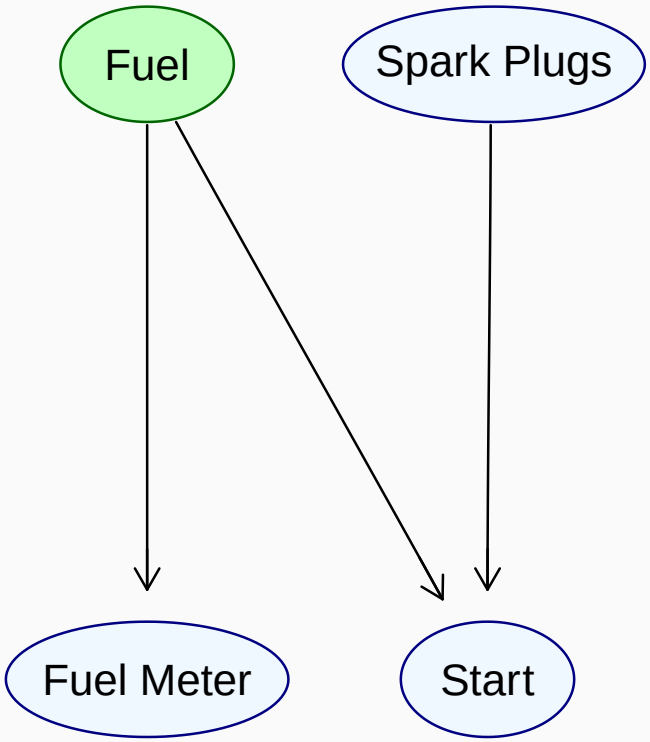


We can easily use the DAG to solve the feature selection problem. The set of nodes that graphically isolates a target node from the rest of the DAG is called its Markov blanket and includes:

- its parents;
- its children;
- other nodes sharing a child.

Since $\perp\!\!\!\perp_G$ implies $\perp\!\!\!\perp_P$, we can restrict ourselves to the Markov blanket to perform any kind of inference on the target node, and disregard the rest.

Markov Blanket: Car Start



The parents, children and spouses of `Fuel`:

$\{\}$

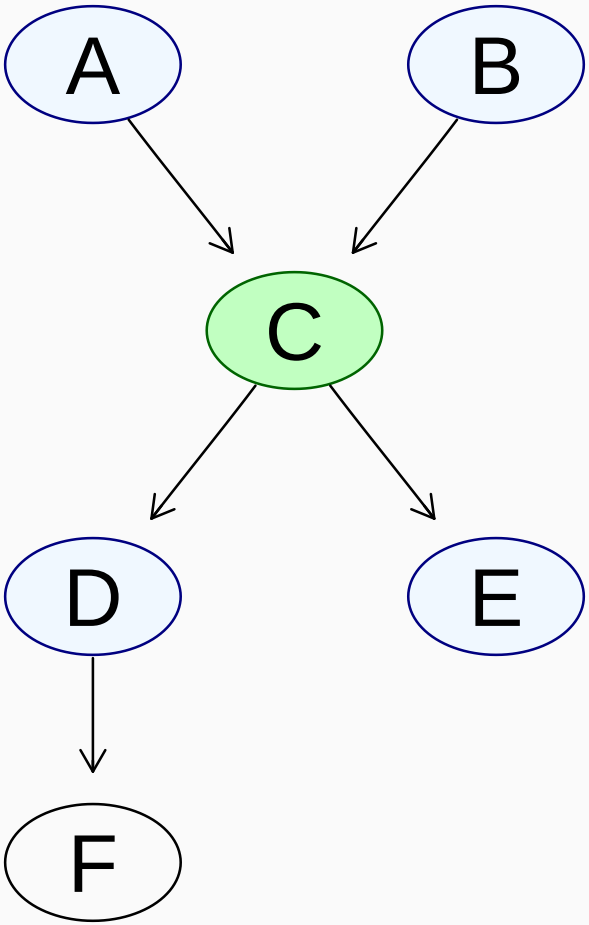
$\{\texttt{Fuel Meter}, \texttt{Start}\}$

$\{\texttt{Spark Plugs}\}$

The Markov blanket of `Fuel`:

$\{\texttt{Fuel Meter}, \texttt{Spark Plugs}, \texttt{Start}\}$

Markov Blanket: The DAG we created earlier



Printing the parents, children and spouses of C:

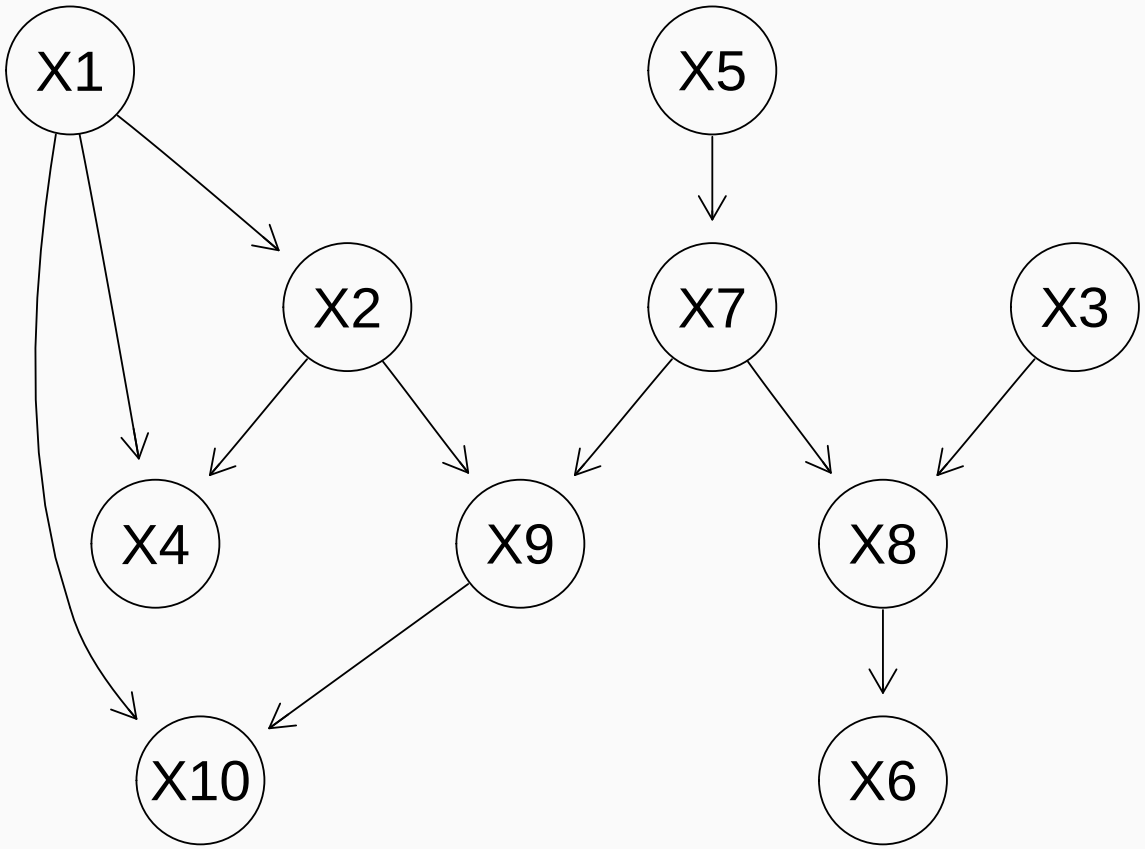
{A, B}
{D, E}
{}

The Markov blanket of C:

{A, B, D, E}

Different DAGs, Same Distribution

A DAG uniquely identifies a factorisation of $\mathrm{P}(\mathbf{X})$; the converse is not necessarily true. Consider this DAG:



Different DAGs, Same Distribution

The decomposition into local distributions is:

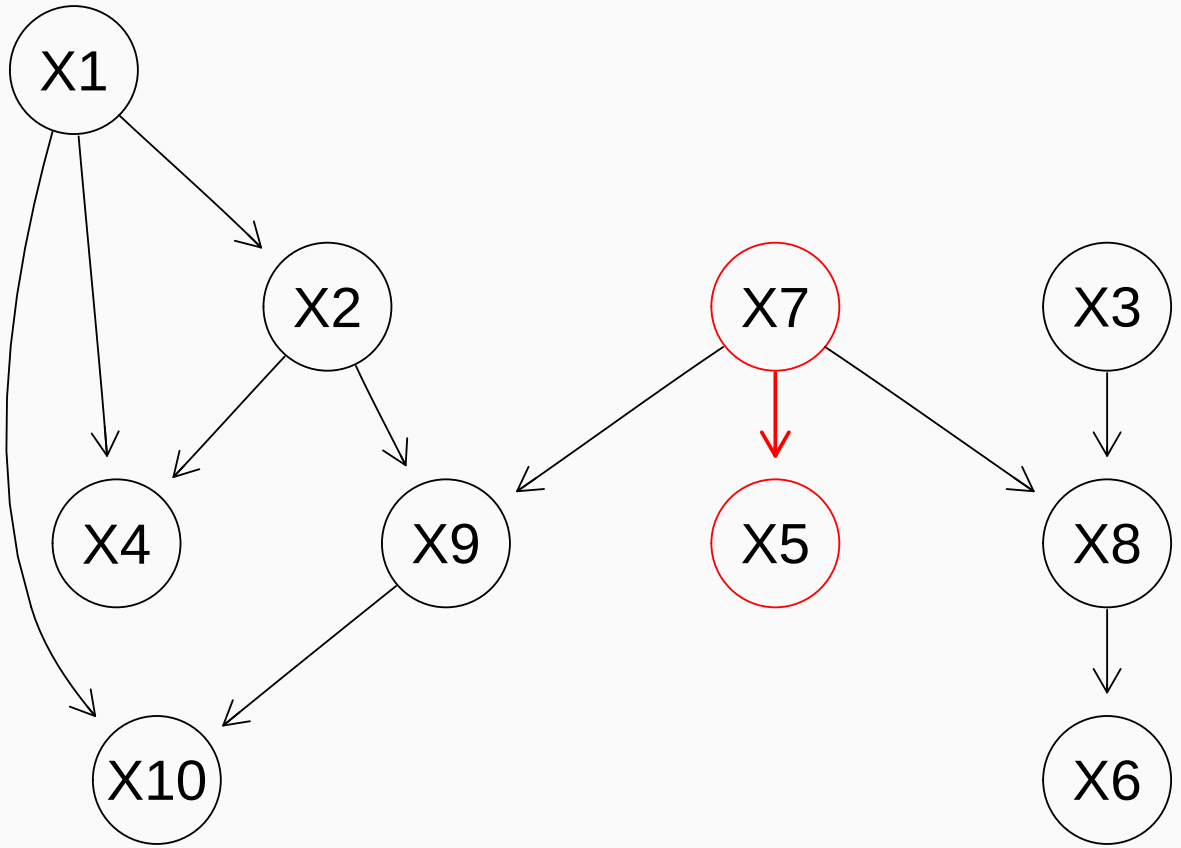
$$\begin{aligned}\mathrm{P}(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}) = \\ \mathrm{P}(X_1)\,\mathrm{P}(X_3)\,\underbrace{\mathrm{P}(X_5)}_{X_5}\,\mathrm{P}(X_6 \mid X_8)\,\mathrm{P}(X_2 \mid X_1)\,\underbrace{\mathrm{P}(X_7 \mid X_5)}_{X_5 \to X_7} \\ \mathrm{P}(X_4 \mid X_1, X_2)\,\mathrm{P}(X_8 \mid X_3, X_7)\,\mathrm{P}(X_9 \mid X_2, X_7)\,\mathrm{P}(X_{10} \mid X_1, X_9).\end{aligned}$$

However, look at $X_5 \to X_7$: $\mathrm{P}(X_7 \mid X_5)\,\mathrm{P}(X_5) = \mathrm{P}(X_5 \mid X_7)\,\mathrm{P}(X_7)$ by Bayes' theorem. Then

$$\begin{aligned}\mathrm{P}(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}) = \\ \mathrm{P}(X_1)\,\mathrm{P}(X_3)\,\underbrace{\mathrm{P}(X_7)}_{X_7}\,\mathrm{P}(X_6 \mid X_8)\,\mathrm{P}(X_2 \mid X_1)\,\underbrace{\mathrm{P}(X_5 \mid X_7)}_{X_7 \to X_5} \\ \mathrm{P}(X_4 \mid X_1, X_2)\,\mathrm{P}(X_8 \mid X_3, X_7)\,\mathrm{P}(X_9 \mid X_2, X_7)\,\mathrm{P}(X_{10} \mid X_1, X_9).\end{aligned}$$

Different DAGs, Same Distribution

The DAG that gives this new, equivalent decomposition is:



Different DAGs, Same Distribution

Next let's look at $X_8 \to X_6$.

$$\begin{aligned}\mathrm{P}(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}) = \\ \mathrm{P}(X_1)\,\mathrm{P}(X_3)\,\mathrm{P}(X_7)\,\underbrace{\mathrm{P}(X_6 \mid X_8)}_{X_8 \to X_6}\,\mathrm{P}(X_2 \mid X_1)\,\mathrm{P}(X_5 \mid X_7) \\ \mathrm{P}(X_4 \mid X_1, X_2)\,\underbrace{\mathrm{P}(X_8 \mid X_3, X_7)}_{X_8 \leftarrow X_3, X_8 \leftarrow X_7}\,\mathrm{P}(X_9 \mid X_2, X_7)\,\mathrm{P}(X_{10} \mid X_1, X_9).\end{aligned}$$

We cannot reverse the $X_8 \to X_6$ as we did with $X_5 \to X_7$ without changing the probability distribution. If we try, we get

$$\mathrm{P}(X_6 \mid X_8)\,\mathrm{P}(X_8 \mid X_3, X_7) = \mathrm{P}(X_8 \mid X_6)\,\mathrm{P}(X_6)\frac{\mathrm{P}(X_8 \mid X_3, X_7)}{\mathrm{P}(X_8)},$$

which does not simplify because X_8 has other parents (X_3, X_7).

Different DAGs, Same Distribution

Finally, let's look at X_1, X_2 and X_4.

$$\begin{aligned}\mathrm{P}(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}) = \\ \underbrace{\mathrm{P}(X_1)}_{X_1} \mathrm{P}(X_3)\,\mathrm{P}(X_5)\,\mathrm{P}(X_6 \mid X_8)\,\underbrace{\mathrm{P}(X_2 \mid X_1)}_{X_1 \to X_2}\,\mathrm{P}(X_7 \mid X_5) \\ \underbrace{\mathrm{P}(X_4 \mid X_1, X_2)}_{X_1 \to X_4, X_2 \to X4}\,\mathrm{P}(X_8 \mid X_3, X_7)\,\mathrm{P}(X_9 \mid X_2, X_7)\,\mathrm{P}(X_{10} \mid X_1, X_9).\end{aligned}$$

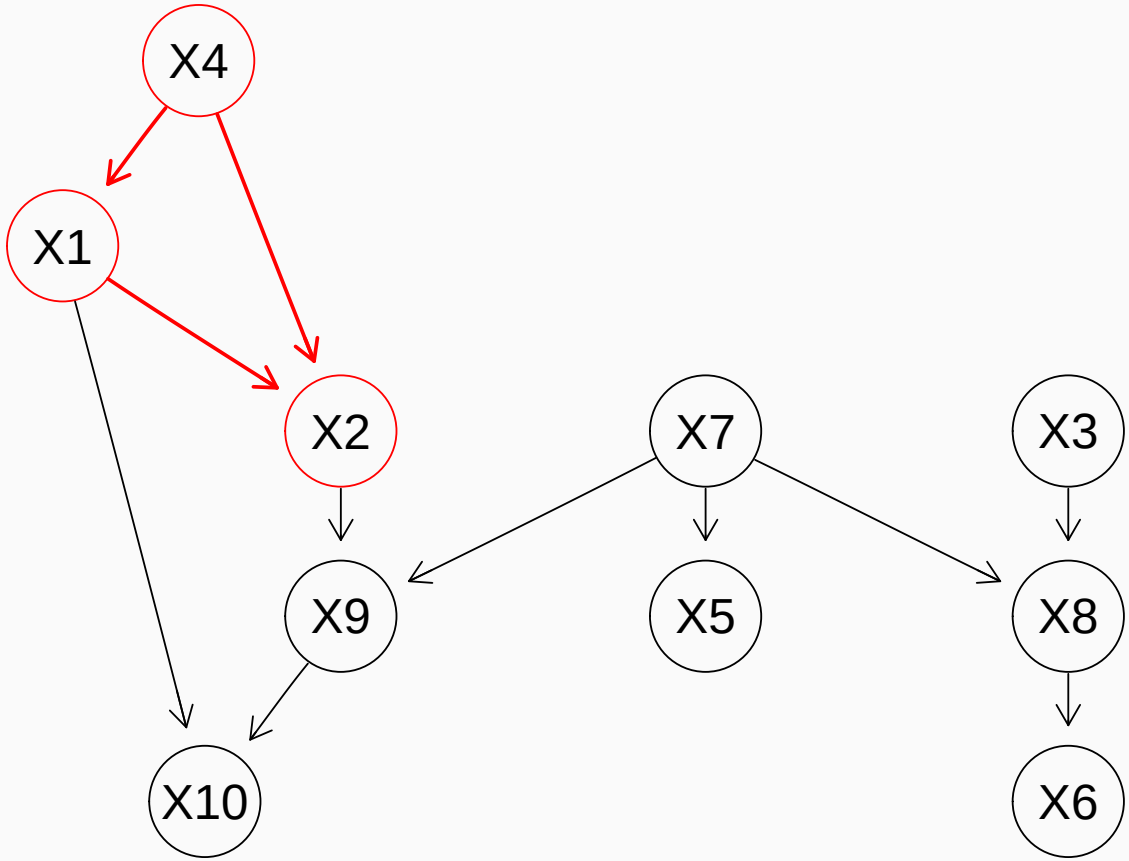
By Bayes' theorem we can say

$$\begin{aligned}\mathrm{P}(X_1)\,\mathrm{P}(X_2 \mid X_1)\,\mathrm{P}(X_4 \mid X_1, X_2) = \mathrm{P}(X_1, X_2, X_4) = \\ \underbrace{\mathrm{P}(X_2)}_{X_2}\,\underbrace{\mathrm{P}(X_2 \mid X_4)}_{X_4 \to X_2}\,\underbrace{\mathrm{P}(X_1 \mid X_2, X_4)}_{X_1 \leftarrow X_2, X_1 \leftarrow X_4}\end{aligned}$$

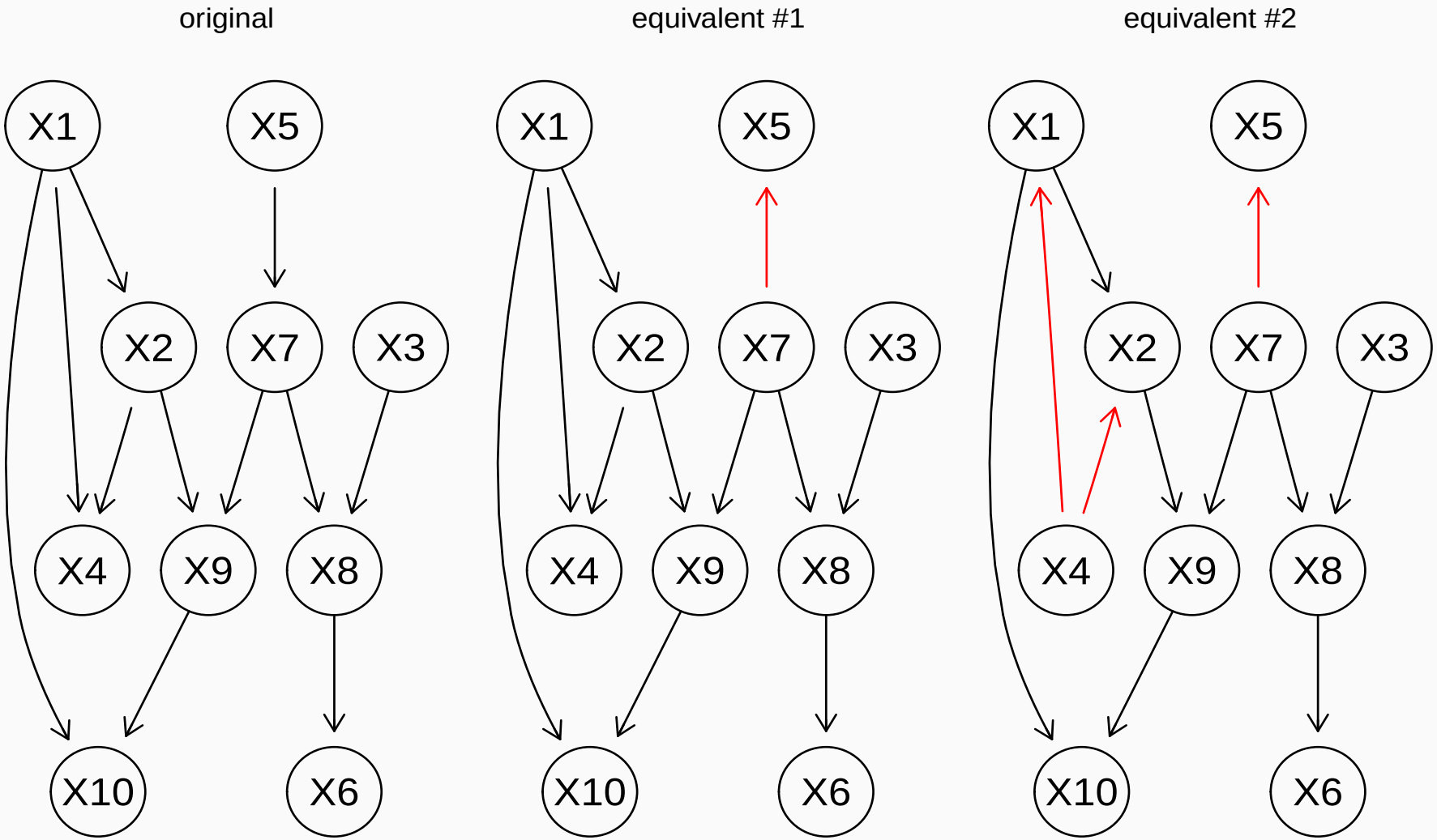
which gives us another DAG again.

Different DAGs, Same Distribution

The DAG that gives this last equivalent decomposition is:



Comparing These Different DAGs



Different DAGs, Same Distribution: Equivalence Classes

To sum it up: we can reverse a number of arcs without changing the dependence structure of $\mathbf{X}$. Since the fundamental connections $A \to C \to B$ and $A \leftarrow C \to B$ are probabilistically equivalent, we can reverse the directions of their arcs as we like as long as we do not create any new v-structure ($A \to C \leftarrow B$, with no arc between A and B).

This means that we can group DAGs into equivalence classes that are uniquely identified by the underlying undirected graph and the v-structures. The directions of other arcs can be either:

- uniquely identifiable because one of the directions would introduce cycles or new v-structures (compelled arcs);
- completely undetermined.

The result is a completed partially directed graph (CPDAG).

It is important to note that even though $A \to C \leftarrow B$ is a convergent connection, it is not a v-structure if A and C are connected by $A \to B$ or $B \to A$. In that case, we are no longer able to identify which nodes are the parents in the connection.

For instance:

$$\underbrace{\mathrm{P}(A)\,\mathrm{P}(B \mid A)\,\mathrm{P}(C \mid A, B)}_{A \to C \leftarrow B,\, A \to B} = \mathrm{P}(A)\,\frac{\mathrm{P}(B, A)}{\mathrm{P}(A)}\,\frac{\mathrm{P}(C, A, B)}{\mathrm{P}(A, B)} =$$

$$= \mathrm{P}(A)\,\mathrm{P}(C, B \mid A) = \underbrace{\mathrm{P}(A)\,\mathrm{P}(B \mid C, A)\,\mathrm{P}(C \mid A)}_{C \to B \leftarrow A,\, A \to C}.$$

Therefore, the fact that the two parents in a v-structure are not connected is crucial in the identification of the correct CPDAG.

Our DAG

From this description we can tell different groups of arcs apart:

Directed arcs:

$$
\begin{aligned}
X_8 &\to X_6 \\
X_1 &\to X_2 \\
X_5 &\to X_7 \\
X_1 &\to X_4 \\
X_2 &\to X_4 \\
X_3 &\to X_8 \\
X_7 &\to X_8 \\
X_2 &\to X_9 \\
X_7 &\to X_9 \\
X_1 &\to X_{10} \\
X_9 &\to X_{10}
\end{aligned}
$$

Undirected arcs:

None.

Compelled arcs:

$$
\begin{aligned}
X_1 &\to X_{10} \\
X_2 &\to X_9 \\
X_3 &\to X_8 \\
X_7 &\to X_8 \\
X_7 &\to X_9 \\
X_8 &\to X_6 \\
X_9 &\to X_{10}
\end{aligned}
$$

V-structures:

$$
\begin{aligned}
X_1 &\to X_{10} \leftarrow X_9 \\
X_3 &\to X_8 \leftarrow X_7 \\
X_2 &\to X_9 \leftarrow X_7
\end{aligned}
$$

The Corresponding CPDAG

Which in the corresponding CPDAG become:

Directed arcs:

$$
\begin{array}{lcl}
X_1 & \rightarrow & X_{10} \\
X_2 & \rightarrow & X_9 \\
X_3 & \rightarrow & X_8 \\
X_7 & \rightarrow & X_8 \\
X_7 & \rightarrow & X_9 \\
X_8 & \rightarrow & X_6 \\
X_9 & \rightarrow & X_{10}
\end{array}
$$

Undirected arcs:

$$
\begin{array}{lcl}
X_1 & - & X_2 \\
X_1 & - & X_4 \\
X_2 & - & X_1 \\
X_2 & - & X_4 \\
X_4 & - & X_1 \\
X_4 & - & X_2 \\
X_5 & - & X_7 \\
X_7 & - & X_5
\end{array}
$$

Compelled arcs:

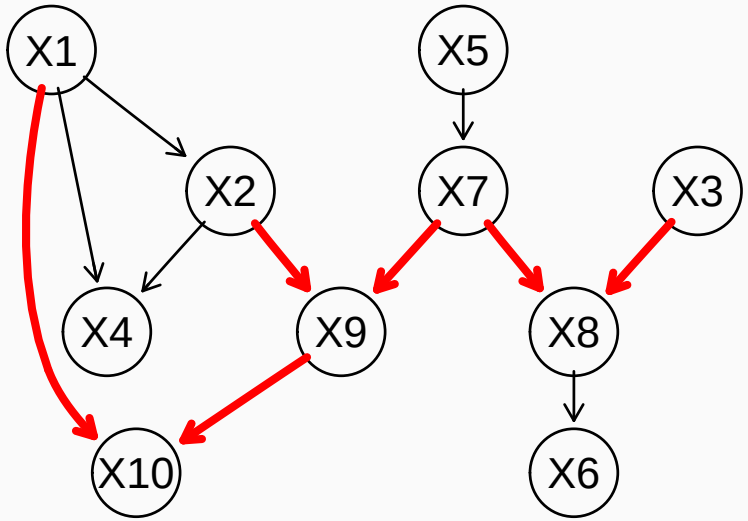
$$
\begin{array}{lcl}
X_1 & \rightarrow & X_{10} \\
X_2 & \rightarrow & X_9 \\
X_3 & \rightarrow & X_8 \\
X_7 & \rightarrow & X_8 \\
X_7 & \rightarrow & X_9 \\
X_8 & \rightarrow & X_6 \\
X_9 & \rightarrow & X_{10}
\end{array}
$$

V-structures:

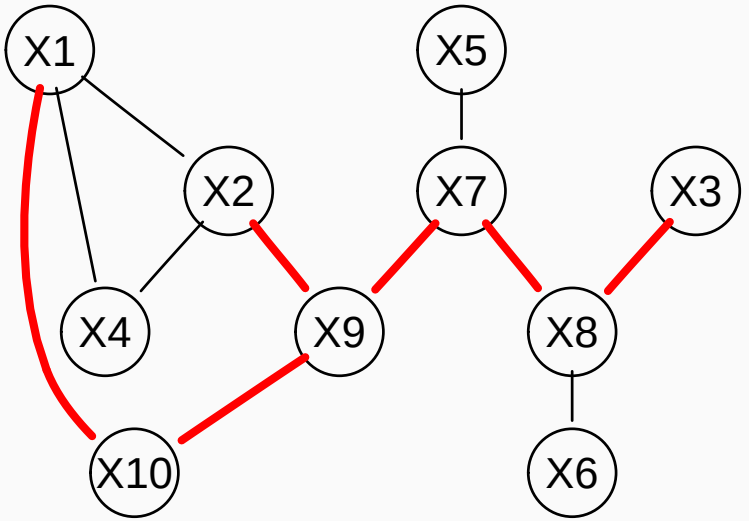
$$
\begin{array}{lcllcl}
X_1 & \rightarrow & X_{10} & \leftarrow & X_9 \\
X_3 & \rightarrow & X_8 & \leftarrow & X_7 \\
X_2 & \rightarrow & X_9 & \leftarrow & X_7
\end{array}
$$

DAG, CPDAG and Equivalent DAGs

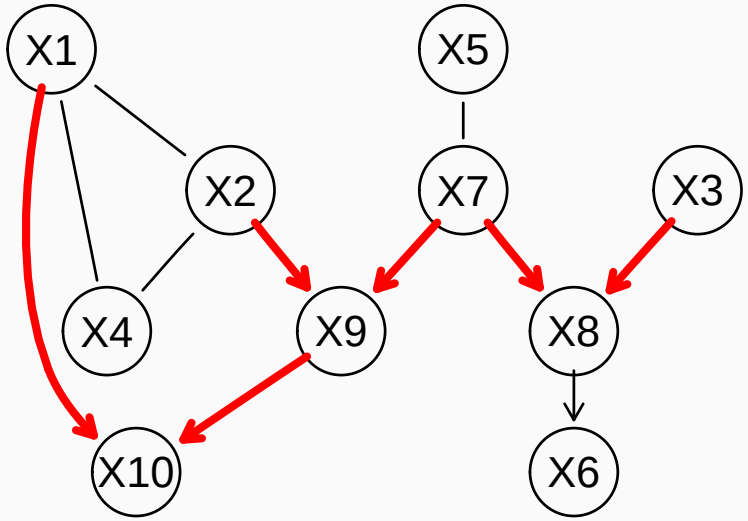
DAG



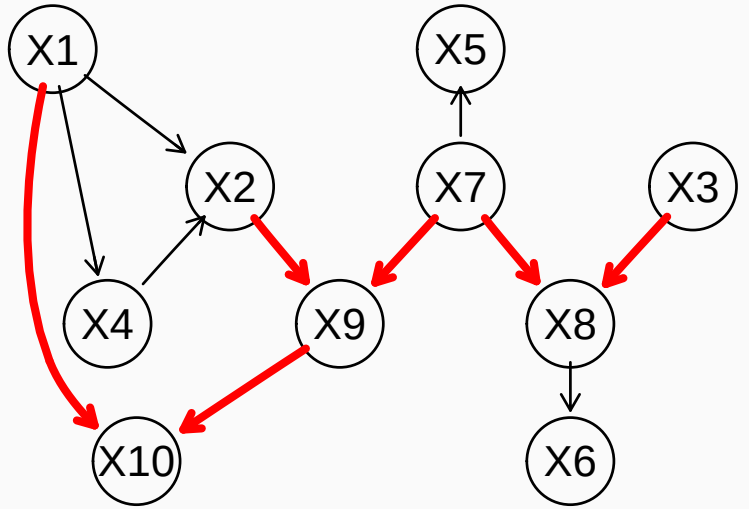
Skeleton



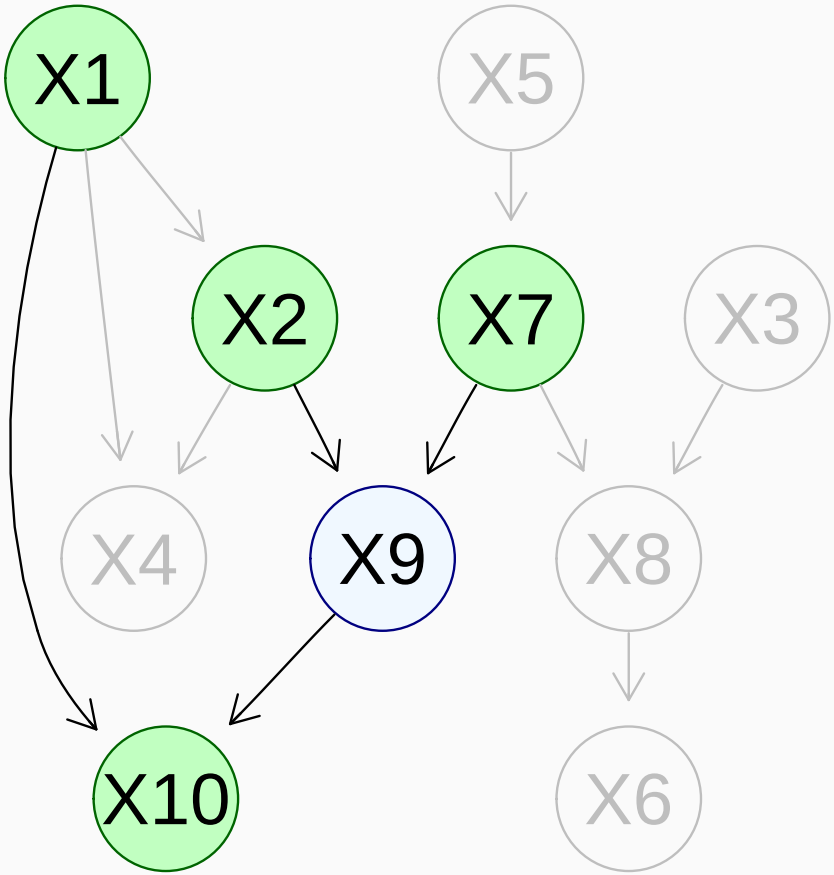
CPDAG

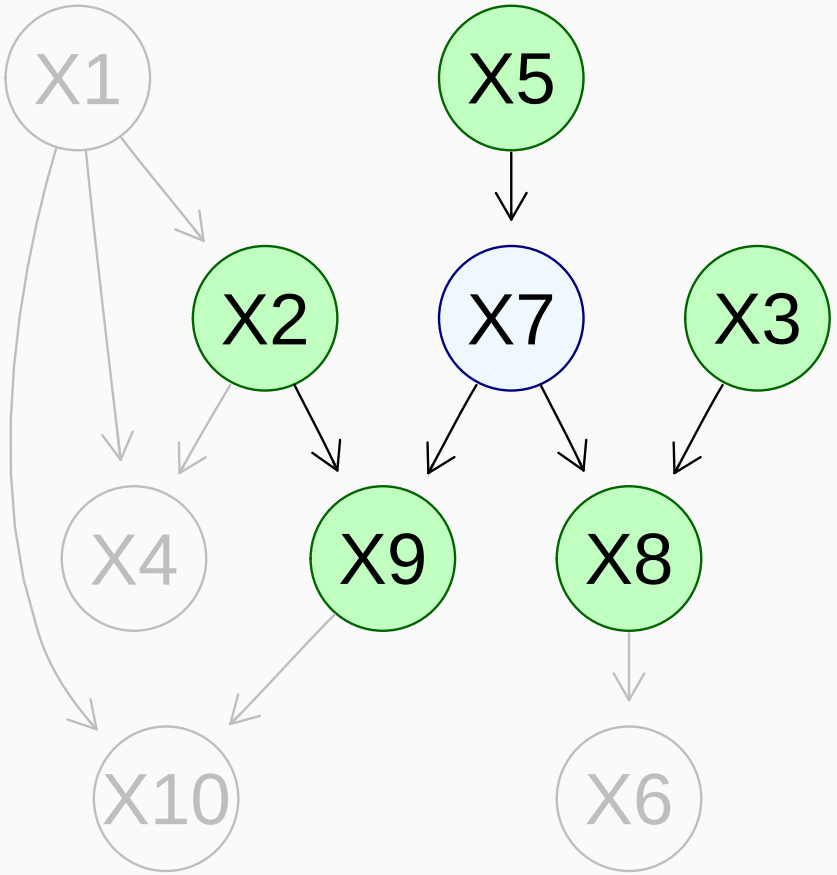


An Equivalent DAG



Two More Examples of Markov Blankets





Markov Blankets are Symmetric

We can also check that Markov blankets are symmetric: if A is in the Markov blanket of B, then B is in the Markov blanket of A.

In which Markov blankets is X_9 in?

X1	X10	X2	X3	X4	X5	X6	X7	X8	X9
TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE

Which nodes are in the Markov blanket of X_9?

X1	X10	X2	X3	X4	X5	X6	X7	X8	X9
TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE

This is a consequence of the fact that if A is a parent of B, then B is a child of A; and if A is a spouse of B, then B is a spouse of A.

Relevant Functions in bnlearn

- creating DAGs: `empty.graph()`, `set.arc()`, `drop.arc()`, `reverse.arc()`.
- model string representations: `modelstring()`, `model2network()`.
- nodes in a DAG: `nodes()`, `parents()`, `children()`, `spouses()`, `nbr()`, `mb()`.
- arcs in a DAG: `arcs()`, `path.exists()`, `dsep()`, `directed.arcs()`, `undirected.arcs()`, `compelled.arcs()`.
- DAG transformation: `subgraph()`, `moral()`, `cpdag()`.
- plotting: `graphviz.plot()`, `graphviz.compare()`.

Summary and Remarks

- BNs are one of the oldest instances of machine learning models.
- BNs are a probabilistic model that use DAGs to make computations systematic in a rigorous way.
- BNs allow computer systems to perform automatically all the computations we did by hand at the beginning of this lecture.
- At the same time, BNs using DAGs means that they provide a qualitative, intuitive way to reason about complex phenomena.

Next:

- What probability distributions do we use to construct a BN?

Thanks!

Any questions?

Bayesian Networks in Policy and Society

Day 2: Models and Distributional Assumptions

Marco Scutari

Dalle Molle Institute for
Artificial Intelligence (IDSIA)

August 9, 2022

What About the Probability Distributions?

The second component of a BN is the probability distribution $\mathrm{P}(\mathbf{X})$. The choice should be such that the BN:

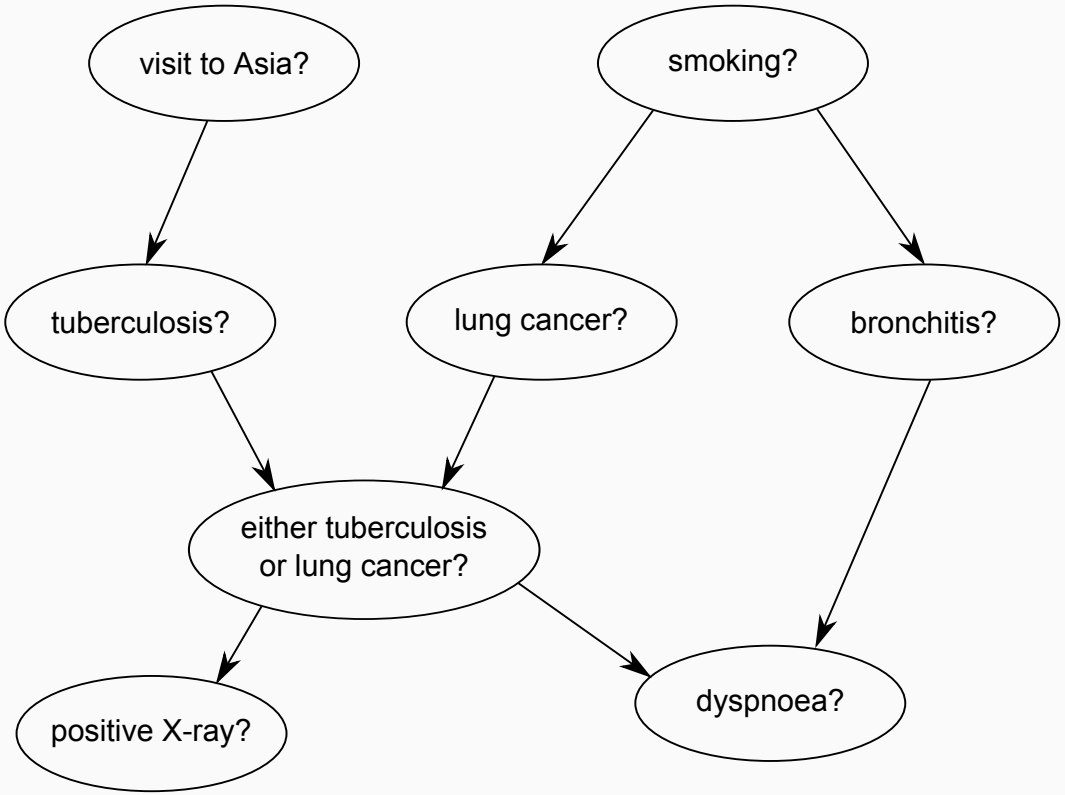
- can be learned efficiently from data;
- is flexible (distributional assumptions should not be too limiting);
- is easy to query to perform inference.

The three most common choices in the literature (by far), are:

- discrete BNs (DBNs), in which $\mathbf{X}$ and the $X_i \mid \Pi_{X_i}$ are multinomial;
- Gaussian BNs (GBNs), in which $\mathbf{X}$ is multivariate normal and the $X_i \mid \Pi_{X_i}$ are univariate normal;
- conditional linear Gaussian BNs (CGBNs), in which X is a mixture of multivariate normals and the $X_i \mid \Pi_{X_i}$ are either multinomial, univariate normal or mixtures of normals.

It has been proved in the literature that exact inference is possible in these three cases, hence their popularity.

Discrete BNs



A classic example of DBN is the ASIA network from Lauritzen & Spiegelhalter (1988), which includes a collection of binary variables. It describes a simple diagnostic problem for tuberculosis and lung cancer.

Total parameters of $\mathbf{X}$:
$2^8 - 1 = 255$

The Global Distribution

The global distribution $\mathrm{P}(\mathbf{X})$ is

$$\mathbf{X} \sim Mul\left(\pi_{\text{<insert one subscript for each node here>}}\right).$$

The parameters $\pi_{\text{<insert one subscript for each node here>}}$ describe the probability of each possible combination of the values of the variables in $\mathbf{X}$.

Problems:

- There are too many parameters: if each variable can take l values $|\Theta| = |\mathbf{X}|^l$ - 1! It is difficult to even tabulate them, much less give them a practical interpretation.
- They are too small: $\approx 1/(|\mathbf{X}|^l)$ on average since they sum up to 1.
- They have no structure: they do not separate main effects, low-order interactions and high-order interactions.

The Local Distributions

The definition of a BN states that $\mathrm{P}(\mathbf{X}) = \prod_{i=1}^{N} \mathrm{P}(X_i \,|\, \Pi_{X_i})$.
Let's assume:

1. Positivity: all probabilities are strictly positive.
2. Parameter independence: probabilities in each $X_i \,|\, \Pi_{X_i}$ that are conditional on different parent configurations are independent.
3. Parameter modularity: probabilities in different $X_i \,|\, \Pi_{X_i}$ are independent.

Then $X_i \,|\, \Pi_{X_i} \sim Mul\,(\pi_{ik\,|\,j})$ where $\pi_{ik\,|\,j} = \mathrm{P}(X_i = k \,|\, \Pi_{X_i} = j)$.

Problems we solved:

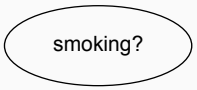
- Each local distribution has only $|\Theta_{X_i}| = (|\Pi_{X_i}| + 1)^l$ parameters.
- Each $\pi_{ik\,|\,j}$ is $\approx 1/l$ on average.

Problems we did not solve:

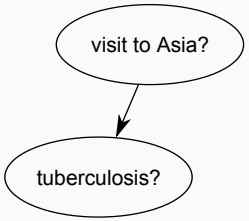
- The $\pi_{ik\,|\,j}$ still do not separate the effects of the Π_{X_i}.

Conditional Probability Tables (CPTs)

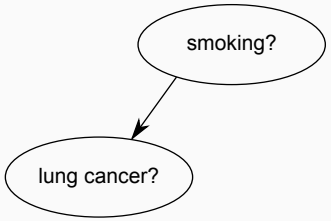
yes	no
0.01	0.99



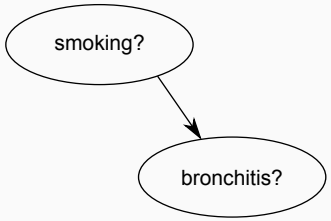
yes	no
0.50	0.50



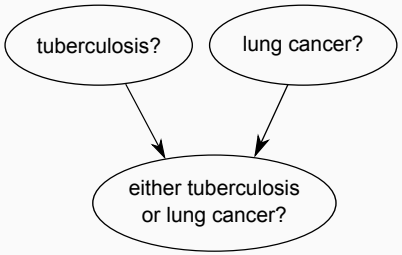
	yes	no
yes	0.05	0.95
no	0.01	0.99



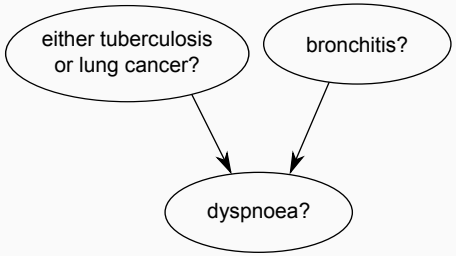
	yes	no
yes	0.10	0.90
no	0.01	0.99



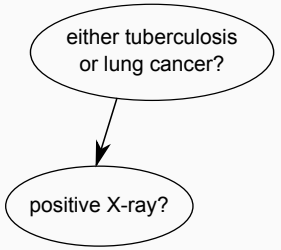
	yes	no
yes	0.60	0.40
no	0.30	0.70



	yes	no
yes:yes	1	0
yes:no	1	0
no:yes	1	0
no:no	0	1



	yes	no
yes:yes	0.90	0.10
yes:no	0.70	0.30
no:yes	0.80	0.20
no:no	0.10	0.90



	yes	no
yes	0.98	0.02
no	0.05	0.95

The local distributions $X_i \mid \Pi_{X_i}$ take the form of conditional probability tables for each node given all the configurations of the values of its parents.

Overall parameters of the $X_i \mid \Pi_{X_i}$: 18

bnlearn: Creating a Discrete BN (ASIA)

```
asia.dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

lv = c("yes", "no")

A.prob = array(c(0.01, 0.99), dim = 2, dimnames = list(A = lv))
S.prob = array(c(0.01, 0.99), dim = 2, dimnames = list(A = lv))
T.prob = array(c(0.05, 0.95, 0.01, 0.99), dim = c(2, 2),
           dimnames = list(T = lv, A = lv))
L.prob = array(c(0.1, 0.9, 0.01, 0.99), dim = c(2, 2),
          dimnames = list(L = lv, S = lv))
B.prob = array(c(0.6, 0.4, 0.3, 0.7), dim = c(2, 2),
          dimnames = list(B = lv, S = lv))
D.prob = array(c(0.9, 0.1, 0.7, 0.3, 0.8, 0.2, 0.1, 0.9), dim = c(2, 2, 2),
          dimnames = list(D = lv, B = lv, E = lv))
E.prob = array(c(1, 0, 1, 0, 1, 0, 0, 1), dim = c(2, 2, 2),
          dimnames = list(E = lv, T = lv, L = lv))
X.prob = array(c(0.98, 0.02, 0.05, 0.95), dim = c(2, 2),
          dimnames = list(X = lv, E = lv))

cpt = list(A = A.prob, S = S.prob, T = T.prob, L = L.prob, B = B.prob,
        D = D.prob, E = E.prob, X = X.prob)
asia.bn = custom.fit(asia.dag, cpt)
```

bnlearn: Conditional Probability Tables (I)

Smoking:

```
asia.bn$S
```

```
  Parameters of node S (multinomial distribution)

Conditional probability table:
 A
 yes   no
0.01 0.99
```

Lung cancer:

```
asia.bn$L
```

```
  Parameters of node L (multinomial distribution)

Conditional probability table:

     S
L      yes   no
  yes 0.10 0.01
  no  0.90 0.99
```

BNLEARN: CONDITIONAL PROBABILITY TABLES (II)

Dyspnoea:

```
asia.bn$D
```

```
  Parameters of node D (multinomial distribution)

Conditional probability table:

, , E = yes

     B
D     yes  no
  yes 0.9 0.7
  no  0.1 0.3

, , E = no

     B
D     yes  no
  yes 0.8 0.1
  no  0.2 0.9
```

Conditional Probability Tables and Logistic Regression

A conditional probability table $\mathrm{P}(X_i \mid \Pi_{X_i})$ is equivalent to a saturated multinomial logistic regression for X_i with all possible interaction terms (of all possible orders) for the Π_{X_i}: a model that has as many covariates as there are free probabilities in the conditional probability table.

For each value k of X_i, via the multinomial link function:

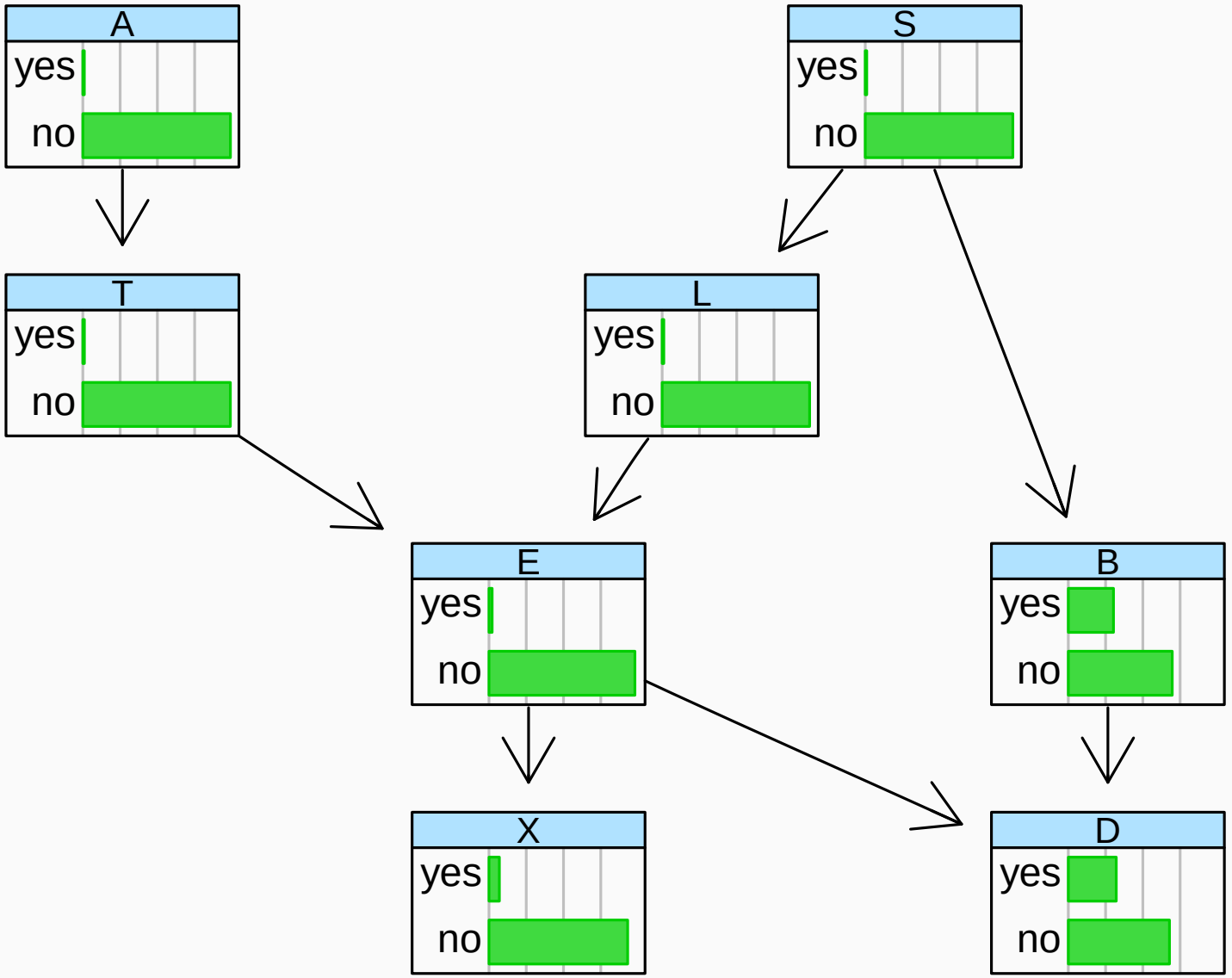
$$\log(\pi_{ik\,|\,\bullet}/\pi_{i1\,|\,\bullet}) = \eta_{ik\,|\,\bullet} \qquad\qquad \eta_{ik\,|\,\bullet} = \mathbf{Z}_i\boldsymbol{\gamma}_{ik}$$

where $\mathbf{Z}_i$ is the design matrix for the saturated model and the $\boldsymbol{\gamma}_{ik}$ are the regression coefficients.

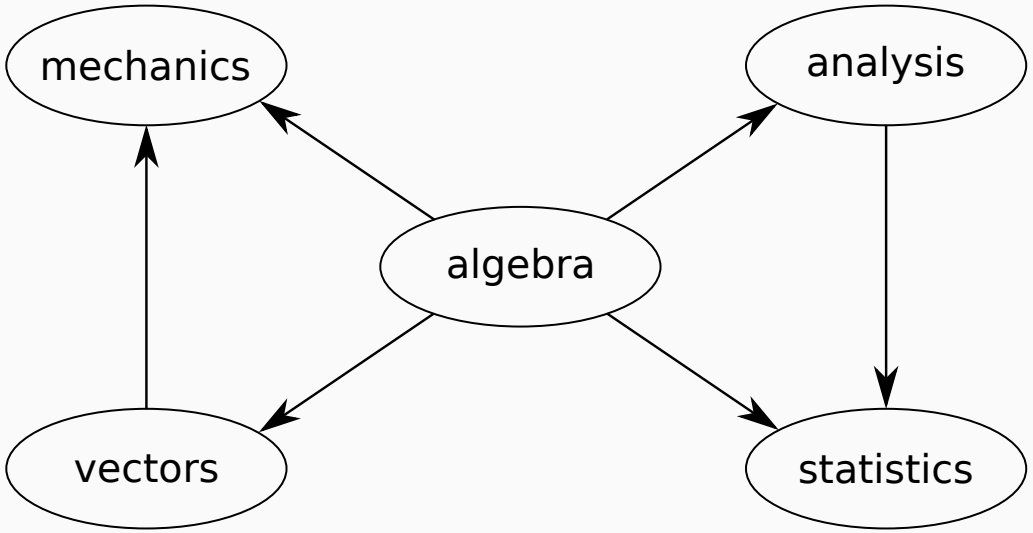
Now we can:

- separate main and interaction effects; and
- simplify the structure of Θ_{X_i} by removing higher-order interactions as needed.

Canonical Plot of a DBN



Gaussian BNs



A classic example of GBN is the MARKS network from Mardia, Kent & Bibby (1979), which describes the relationships between the marks on 5 math-related topics.

Total parameters of $\mathbf{X}$: $5 + 15 = 20$

The global distribution $\mathrm{P}(\mathbf{X})$ is

$$\mathbf{X} \sim N(\boldsymbol{\mu}, \Sigma)$$

where $\boldsymbol{\mu}$ is the vector of the means ($|\mathbf{X}| \times 1$) and Σ is the covariance matrix ($|\mathbf{X}| \times |\mathbf{X}|$).

Classical multivariate statistics tells us that

1. if we assume that Σ is full rank, we can compute the precision matrix $\Omega = \Sigma^{-1}$;
2. if $\Omega_{ij} = 0$ then $X_i \perp\!\!\!\perp_P X_j \,|\, \mathbf{X} \setminus \{X_i, X_j\}$.

In a GBN, the absence of an arc $X_i \to X_j$ in the DAG implies $X_i \perp\!\!\!\perp_G X_j \,|\, \mathbf{X} \setminus \{X_,X_j\}$, which in turn implies $X_i \perp\!\!\!\perp_P X_j \,|\, \mathbf{X} \setminus \{X_,X_j\}$.

As a result, GBNs can only model linear dependencies.

Partial Correlations and Local Distributions

The local distributions $\mathrm{P}(X_i \mid \Pi_{X_i})$ have two equivalent parametrisations:

- using partial correlations;
- using linear regressions.

The partial correlation between X_i and each of the Π_{X_i} given the others can be computed from Σ:

1. subset Σ to $\Sigma^{(i)} = \Sigma_{\{X_i, \Pi_{X_i}\}}$;
2. compute $\Omega^{(i)}$, the inverse of $\Sigma^{(i)}$;
3. the partial correlation is $\rho_{ij} = -\Omega^{(i)}_{1,j} / \sqrt{\Omega^{(i)}_{1,1} / \Omega^{(i)}_{j,j}}$.

The equivalent coefficients in the linear regression model

$$X_i = \mu_{X_i} + \Pi_{X_i} \boldsymbol{\beta}_{X_i} + \varepsilon_{X_i}, \qquad \varepsilon_{X_i} \sim N(0, \sigma^2_{X_i})$$

can be computed as $\beta_{X_i,j} = \rho_{ij} / \sqrt{\Sigma_{i,i} / \Sigma_{j,j}}$.

The Local Distributions

In practical applications, the linear regression parametrisation is more common. For the MARKS data, it works out as follows:

$$
\begin{aligned}
\mathsf{ALG} &= 50.60 + \varepsilon_{\mathsf{ALG}} \sim N(0, 112.8) \\
\mathsf{ANL} &= -3.57 + 0.99\mathsf{ALG} + \varepsilon_{\mathsf{ANL}} \sim N(0, 110.25) \\
\mathsf{MECH} &= -12.36 + 0.54\mathsf{ALG} + 0.46\mathsf{VECT} + \varepsilon_{\mathsf{MECH}} \sim N(0, 195.2) \\
\mathsf{STAT} &= -11.19 + 0.76\mathsf{ALG} + 0.31\mathsf{ANL} + \varepsilon_{\mathsf{STAT}} \sim N(0, 158.8) \\
\mathsf{VECT} &= 12.41 + 0.75\mathsf{ALG} + \varepsilon_{\mathsf{VECT}} \sim N(0, 109.8)
\end{aligned}
$$

Note that:

- parents only contribute main effects, no interactions;
- they only appear in their natural form, that is, no transforms.

Overall parameters of the $X_i \mid \Pi_{X_i}$: $11 + 5 = 16$

BNLEARN: Creating a Gaussian BN

```
marks.dag =
  model2network("[ALG][ANL|ALG][MECH|ALG:VECT][STAT|ALG:ANL][VECT|ALG]")

ALG.dist = list(coef = c("(Intercept)" = 50.60), sd = 10.62)
ANL.dist = list(coef = c("(Intercept)" = -3.57, ALG = 0.99), sd = 10.5)
MECH.dist =
  list(coef = c("(Intercept)" = -12.36, ALG = 0.54, VECT = 0.46), sd = 13.97)
STAT.dist =
  list(coef = c("(Intercept)" = -11.19, ALG = 0.76, ANL = 0.31), sd = 12.61)
VECT.dist = list(coef = c("(Intercept)" = 12.41, ALG = 0.75), sd = 10.48)

ldist = list(ALG = ALG.dist, ANL = ANL.dist, MECH = MECH.dist,
          STAT = STAT.dist, VECT = VECT.dist)
marks.bn = custom.fit(marks.dag, ldist)
```

Note that we specify the regression coefficients and the standard deviation of the residuals in keeping with the parameterisation used by R.

bnlearn: Local Linear Regressions

```
marks.bn[c("ALG", "STAT")]
```

```
$ALG

  Parameters of node ALG (Gaussian distribution)

Conditional density: ALG
Coefficients:
(Intercept)
       50.6
Standard deviation of the residuals: 10.6

$STAT

  Parameters of node STAT (Gaussian distribution)

Conditional density: STAT | ALG + ANL
Coefficients:
(Intercept)          ALG          ANL
     -11.19         0.76         0.31
Standard deviation of the residuals: 12.6
```

From Local to Global Distributions (I)

Reconstructing the global distribution from the local linear regression is also an interesting exercise. This is how `gbn2mvnorm()` does it:

1. Count the nodes and sort them in topological order (top to bottom in the DAG).

```
nodes = nodes(marks.bn)
nnodes = length(nodes)
ordered = node.ordering(marks.bn)
```

2. Compute the mean vector μ, sweeping the nodes in topological order.

```
mu = structure(rep(0, nnodes), names = nodes)
for (node in ordered) {
  pars = marks.bn[[node]]$parents
  coefs = marks.bn[[node]]$coefficients

  mu[node] = sum(c(1, mu[pars]) * coefs[c("(Intercept)", pars)])
}#FOR
mu
```

```
  ALG  ANL MECH STAT VECT
 50.6 46.5 38.1 41.7 50.4
```

From Local to Global Distributions (II)

3. Sweep the nodes again to construct an auxiliary matrix cumulating up the regression coefficients and the residual standard deviations in the lower triangular part.

```
chol = matrix(0, nrow = nnodes, ncol = nnodes,
         dimnames = list(ordered, ordered))
for (node in ordered) {
  pars = marks.bn[[node]]$parents
  coefs = marks.bn[[node]]$coefficients
  stderr = marks.bn[[node]]$sd

  chol[node, node] = stderr
  chol[node, ] = chol[node, ] + t(coefs[pars]) %*% chol[pars, ]
}
chol
```

```
         ALG   ANL  VECT MECH STAT
ALG   10.62  0.00  0.00    0  0.0
ANL   10.51 10.50  0.00    0  0.0
VECT   7.96  0.00 10.48    0  0.0
MECH   9.40  0.00  4.82   14  0.0
STAT 11.33  3.25  0.00    0 12.6
```

From Local to Global Distributions (III)

4. Compute the cross-product of `chol` to get the covariance matrix Σ.

```
sigma = (chol %*% t(chol))[nodes, nodes, drop = FALSE]
sigma
```

```
        ALG   ANL  MECH  STAT  VECT
ALG   112.8 111.7  99.8 120.3  84.6
ANL   111.7 220.8  98.8 153.3  83.7
MECH   99.8  98.8 306.7 106.5 125.4
STAT  120.3 153.3 106.5 298.0  90.2
VECT   84.6  83.7 125.4  90.2 173.3
```

For comparison, the raw value from the original MARKS data set:

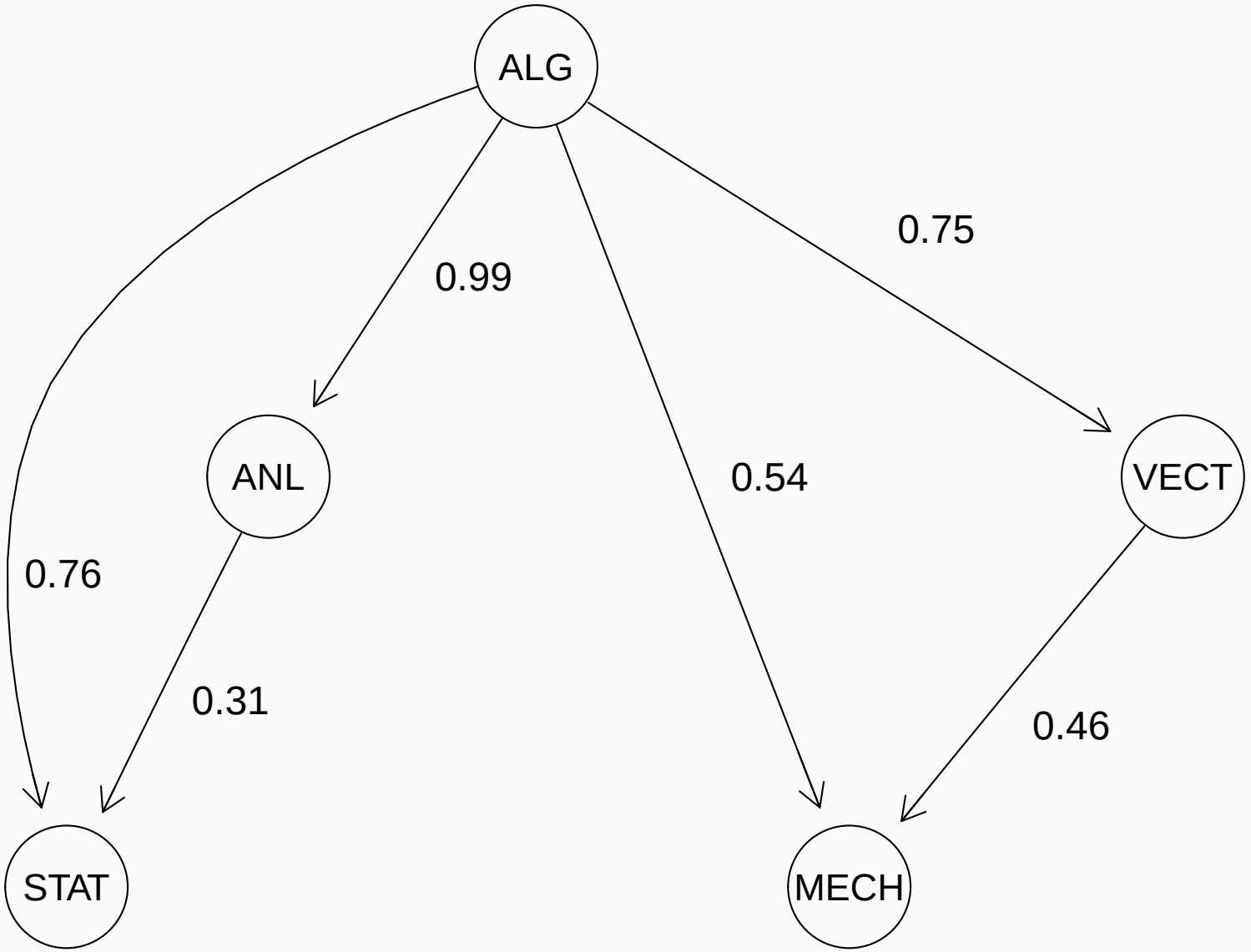
```
colMeans(marks)[nodes]
```

```
  ALG   ANL MECH STAT VECT
 50.6  46.7 39.0 42.3 50.6
```

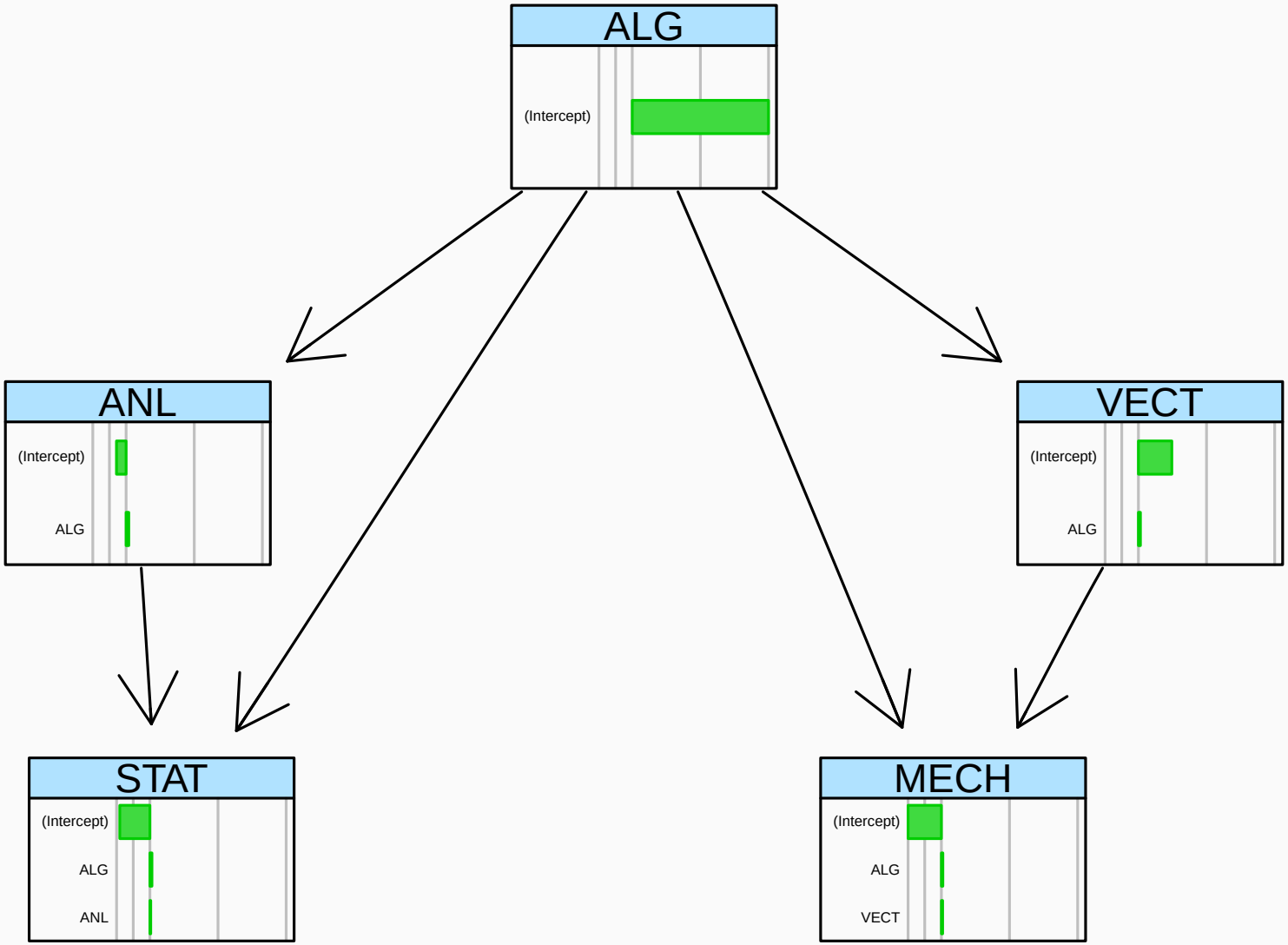
```
cov(marks)[nodes, nodes]
```

```
        ALG   ANL MECH STAT  VECT
ALG   112.9 112.1  102  122  85.2
ANL   112.1 220.4  106  156  94.7
MECH  101.6 106.3  306  117 127.2
STAT  121.9 155.5  117  298  99.0
VECT   85.2  94.7  127   99 172.8
```

Gaussian Bayesian Networks and Wright's Path Analysis



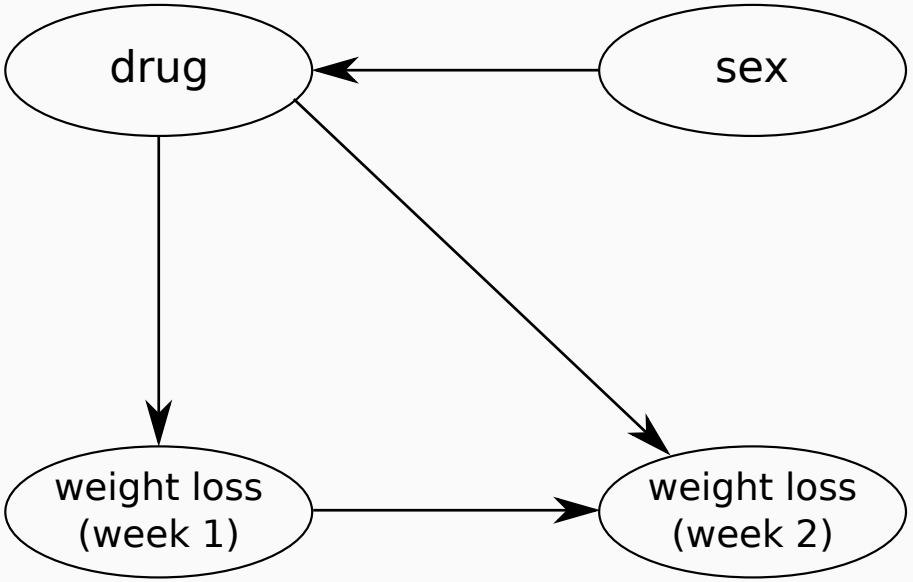
Canonical Plot of a GBN



Conditional Linear Gaussian BNs

CGBNs contain both discrete and continuous nodes, and combine DBNs and GBNs as follows to obtain a mixture-of-Gaussians distribution:

- continuous nodes cannot be parents of discrete nodes;
- the local distribution of each discrete node is a CPT;
- the local distribution of each continuous node is a set of linear regression models, one for each configurations of the discrete parents, with the continuous parents acting as regressors.



A classic example is the RATS' WEIGHTS network from Edwards (1995), which describes weight loss in a drug trial on rats.

The resulting local distribution for the first weight loss for drugs "D1", "D2" and "D3" is:

$$\begin{aligned}
\texttt{WL1}_{\texttt{D1}} &= 7 + \varepsilon_{\texttt{WL1}\,|\,\texttt{D1}} \sim N(0, 2.5) \\
\texttt{WL1}_{\texttt{D2}} &= 7.50 + \varepsilon_{\texttt{WL1}\,|\,\texttt{D2}} \sim N(0, 2) \\
\texttt{WL1}_{\texttt{D3}} &= 14.75 + \varepsilon_{\texttt{WL1}\,|\,\texttt{D3}} \sim N(0, 11)
\end{aligned}$$

with just the intercepts since WL1 has no continuous parents. The local distribution for the second weight loss is:

$$\begin{aligned}
\texttt{WL2}_{\texttt{D1}} &= 1.02 + 0.89\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D1}} \sim N(0, 3.2) \\
\texttt{WL2}_{\texttt{D2}} &= -1.68 + 1.35\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D2}} \sim N(0, 4) \\
\texttt{WL2}_{\texttt{D3}} &= -1.83 + 0.82\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D3}} \sim N(0, 1.9)
\end{aligned}$$

and contains a regression coefficient for WL1.

The Global Distribution (I)

The discrete nodes define the components of a mixture, each component is the joint multivariate distribution of the continuous nodes.

Drug has states "D1", "D2" and "D3". Sex has states "M" and "F". So the mixture has 6 components:

$$\Big\{\{\texttt{"D1"},\texttt{"M"}\},\{\texttt{"D1"},\texttt{"F"}\},\{\texttt{"D2"},\texttt{"M"}\},\{\texttt{"D2"},\texttt{"F"}\},\{\texttt{"D3"},\texttt{"M"}\},\{\texttt{"D3"},\texttt{"F"}\}\Big\}$$

The first component, for which Drug is "D1" and sex is "M", identifies the local distributions of weight loss as:

$$\begin{aligned}\texttt{WL1}_{\texttt{D1}} &= 7 + \varepsilon_{\texttt{WL1}\,|\,\texttt{D1}} \sim N(0, 2.5)\\ \texttt{WL2}_{\texttt{D1}} &= 1.02 + 0.89\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D1}} \sim N(0, 3.2)\end{aligned}$$

Together, they can be thought of as a GBN and therefore we can write them as a multivariate normal distribution as we did earlier.

The Global Distribution (II)

The component for {"D1", "F"} is identical to that for {"D1", "M"} because weight loss is independent from sex given drug.

Along the same lines, the components for both {"D2", "M"} and {"D2", "F"} identify the local distributions of weight loss as:

$$\begin{aligned}\texttt{WL1}_{\texttt{D2}} &= 7.50 + \varepsilon_{\texttt{WL1}\,|\,\texttt{D2}} \sim N(0, 2)\\ \texttt{WL2}_{\texttt{D2}} &= -1.68 + 1.35\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D2}} \sim N(0, 4)\end{aligned}$$

For {"D3", "M"} and {"D3", "F"}:

$$\begin{aligned}\texttt{WL1}_{\texttt{D3}} &= 14.75 + \varepsilon_{\texttt{WL1}\,|\,\texttt{D3}} \sim N(0, 11)\\ \texttt{WL2}_{\texttt{D3}} &= -1.83 + 0.82\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D3}} \sim N(0, 1.9)\end{aligned}$$

In total, we have a mixture model with six components, each with a bivariate normal distribution.

BNLEARN: Creating a Conditional Linear Gaussian BN

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

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

The regression coefficients are stored in a matrix with one conditional regression in each column: each column corresponds to one configuration of the discrete parents and each row to one of the continuous parents.

bnlearn: Mixtures of Linear Regressions

```
bn$WL2
```

```
  Parameters of node WL2 (conditional Gaussian distribution)

Conditional density: WL2 | DRUG + WL1
Coefficients:
                  0      1      2
(Intercept)    1.02  -1.68  -1.83
WL1            0.89   1.35   0.82
Standard deviation of the residuals:
   0     1     2
1.78  2.00  1.37
Discrete parents' configurations:
   DRUG
0    D1
1    D2
2    D3
```

The local distribution of each continuous node contains a set of regressions, all with the same coefficients. Consider WL2: we can write the intercept and each coefficient as the deviation from the respective means across the drugs.

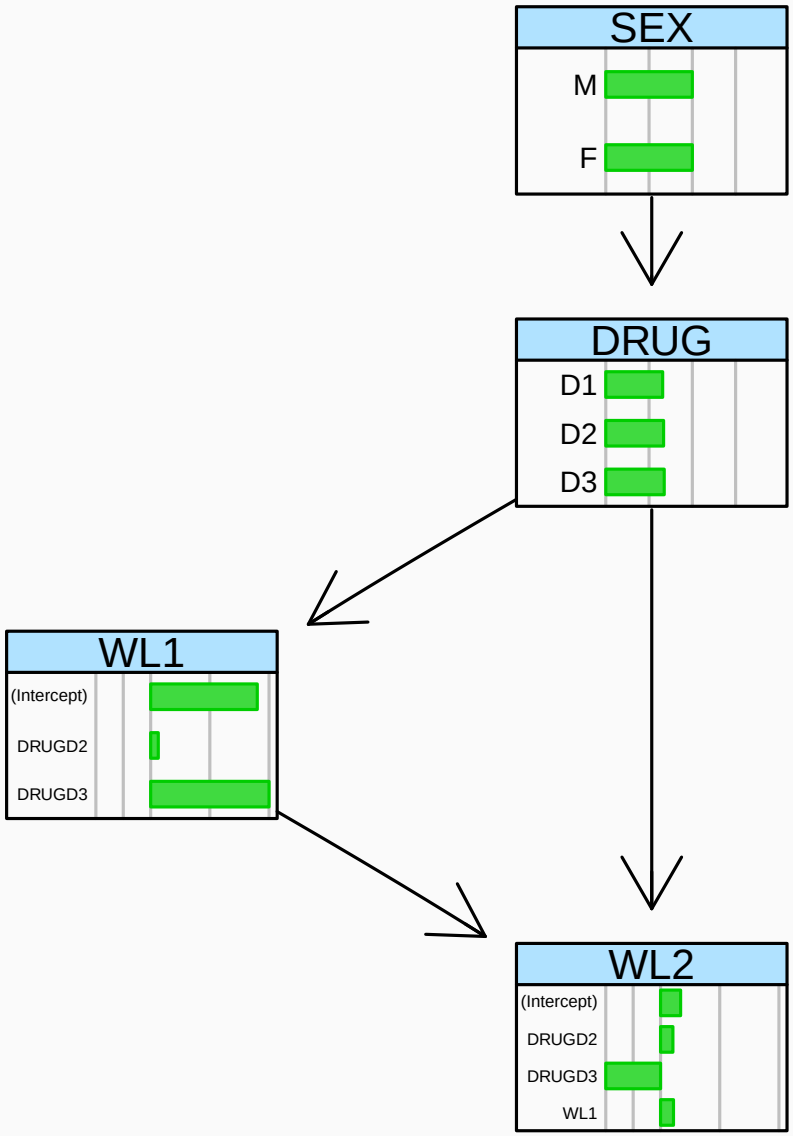
$$\begin{aligned}
\texttt{WL2}_{\texttt{D1}} &= (1.85 - 0.83) + (-0.13 + 1.02)\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D1}} \\
\texttt{WL2}_{\texttt{D2}} &= (-0.85 - 0.83) + (0.33 + 1.02)\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D2}} \\
\texttt{WL2}_{\texttt{D3}} &= (-1.00 - 0.83) + (-0.20 + 1.02)\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D3}}
\end{aligned}$$

This representation looks much like a mixed-effects models with a fixed intercept (0.83), a fixed slope (1.02), a random intercept (one for each drug) and a random slope (one for each drug).

Limitations of These Probability Distributions

- No real-world data set follows a multivariate Gaussian distribution: even if the marginal distributions are normal, not all dependence relationships are linear.
- Computing partial correlations can be problematic because of singularities.
- Parametric assumptions for mixed data have strong limitations: they impose constraints on which arcs may be present in the DAG.
- Discretisation is a common solution to the above problems, but it may discard useful information and it is tricky to get right (choosing a set of intervals such that the dependence relationships involving the original variable are preserved). On the other hand, dependencies are no longer required to be linear.
- Ordinal variables are treated as categorical, again losing information.

Canonical Plot of a CGBN



If the structure of the model is known, the problem of estimating the parameters of the global distribution can be solved by estimating the parameters of each local distribution, one at a time.

Common choices are:

- Maximum likelihood estimators: just the usual empirical estimators. Often described as either maximum entropy or minimum divergence estimators in information-theoretic literature.
- Bayesian posterior estimators: based on conjugate priors to keep computations fast, simple and in closed form.
- Shrinkage estimators: regularised estimators based either on James-Stein or Bayesian shrinkage results.

Maximum Likelihood and Maximum Entropy Estimators

The classic estimators for (conditional) probabilities and (partial) correlations / regression coefficients have known limitations:

- maximum likelihood estimates are unstable in most multivariate problems, both discrete and continuous;
- for the multivariate Gaussian distribution, James & Stein proved in the 1950s that the maximum likelihood estimator for the mean is not admissible in 3+ dimensions;
- partial correlations are often ill-behaved because of that, even with Moore-Penrose pseudo-inverses;
- maximum likelihood estimates are non-smooth and create problems when using the graphical model for inference.

Maximum a Posteriori Bayesian Estimators

Bayesian posterior estimates are the sensible choice for parameter estimation according to Koller's & Friedman's tome on graphical models. Choices for the priors are limited (for computational reasons) to conjugate distributions, namely:

- the Dirichlet for discrete models,

$$Dir(\alpha_{k\,|\,\Pi_{X_i}=\pi}) \quad \overset{data}{\longrightarrow} \quad Dir(\alpha_{k\,|\,\Pi_{X_i}=\pi} + n_{k\,|\,\Pi_{X_i}=\pi})$$

  meaning that $\hat{p}_{k\,|\,\Pi_{X_i}=\pi} = \alpha_{k\,|\,\Pi_{X_i}=\pi} / \sum_{\pi} \alpha_{k\,|\,\Pi_{X_i}=\pi}$;
- the Inverse Wishart for Gaussian models,

$$IW(\Psi, m) \quad \overset{data}{\longrightarrow} \quad IW(\Psi + n\Sigma, m + n).$$

In both cases, the only free parameter with a non-informative prior is the equivalent or imaginary sample size, which gives the relative weight of the prior compared to the observed sample.

Bayesian LASSO and Ridge Regression

Gaussian graphical models, being closely related with linear regression, have also used ridge regression (L_2 regularisation) and LASSO (L_1 regularisation) in their Bayesian capacity.

LASSO corresponds to a Laplace prior on the regression coefficients,

$$\beta_k \,|\, \sigma^2 \sim Laplace(0, \sigma^2).$$

Ridge Regression corresponds to a Gaussian prior,

$$\beta_k \,|\, \sigma^2 \sim N(0, \sigma^2).$$

In both cases, tuning the σ^2 parameter is crucial, as it takes the role of the λ regularisation parameter found in the original frequentist definitions of these methods. Furthermore, excessive regularisation might lead to zero coefficients that would make a node independent of its parents.

Shrinkage estimation is based on results from James & Stein on the estimation of the mean of a multivariate Gaussian distribution, and takes the form

$$\tilde{\theta} = \lambda t + (1 - \lambda)\hat{\theta} \qquad \lambda \in [0, 1]$$

where the optimal λ (with respect to squared loss) can be estimated in closed form as

$$\lambda^* = \min \left(\frac{\sum_k \mathrm{VAR}(\hat{\theta}_k) - \mathrm{COV}(\hat{\theta}_k, t_k) + \mathrm{Bias}(\hat{\theta}_k)\,\mathrm{E}(\hat{\theta}_k - t_k)}{\sum_k (\hat{\theta}_k - t_k)^2}, 1 \right).$$

The James-Stein estimator $\tilde{\theta}$ dominates the maximum likelihood estimator $\hat{\theta}$ and converges to the latter as the sample size grows. It can be interpreted as an empirical Bayes estimator.

For discrete data, conditional probabilities $\pi_{ik\,|\,j}$ are estimated as

$$\tilde{\pi}_{ikj} = \lambda^* t_{ikj} + (1-\lambda^*)\hat{\pi}_{ikj}, \quad \lambda^* = \min\left(\frac{1-\sum_k \hat{\pi}^2_{ikj}}{(n-1)\sum_k (t_{ikj}-\hat{p}_{ikj})^2}, 1\right),$$

where t is the uniform (discrete) distribution.

For continuous data, correlations end up being estimated from the shrunk covariance matrix $\widetilde{\Sigma}$

$$\tilde{\sigma}_{ii} = \hat{\sigma}_{ii}, \quad \tilde{\sigma}_{ij} = (1-\lambda^*)\hat{\sigma}_{ij}, \quad \lambda^* = \min\left(\frac{\sum_{i\neq j} \mathrm{VAR}(\hat{\sigma}_{ij})}{\sum_{i\neq j} \hat{\sigma}^2_{ij}}, 1\right)$$

where t is $\mathrm{diag}(\widehat{\Sigma})$. $\widetilde{\Sigma}$ is guaranteed to have full rank, so it can be safely inverted to get partial correlations.

Parameter learning is implemented in `bn.fit()` and defaults to `method = "mle"`; for discrete data we can also use Bayesian posterior estimation with `method = "bayes"` with an imaginary sample size `iss`.

```
fitted = bn.fit(asia.dag, asia, method = "mle")
coef(fitted$X)
```

```
      E
X          no     yes
  no  0.95659 0.00541
  yes 0.04341 0.99459
```

```
fitted = bn.fit(asia.dag, asia, method = "bayes", iss = 20)
coef(fitted$X)
```

```
      E
X          no    yes
  no  0.9556 0.0184
  yes 0.0444 0.9816
```

bnlearn implements only `method = "mle-g"` for GBNs, but we can use `penalized()` to replace parameter estimates with ridge, LASSO, or elastic net estimates.

```
library(penalized)
fitted = bn.fit(marks.dag, marks, method = "mle-g")
coef(fitted$STAT)
```

```
  (Intercept)          ALG          ANL
      -11.192        0.765        0.316
```

```
fitted$STAT = penalized(response = marks[, "STAT"],
                penalized = marks[, parents(fitted, "STAT")],
                lambda2 = 100, model = "linear", trace = FALSE)
coef(fitted$STAT)
```

```
  (Intercept)          ALG          ANL
      -10.788        0.753        0.321
```

We can also fit the parameters directly using a DAG and any other regression functions, and collecting their coefficients in a BN with `custom.fit()`.

Same for CGBNs.

```
library(lme4)
data(rats, library = "gRbase")
rats[, c("W1", "W2")] = list(as.numeric(rats$W1), as.numeric(rats$W2))
names(rats) = c("SEX", "DRUG", "WL1", "WL2")
fitted = bn.fit(rats.dag, data = rats, method = "mle-cg")
coef(fitted$WL2)
```

```
                0     1      2
(Intercept) 1.028 -1.68 -1.835
WL1         0.889  1.36  0.819
```

```
ldist = list(coef = array(0, dim = c(2, 3),
          dimnames = list(c("(Intercept)", "WL1"), NULL)), sd = rep(0, 3))
model = lmer(WL2 ~ WL1 + (1 | DRUG), data = rats)
for (i in seq(ncol(ldist$coef)))
  ldist$coef[, i] = fixef(model) + ranef(model)[["DRUG"]][i, ]
for (i in seq(length(ldist$sd)))
  ldist$sd[i] = sd(resid(model)[rats[, "DRUG"] == levels(rats[, "DRUG"])[i]])
fitted$WL2 = ldist
coef(fitted$WL2)
```

```
                0    1      2
(Intercept) 1.57 2.36 -1.42
WL1         1.54 2.32 -1.45
```

The definition of a BN just says

$$\mathrm{P}(\mathbf{X}, \Theta) = \prod_{i=1}^{N} \mathrm{P}(X_i \mid \Pi_{X_i}; \Theta_{X_i}) \quad \text{where} \quad \Pi_{X_i} = \{\text{parents of } X_i\}.$$

We are free to choose our $\mathrm{P}(X_i \mid \Pi_{X_i}; \Theta_{X_i})$, treating any hierarchical Bayesian model as a BN and using it as such. However, we should include nodes only for the variables in the model, not their parameters, to be able to interpret it in the same way as a classical BN.

What we gain:

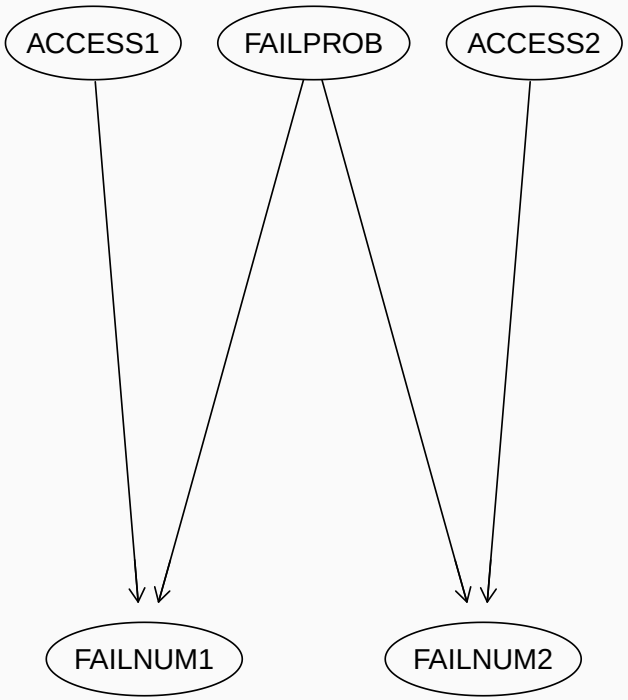
- Freedom! Power!
- The ability to use Bayesian modelling software such as Stan.

What we lose:

- Closed-form results for density functions, parameter estimates, scores, tests, etc.
- The ability to use **bnlearn**.

A General BN

A simple example is the RELIABILITY TESTING model for two-phase testing (somewhat amended here) by Fenton & Neil, assessing the distribution of the number of failures based on the number of accesses and the system's reliability.



$$\texttt{FAILPROB} \sim Beta\,(2,5)$$
$$\texttt{ACCESS1} \sim Pois\,(20)$$
$$\texttt{ACCESS2} \sim Pois\,(400)$$
$$\texttt{FAILNUM1} \sim Bi\,(\texttt{ACCESS1}, \texttt{FAILPROB})$$
$$\texttt{FAILNUM2} \sim Bi\,(\texttt{ACCESS2}, \texttt{FAILPROB})$$

RSTAN: Building a General BN as a Hierarchical Model (I)

This is the Stan model specification to generate observations from the BN, given its parameters. We store that as a string in a variable `stancode` to use Stan through R via **rstan**.

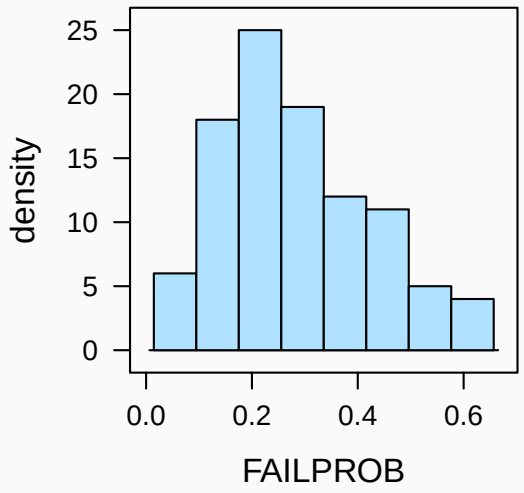
```
data {
  vector[2] Fp;  // shape parameters for the beta distribution.
  real A1p;      // expected accesses per second, phase 1
  real A2p;      // expected accesses per second, phase 2.
}
generated quantities {
  real FAILPROB;
  int ACCESS1;
  int ACCESS2;
  int FAILNUM1;
  int FAILNUM2;

  FAILPROB = beta_rng(Fp[1], Fp[2]);
  ACCESS1 = poisson_rng(A1p);
  ACCESS2 = poisson_rng(A2p);
  FAILNUM1 = binomial_rng(ACCESS1, FAILPROB);
  FAILNUM2 = binomial_rng(ACCESS2, FAILPROB);
}
```

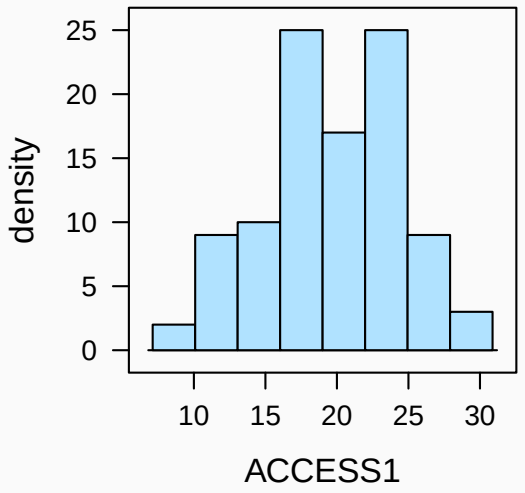
rstan: Building a General BN as a Hierarchical Model (II)

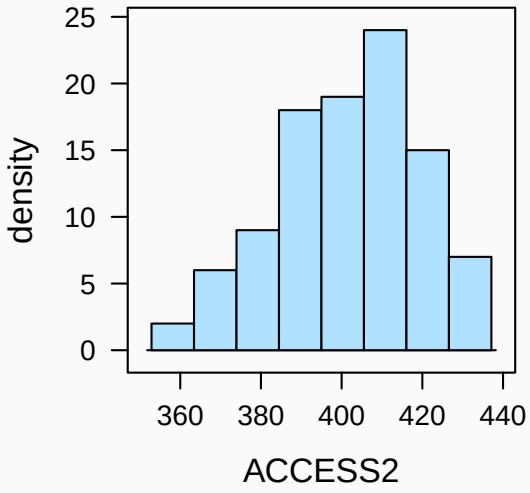
```
library(rstan)
reliability.bn = stan_model(model_code = stancode)
params = list(
  Fp = c(2, 5),
  A1p = 20,
  A2p = 400
)
data = sampling(reliability.bn, algorithm = "Fixed_param", data = params,
        thin = 10, iter = 500, seed = 42)
nodes = c("FAILPROB", "ACCESS1", "ACCESS2", "FAILNUM1", "FAILNUM2")
reliability = as.data.frame(extract(data)[nodes])
```

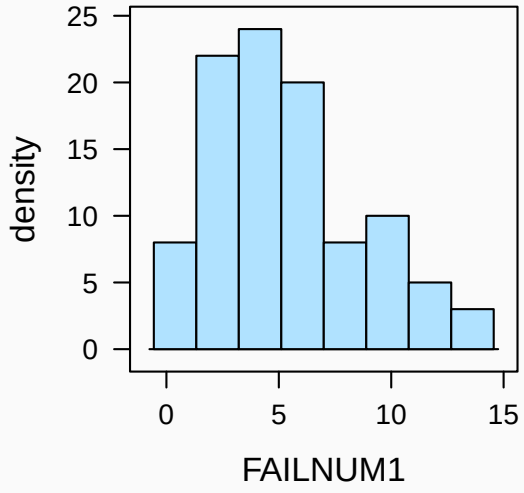
Unlike **bnlearn**'s `bn.fit` objects, **rstan**'s models are not accessible programmatically from R because they live in the C++ code of the Stan library.

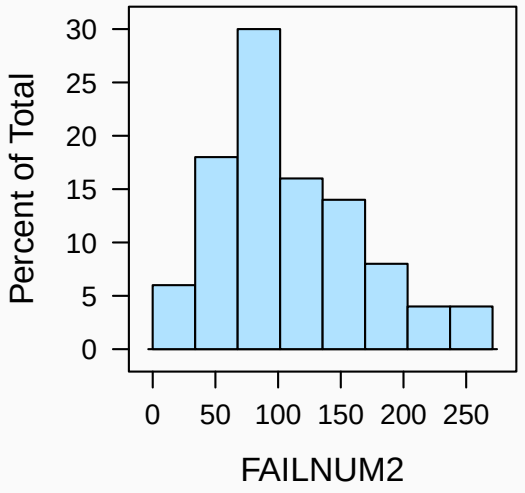
Some Plots of the Empirical Marginal Distributions











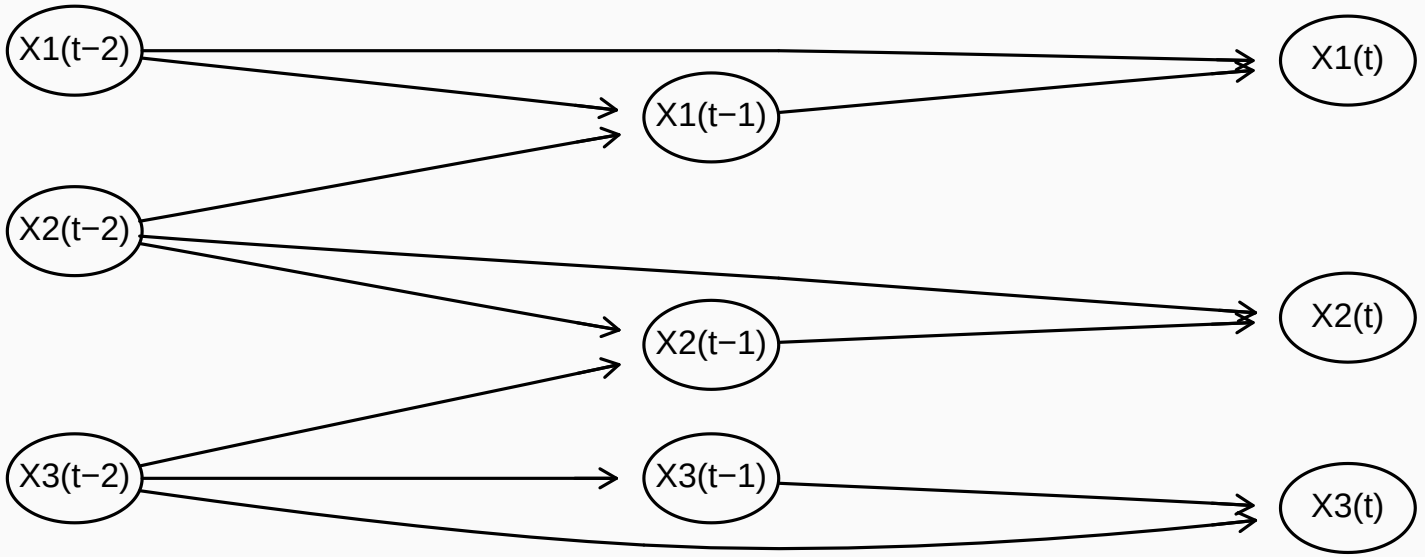
Vector Auto-Regressive Processes

Multivariate time series can modelled as vector auto-regressive (VAR) processes. A $\mathrm{VAR}(\mathbf{X}, p)$ of order p has the form

$$\mathbf{X}(t) = A_1\mathbf{X}(t-1) + ... + A_p\mathbf{X}(t-p) + B + \varepsilon_{\mathbf{X}}, (t)$$

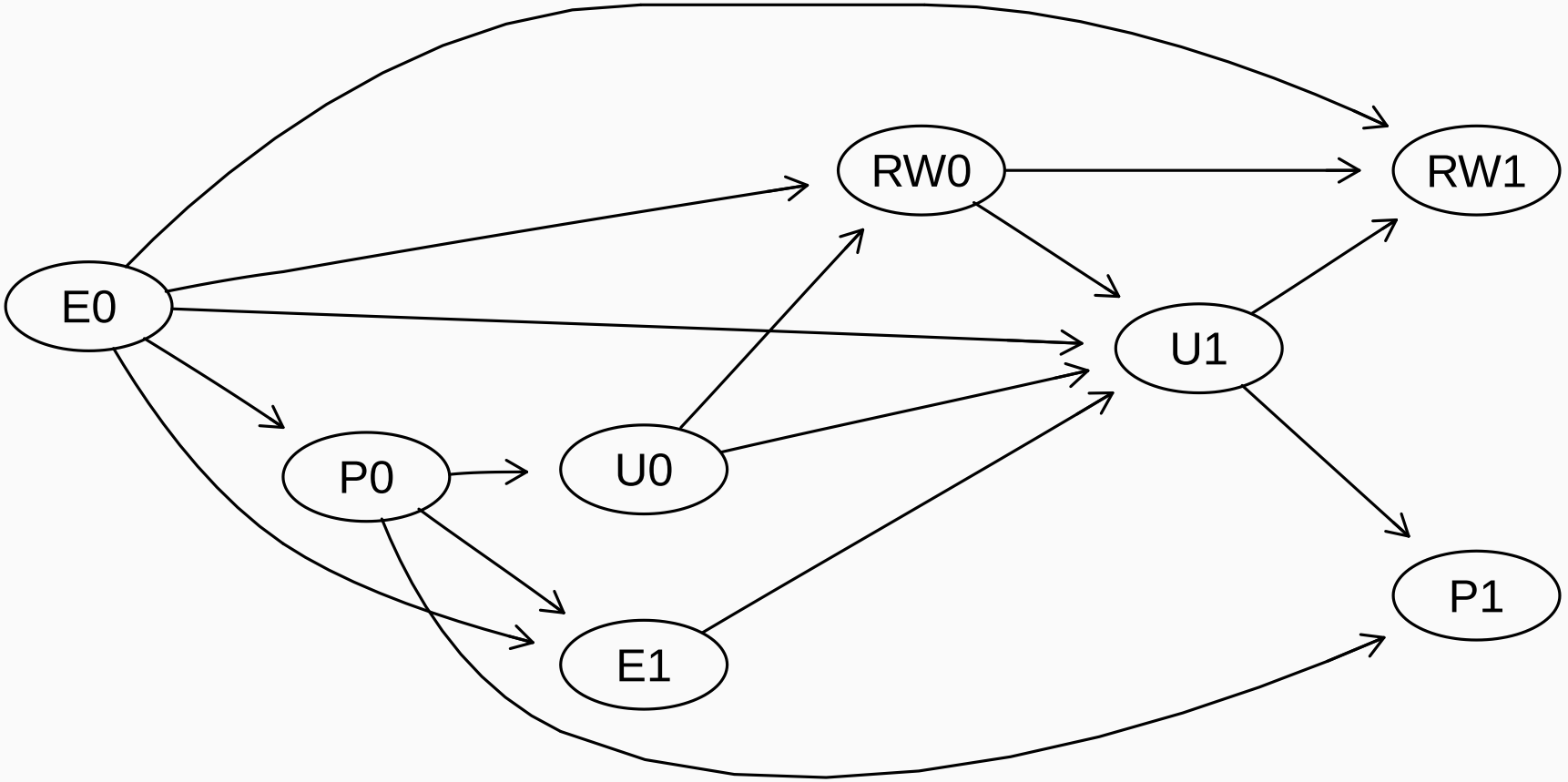
where $\mathbf{X}(t)$ are the variables measured at time t.

In graphical form, for instance ($p = 2$):



Does it remind you of something?

The classic CANADA macroeconomics time series, studied by Lütkepohl and Pfaff. E is employment, P is productivity, RW is the real wage and U is the unemployment rate.



The Graphical Structure

Any stochastic process in discrete time has a similar graphical representation and can be treated as a BN. In such a dynamic BN:

- All variables $\mathbf{X}$ become separate nodes in each time point.
- Arcs can only go forward in time: the graph is still a DAG.
- We may allow instantaneous arcs between nodes in the same time point, but their interpretation is often not straightforward.
- We can model feedback loops between variables.

We typically assume that:

- arcs can only come from nodes in the previous time point (so, the time series is of order $p = 1$).
- no instantaneous arcs, unless time points are averages over periods.
- the time series is homogeneous and stationary.

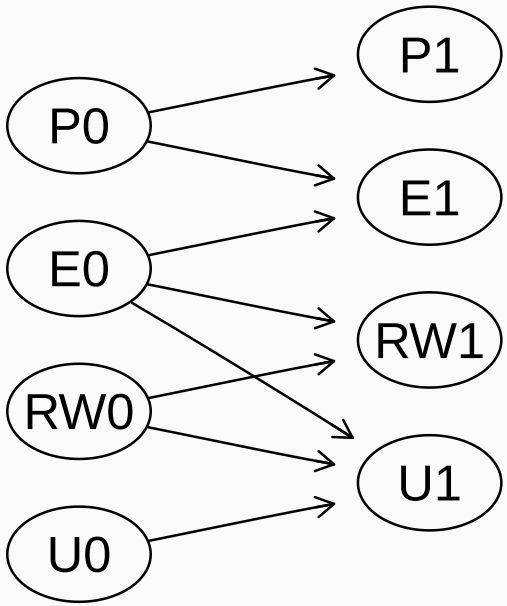
We do not assume any particular distribution for the nodes.

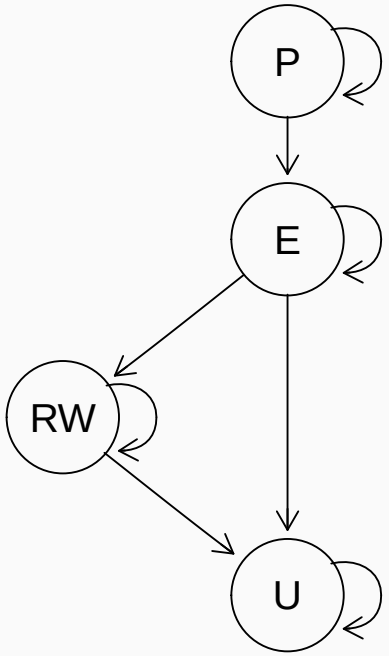
Rolled-Up and Unrolled Graphs for a Dynamic BN

With these assumptions, we can completely represent a dBN with:

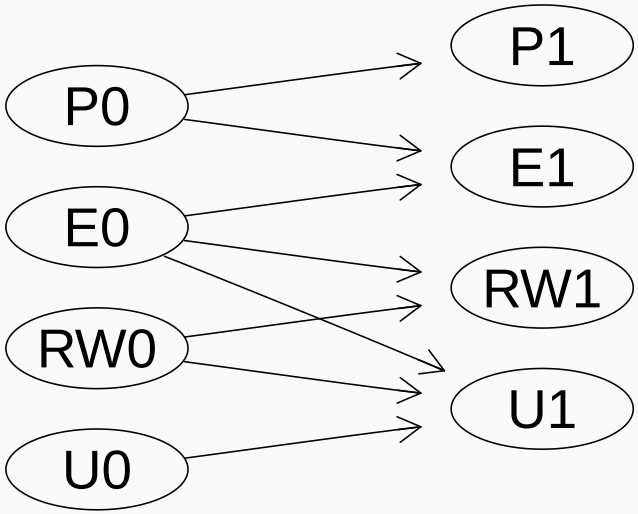
- a 2-time BN representing the transition between two consecutive time points;
- the marginal distribution of each variable in $\mathbf{X}$, which models the dBN at time $t = 0$.

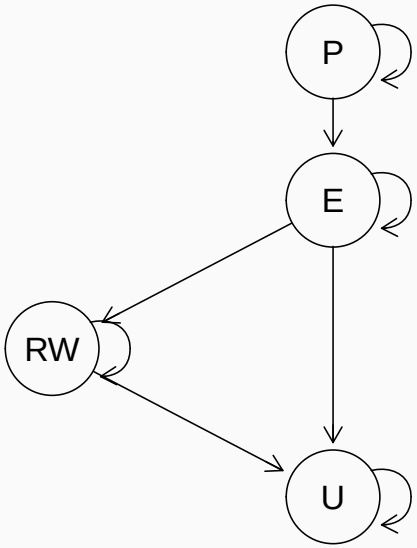
The 2-time BN has two representations: unrolled and rolled-up.

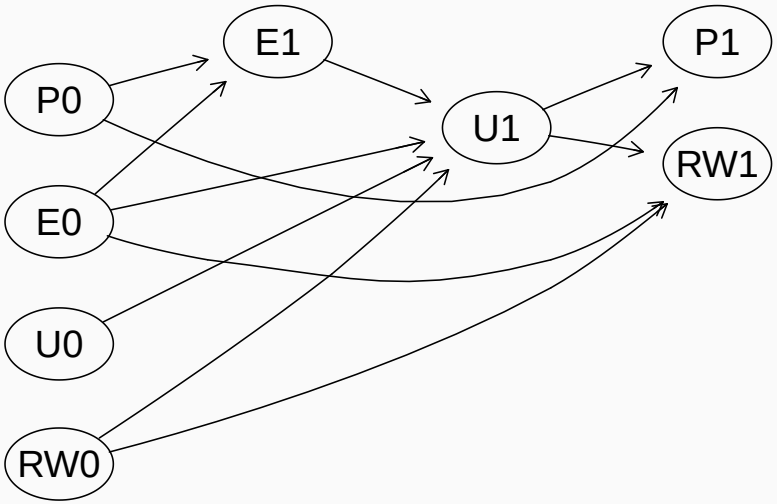


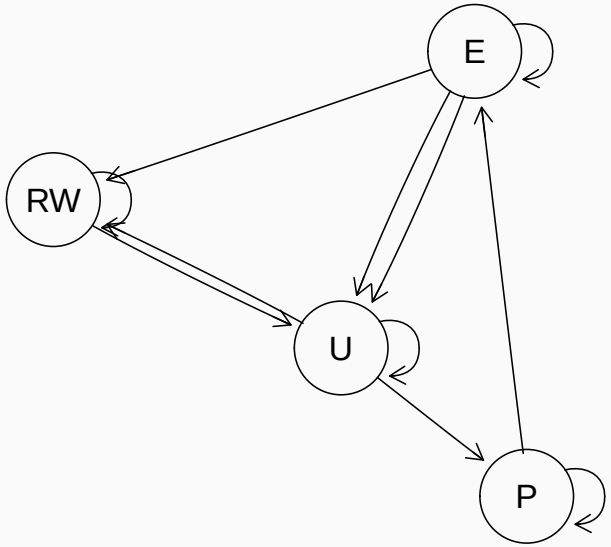


2-Time BNs With and Without Instantaneous Arcs







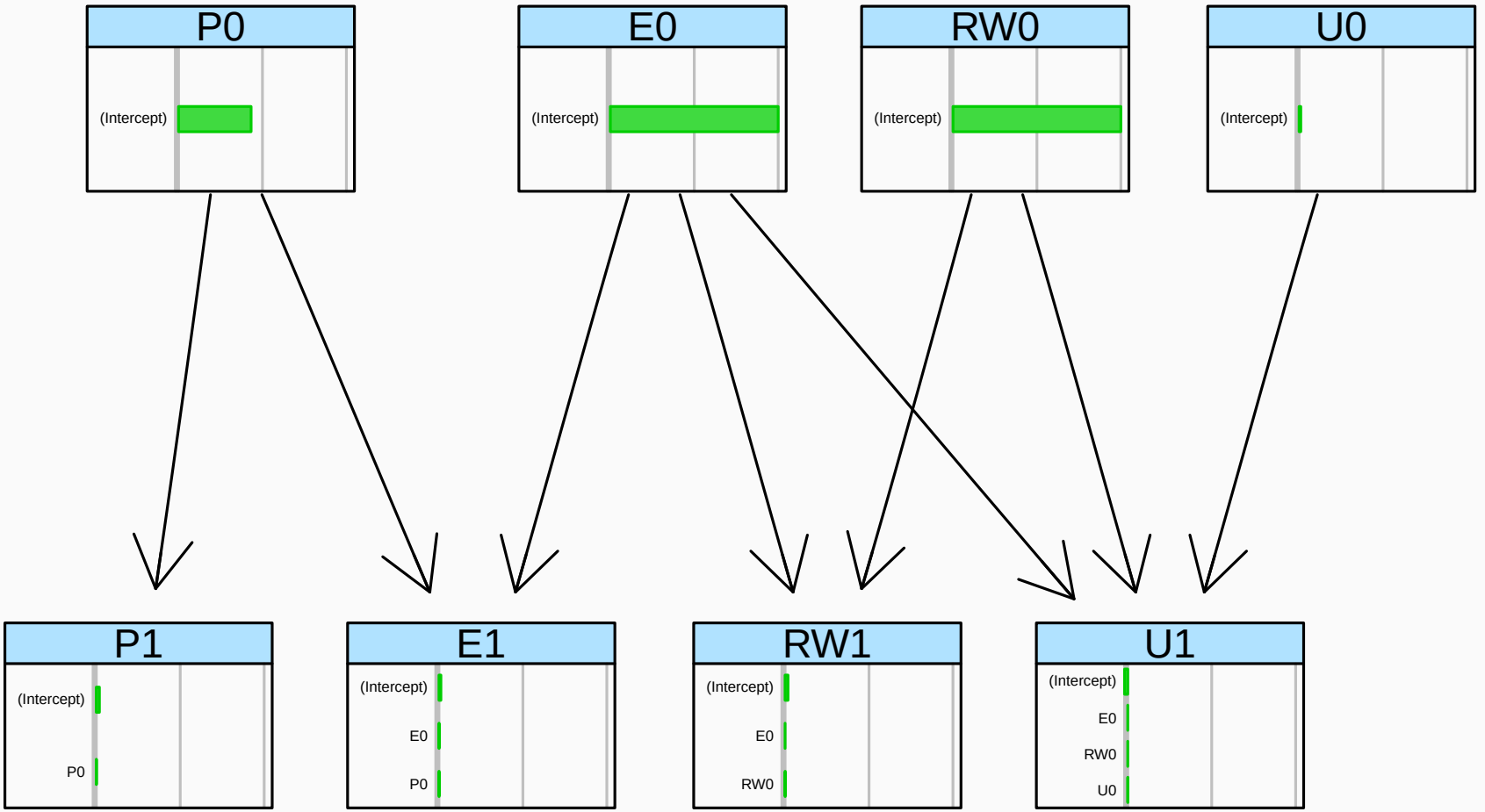


bnlearn: Creating a Dynamic BN for a VAR

```
canada.dag =
  model2network("[P0][E0][U0][RW0][P1|P0][E1|E0:P0][U1|U0:E0:RW0][RW1|RW0:E0]")
P0.dist = list(coef = c("(Intercept)" = 408), sd = 4.23)
E0.dist = list(coef = c("(Intercept)" = 944), sd = 9.07)
U0.dist = list(coef = c("(Intercept)" = 9.34), sd = 1.6)
RW0.dist = list(coef = c("(Intercept)" = 944), sd = 9.07)
P1.dist = list(coef = c("(Intercept)" = 17.51, P0 = 0.96), sd = 0.70)
E1.dist = list(coef = c("(Intercept)" = 10.89, E0 = 1.04, P0 = 1.04), sd = 0.52)
U1.dist = list(coef = c("(Intercept)" = -17.85, E0 = 0.02, RW0 = -0.004,
                        U0 = 1.004), sd = 0.43)
RW1.dist = list(coef = c("(Intercept)" = 16.45, E0 = -0.04, RW0 = 1.03),
                sd = 0.88)
ldist = list(P0 = P0.dist, E0 = E0.dist, U0 = U0.dist, RW0 = RW0.dist,
             P1 = P1.dist, E1 = E1.dist, U1 = U1.dist, RW1 = RW1.dist)
canada.bn = custom.fit(canada.dag, ldist)
```

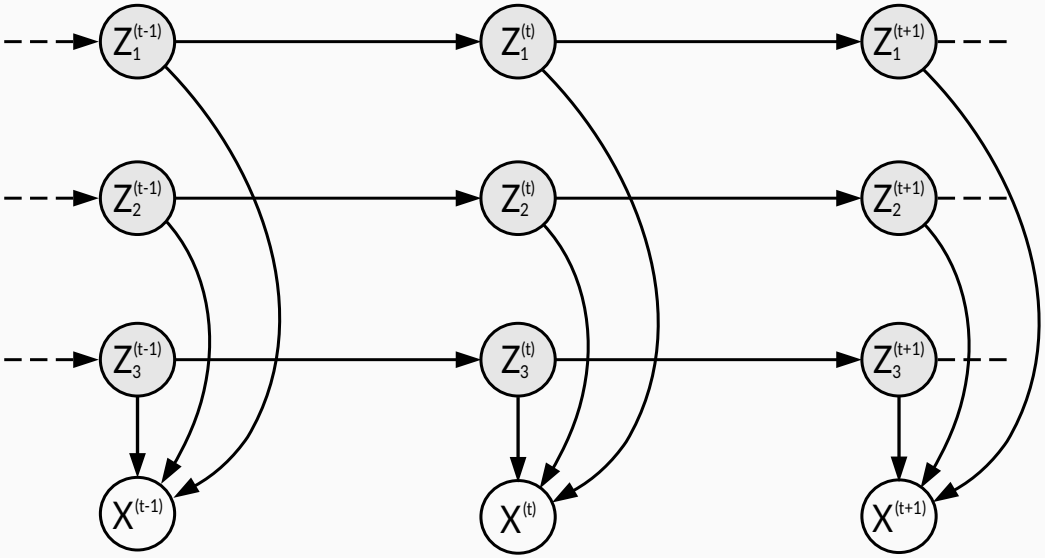
Code-wise, it looks much like creating a GBN. If the nodes were discrete, the code would look like that for creating a DBN. With mixed node types, the code would look like that for creating CGBNs or general BNs.

Canonical Plots Work for Dynamic BNs as Well

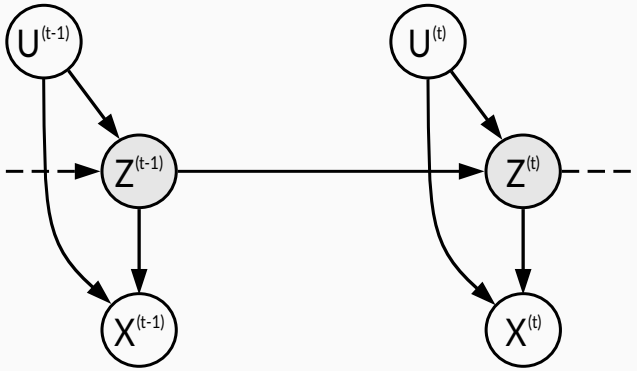


Many Models are Dynamic BNs

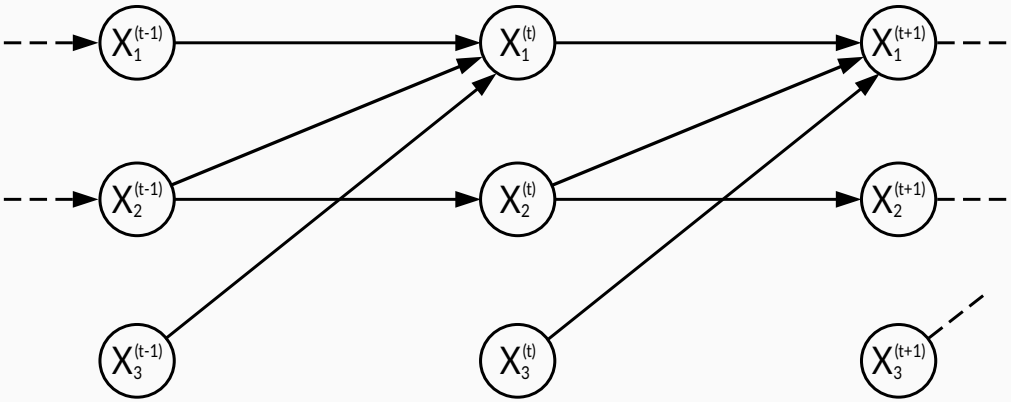
Factorial HMM



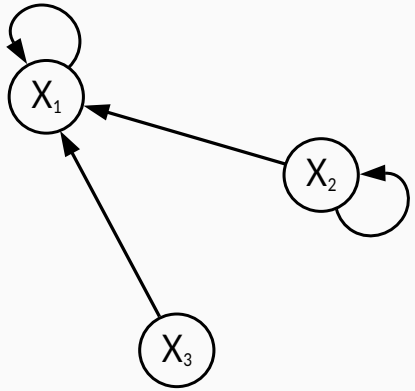
Kalman Filter



Unrolled VAR



Rolled VAR



Dynamic BN: Hidden Markov Models

Hidden Markov models (HMMs) are one of the most widespread approaches to model phenomena with hidden state, that is, in which the behaviour of the observed variables $\mathbf{X}$ depends on that of one or more discrete latent variables $\mathbf{Z}$ as well as on other variables in $\mathbf{X}$.

In dBN terms, an HMM model with M latent variables can be written as

$$\mathrm{P}\left(\mathbf{X}^{(t)} \mid \mathbf{X}^{(t-1)}, \mathbf{Z}^{(t)}\right) = \prod_{i=1}^{N} \mathrm{P}\left(X_i^{(t)} \mid \Pi_{X_i^{(t)}}, \mathbf{Z}^{(t)}\right)$$

$$\mathrm{P}\left(\mathbf{Z}^{(t)}\right) = \prod_{j=1}^{M} \mathrm{P}\left(Z_j^{(t)} \mid \Pi_{Z_j^{(t)}}\right),$$

with the restriction that the parents of $Z_j^{(t)}$ can only be other latent variables. In the vast majority of the literature all variables are assumed to be discrete. Depending on the choice of $\Pi_{Z_j^{(t)}}$, we can obtain various HMM variants such as hierarchical HMMs and factorial HMMs.

Dynamic BN: Vector Auto-Regressive Models

Vector auto-regressive models (VARs) are a straightforward multivariate extension of univariate auto-regressive time series for continuous variables.

VARs are defined as

$$\mathbf{X}^{(t)} = A_1\mathbf{X}^{(t-1)} + ... + A_L\mathbf{X}^{(t-L)} + \varepsilon_t, \qquad \varepsilon_t \sim N(0, \Sigma),$$

for some fixed Markov order L. We can rewrite that as

$$\mathbf{X}^{(t)} \mid \mathbf{X}^{(t-1)}, ..., \mathbf{X}^{(t-L)} \sim N\left(A_1\mathbf{X}^{(t-1)} + ... + A_L\mathbf{X}^{(t-L)}, \varepsilon_t\right)$$

and then restrict the parents of each $X_i^{(t)}$ to those for which the corresponding regression coefficients in $A_1, ..., A_L$ are different from zero using the one-to-one correspondence between regression coefficients and partial correlations. Formally, $X_j^{(t-l)} \in \Pi_{X_i^{(t)}}$ if and only if $A_l[i, j] \neq 0$, which makes it possible to write a VAR as a Gaussian dBN.

Dynamic BNs: Kalman Filters

Kalman filters combine traits of both HMMs and VARs: like VARs, they are linear Gaussian dBNs; but they also have latent variables like HMMs.

In their simplest form, Kalman Filters include a layer of one or more latent variables that model the unobservable part of the phenomenon,

$$\mathbf{Z}^{(t)} = A\mathbf{Z}^{(t-1)} + B\mathbf{U}^{(t)} + \zeta_t, \qquad \zeta_t \sim N(0, \Psi)$$

feeding into one or more observed variables

$$\mathbf{X}^{(t)} = C\mathbf{Z}^{(t)} + D\mathbf{U}^{(t)} + \varepsilon_t, \qquad \varepsilon_t \sim N(0, \Sigma)$$

with independent Gaussian noise added in both layers. Both layers often include additional (continuous) explanatory variables $\mathbf{U}$ and can also be augmented with (discrete) switching variables to allow for different regimes. If we exclude the latter, the assumption is that the system is jointly Gaussian: that makes it possible to frame Kalman filters as dBNs in the same way we did for VARs.

Relevant Functions in **bnlearn**

- creating BNs: `model2network()` `custom.fit()`.
- plotting DAGs: `graphviz.plot()`, along with `nodeRenderInfo()`, `edgeRenderInfo()` and `renderGraph()` from the **Rgraphviz** package.
- plotting BNs: `graphviz.chart()`.

Borrowed from **rstan**: `stan_model()` and `sampling()`.

Summary and Remarks

- BNs can take quite different forms depending on what assumptions we make on their local/global distributions and on their structure.
- Discrete BNs (DBNs) model categorical data using conditional probability tables (or equivalently multinomial logistic regressions).
- Gaussian BNs (GBNs) model continuous data as multivariate normal distributions (or equivalently linear regressions).
- Conditional Gaussian BNs (CGBNs) model mixed discrete and continuous data as a mixture model (like a mixed-effects model).
- General BNs are essentially hierarchical Bayesian models.
- Dynamic BNs (dBNs) model a variety of time series and stochastic processes in discete time.

Next:

- How do we make a computer system answer questions using a BN?

Thanks!

Any questions?

Bayesian Networks in Policy and Society

Day 3: Probabilistic Inference

Marco Scutari

Dalle Molle Institute for
Artificial Intelligence (IDSIA)

August 10, 2022

Events, Evidence and Queries

A BN represents a working model of the world that a computer can understand; but how does a computer system use it to help and perform its assigned task?

We ask questions, and the computer system performs probabilistic inference to answer them and decide what to do in the process.

Questions that can be asked are called queries and are typically about an event of interest given some evidence. The evidence is the input to the computer system ("Someone with a high-school degree.") and the event is the output ("A man driving a car."). This is often called belief update: we observe some evidence and we update our beliefs before taking action.

Events, Evidence and Queries

The two most common queries are

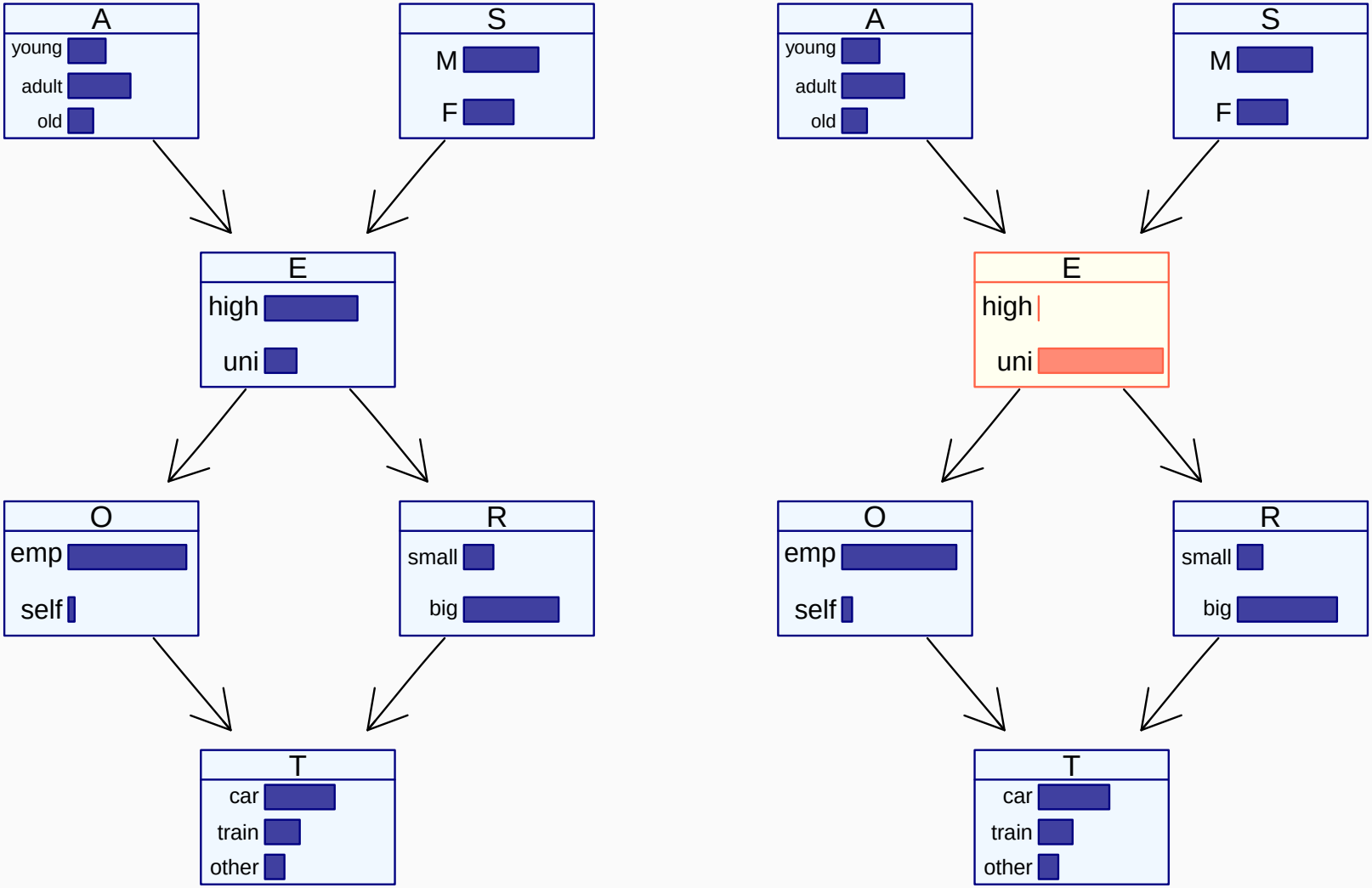
- conditional probability queries ("What is the probability that someone with high-school degree is a man driving a car?"); and
- most probable explanation queries ("What is the most probable sex and mode of transportation for someone with a high-school degree?")

In both cases the evidence is hard evidence: we set some variables to particular values. Then the computer system checks how the probabilities of other variables change and provides an answer to the query.

No more manual probability calculations..

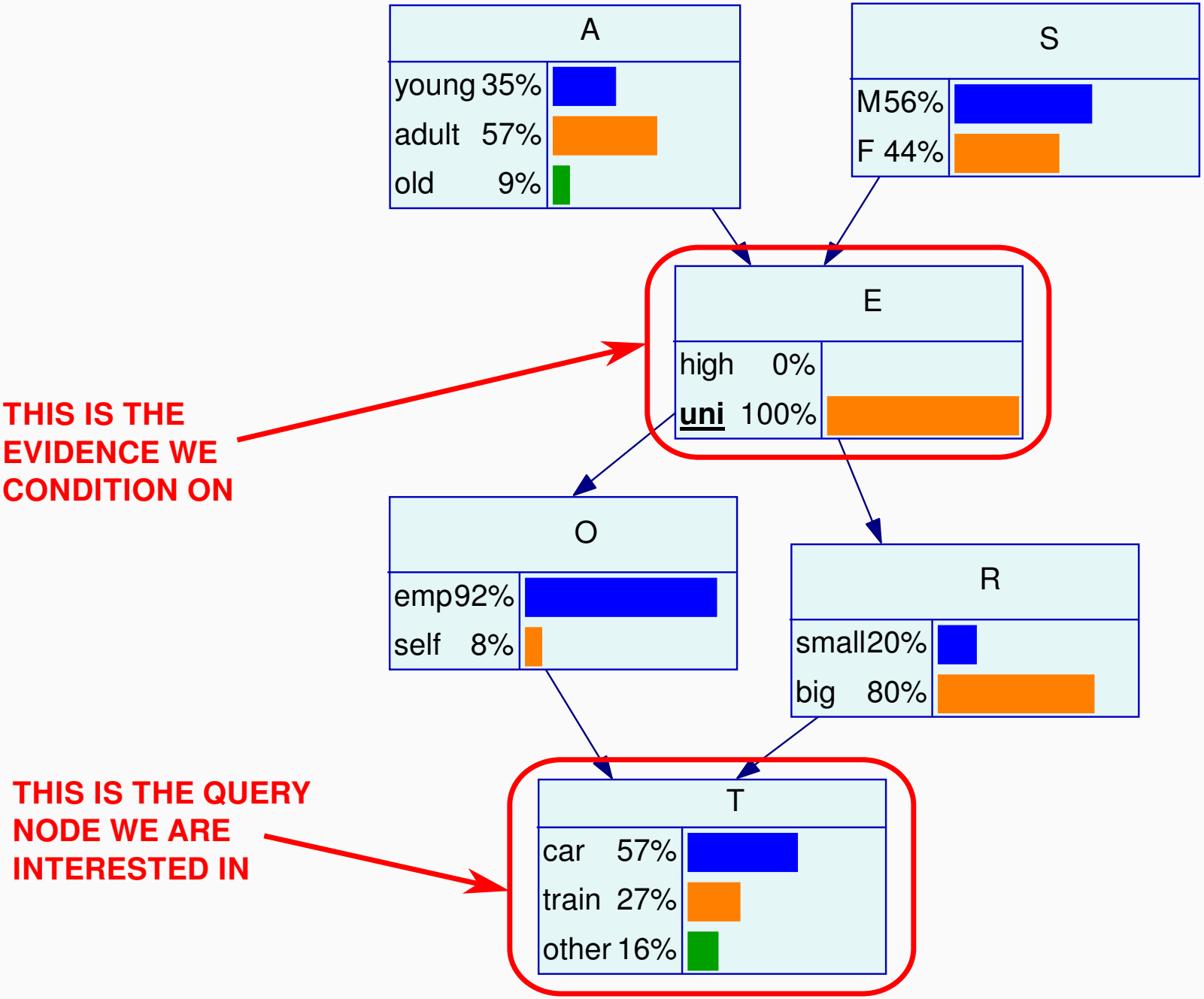
NOTE: we will initially consider only DBNs for ease of exposition, and get back to other types of BNs later.

The Effects of Conditioning on Hard Evidence

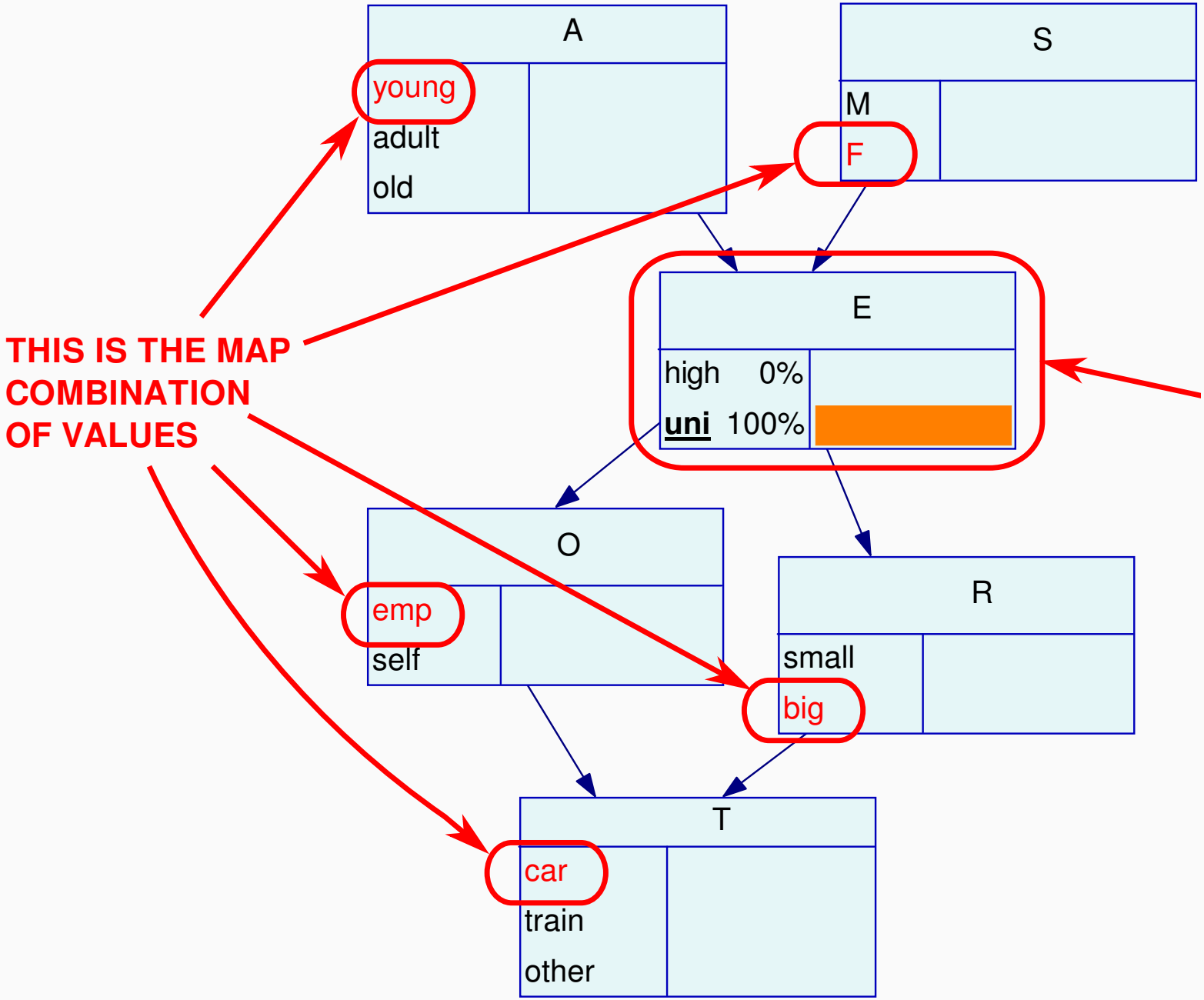


The original survey BN (left), and the posterior BN with hard evidence on Education (right).

Conditional Probability Queries in Pictures



Maximum a Posteriori Queries in Pictures



Exact and Approximate Inference

There are two approaches to answer queries using BNs.

Exact algorithms use the DAG to schedule and perform repeated applications of Bayes theorem on the local probability distributions in the BN. In other words, the computer system uses the DAG to perform all the math we did by hand in earlier lectures.

The two best known are

- variable elimination; and
- belief updates based on junction trees.

PROS: they return exact values for the probabilities of interest.
CONS: they do not scale well when BNs have many nodes and many arcs.

Exact and Approximate Inference

Approximate algorithms use the BN as a model of the world in a very literal sense. In the real world to answer some question in a scientific, rigorous way we would perform an experiment and observe the outcome. Approximate algorithms imitate this process by generating random observations from the BN, thus running a simulated experiment that approximates reality.

The two best known are

- logic sampling; and
- likelihood weighting.

PROS: they scale really well when BNs have many nodes and many arcs.
CONS: they return approximate, estimated values for the probabilities of interest.

The Logic Sampling Algorithm

INPUT: a BN, evidence E and query event Q.

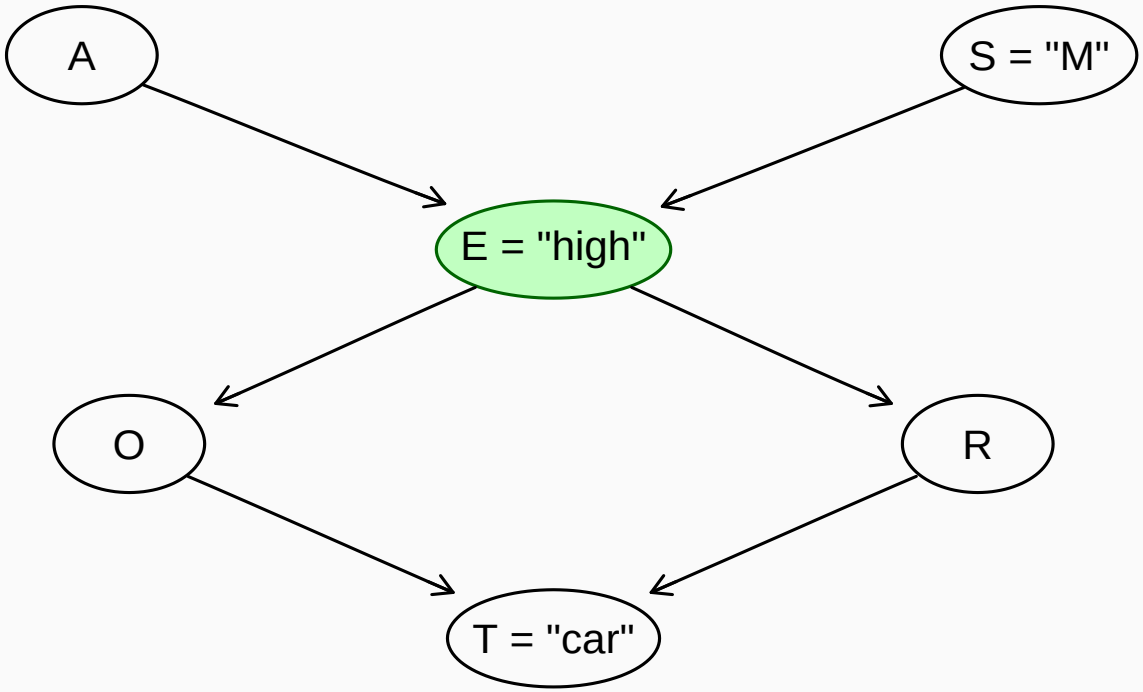
1. Order the variables in $\mathbf{X}$ according to the topological ordering in the DAG (from top to bottom), so that parents come before children.
2. Set $n_E = 0$ and $n_{E,Q} = 0$.
3. For a suitably large number of samples $\mathbf{x}$:
   - 3.1 generate a random value from each $X_i \mid \Pi_{X_i}$ taking advantage of the fact that, thanks to the topological ordering, by the time we are considering X_i we have already generated the values of all its parents Π_{X_i};
   - 3.2 if $\mathbf{x}$ includes E, set $n_E = n_E + 1$;
   - 3.3 if $\mathbf{x}$ includes both Q and E, set $n_{E,Q} = n_{E,Q} + 1$.
4. The answer to the query is the estimated probability $n_{E,Q}/n_E$.

A Survey Example

Consider:

- the evidence: someone whose Education (`E`) level is a high school diploma (`high`)...
- the event: ... is a man (`S` is equal to `M`) uses a `car` as a means of Transportation (`T`).

We will answer this query using the different inference algorithms.



Stepping Through Logic Sampling

First, we sample from the BN with `rbn()`, which takes a `bn.fit` object and the number of random samples to generate as arguments.

```
particles = rbn(bn, 10^6)
head(particles, n = 5)
```

```
       A    E   O   R S     T
1   old high emp big M train
2   old high emp big M   car
3 adult high emp big F   car
4   old high emp big M other
5 young high emp big M   car
```

The samples have the correct types and format as derived from the BN, and they are stored in a data frame that has the same structure as that of the data that were used to learn the BN (if any).

Stepping Through Logic Sampling

Then we count how many of those samples that match the evidence E to estimate $\mathrm{P}(E)$.

```
partE = particles[(particles[, "E"] == "high"), ]
nE = nrow(partE)
```

We also count how many of those samples that match the evidence E and the query event Q to estimate $\mathrm{P}(Q, E)$.

```
partEQ =
  partE[(partE[, "S"] == "M") & (partE[, "T"] == "car"), ]
nEQ = nrow(partEQ)
```

Finally, we estimate

$$\mathrm{P}(Q \mid E) = \frac{\mathrm{P}(Q, E)}{\mathrm{P}(E)}.$$

```
nEQ/nE
```

```
[1] 0.343
```

The cpquery() Function

These steps are implemented in cpquery(), with the obvious arguments:

- event is Q;
- evidence is E;
- method is "ls" for logic sampling (the default);
- n is the number of random samples.

```
cpquery(bn, event = (S == "M") & (T == "car"),
          evidence = (E == "high"), method = "ls", n = 10^6)
```

```
[1] 0.343
```

Both event end evidence are expressions that are evaluated on the random samples much like subset() would, so they must evaluate to a vector of TRUE and FALSE values (hence & and not &&).

More Advanced Queries with cpquery()

Specifying the arguments requires some care, but the result is an extremely flexible framework to compute the probability of arbitrary combinations of events.

As an example of a more complex query, we can compute

$$\mathrm{P}(\mathtt{S} = \texttt{"M"}, \mathtt{T} = \texttt{"car"} \,|\, \{\mathtt{A} = \texttt{"young"}, \mathtt{E} = \texttt{"uni"}\} \cup \{\mathtt{A} = \texttt{"adult"}\}),$$

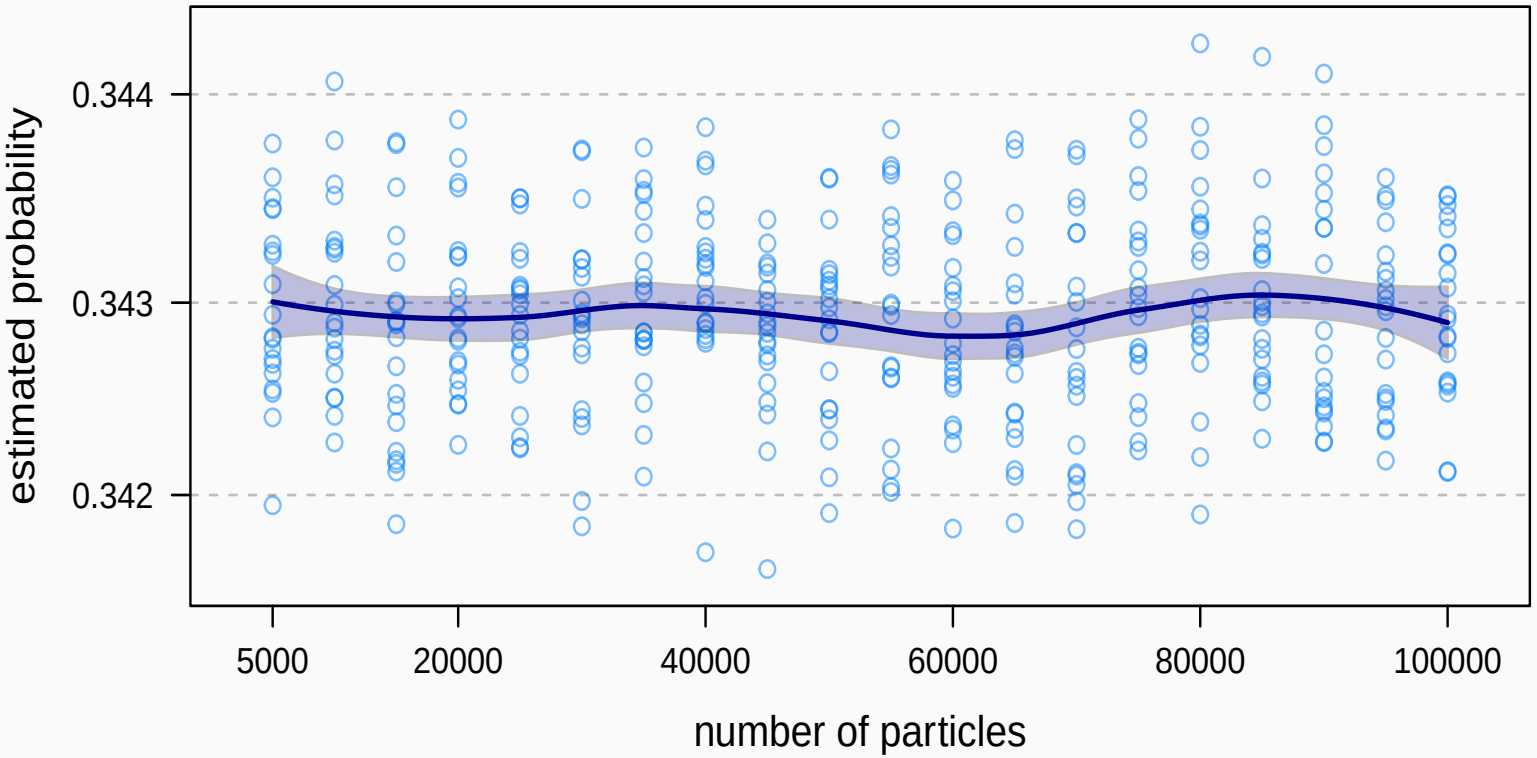
the probability of a man travelling by car given that his Age is "young" and his Education is "uni" or that he is an "adult", regardless of his Education. That would be:

```
cpquery(bn, event = (S == "M") & (T == "car"),
  evidence = ((A == "young") & (E == "uni")) | (A == "adult"))
```

```
[1] 0.349
```

Stepping Through Logic Sampling

```
nparticles = seq(from = 5 * 10^3, to = 10^5, by = 5 * 10^3)
prob = matrix(0, nrow = length(nparticles), ncol = 20)
for (i in seq_along(nparticles))
  for (j in 1:20)
    prob[i, j] = cpquery(bn, event = (S == "M") & (T == "car"),
                         evidence = (E == "high"), method = "ls", n = 10^6)
```



The Limits of Logic Sampling

Notice anything in the figure in the previous slide?

- Logic sampling is obviously affected by sampling variability: every time we run it we get a different estimate of the probability that is the answer to our query because the random samples we generate will be different.

- Sampling variability decreases with the number of samples we generate, but it never goes to zero: there is always some uncertainty around the exact value we estimate (here 0.343 ± 0.001).

- Remember that we essentially discard all random samples that do not match the evidence we condition on, so if the evidence has low probability we are throwing out almost all samples we generate.

The Likelihood Weighting Algorithm

An improvement over logic sampling, designed to solve this problem, is the likelihood weighting algorithm. Unlike logic sampling, all the random samples generated by likelihood weighting include the evidence E by design.

1. Order the variables in $\mathbf{X}$ according to the topological ordering in the DAG (from top to bottom), so that parents come before children.
2. Set $w_E = 0$ and $w_{E,Q} = 0$.
3. For a suitably large number of samples $\mathbf{x}$:
   - 3.1 generate a random value from each $X_i \mid \Pi_{X_i}$ and fix the relevant variables to the values specified by the evidence E.
   - 3.2 compute the weight $w_{\mathbf{x}} = \mathrm{P}(E)$.
   - 3.3 set $w_E = w_E + w_{\mathbf{x}}$;
   - 3.4 if $\mathbf{x}$ includes Q , set $w_{E,Q} = w_{E,Q} + w_{\mathbf{x}}$.
4. The answer to the query is the estimated probability $w_{E,Q}/w_E$.

Stepping Through Likelihood Weighting

We do not want to sample from the original BN, but from the BN in which all the nodes covered by E are fixed. This network is called the mutilated network.

Compare:

```
coef(bn$E)
```

```
, , S = M

      A
E      young adult  old
  high  0.75  0.72 0.88
  uni   0.25  0.28 0.12

, , S = F

      A
E      young adult  old
  high  0.64  0.70 0.90
  uni   0.36  0.30 0.10
```

```
parents(bn, "E")
```

```
[1] "A" "S"
```

```
mutbn = mutilated(bn, list(E = "high"))
coef(mutbn$E)
```

```
high  uni
   1    0
```

```
parents(mutbn, "E")
```

```
character(0)
```

No parents, and the value is that in the evidence with probability equal to 1.

Stepping Through Likelihood Weighting

Simply sampling from `mutbn` is not a correct way of answering our query! A simple empirical check tells us that the naive estimate we would draw from `mutbn` is wrong, since it does not match the exact value we got earlier.

```
particles = rbn(mutbn, 10^6)
partE = particles[(particles[, "E"] == "high"), ]
partEQ = partE[(particles[, "S"] == "M") &
               (particles[, "T"] == "car"), ]
nrow(partEQ) / nrow(partE)
```

```
[1] 0.336
```

That is because `nrow(partE)` is identical to `nrow(particles)` by construction, so the conditional probability is not computed correctly. What we get is:

$$\mathrm{P}(Q, E) = \frac{n_{E,Q}}{n} \neq \mathrm{P}(Q \mid E).$$

Stepping Through Likelihood Weighting

The weights adjust for the fact that we are sampling from the mutilated BN instead of the original BN. The weights are just the likelihood components associated with the nodes we are conditioning on (E in this case):

```
w = logLik(bn, particles, nodes = "E", by.sample = TRUE)
wEQ = sum(exp(w[(particles[, "S"] == "M") &
                (particles[, "T"] == "car")]))
wE = sum(exp(w))
wEQ/wE
```

```
[1] 0.343
```

NOTE: the likelihood of an observation has the same mathematical expression as its probability, so for practical purposes here it is just $\mathrm{P}(E)$. `logLik()` returns $\log \mathrm{P}(E)$ in the code above.

Stepping Through Likelihood Weighting

More conveniently, we can perform likelihood weighting with `cpquery()` by setting `method = "lw"` and specifying the evidence as a named list with one element for each node we are conditioning on.

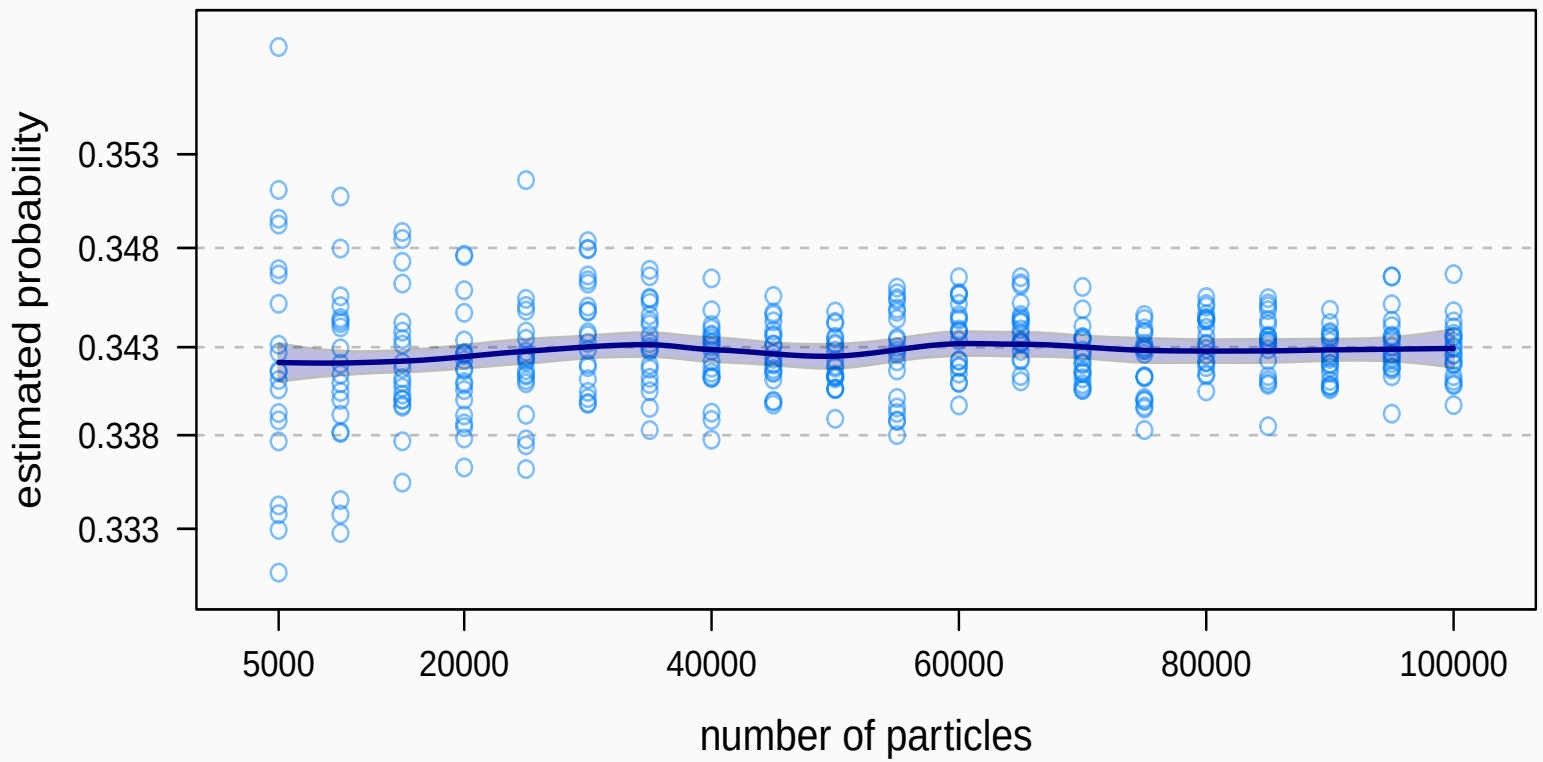
```
cpquery(bn, event = (S == "M") & (T == "car"),
          evidence = list(E = "high"), method = "lw", n = 5 * 10^4)
```

```
[1] 0.344
```

The estimate we obtain is still very precise with small numbers of random samples, as was the case for logic sampling, but the variability of the estimated probabilities is actually larger. There is no guarantee that likelihood weighting will always have lower variance than logic sampling.

Stepping Through Likelihood Weighting

```
nparticles = seq(from = 5 * 10^3, to = 10^5, by = 5 * 10^3)
prob = matrix(0, nrow = length(nparticles), ncol = 20)
for (i in seq_along(nparticles))
  for (j in 1:20)
    prob[i, j] = cpquery(bn, event = (S == "M") & (T == "car"),
                   evidence = list(E = "high"), method = "lw",
                   n = nparticles[i])
```



Then Why Use Likelihood Weighting?

Logic sampling will be computationally inefficient and very inaccurate if $\mathrm{P}(E)$ is small because most random samples will be discarded without contributing to the estimation of $\mathrm{P}(Q \mid E)$.

```
extreme.dag = model2network("[A][B|A]")
A.prob = array(c(0.999999, 0.000001), dim = 2,
           dimnames = list(A = c("a1", "a2")))
B.prob = array(c(0.5, 0.5, 0.75, 0.25), dim = c(2, 2),
           dimnames = list(B = c("b1", "b2"), A = c("a1", "a2")))
extreme.bn = custom.fit(extreme.dag, list(A = A.prob, B = B.prob))
cpquery(extreme.bn, event = (B == "b2"), evidence = (A == "a2"),
  method = "ls", n = 10^6)
```

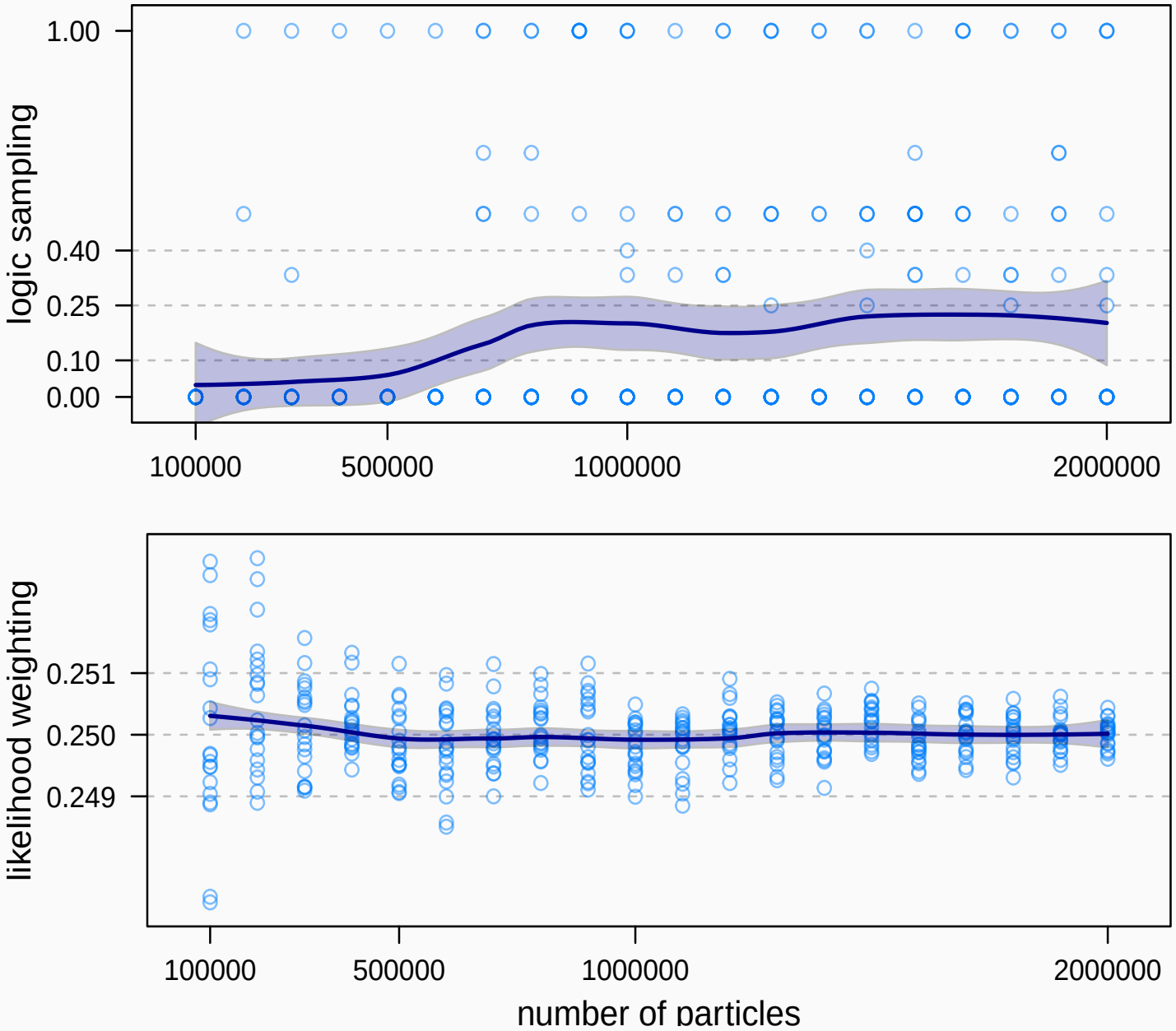
```
[1] 0.333
```

This simply does not happen with likelihood weighting.

```
cpquery(extreme.bn, event = (B == "b2"), evidence = list(A = "a2"),
  method = "lw", n = 5 * 10^3)
```

```
[1] 0.249
```

A Comparison for Different Numbers of Random Samples



Extensions of Likelihood Weighting

The `event` is still a general expression, which means it is possible to describe complex events. However, likelihood weighting relies on the fact that the `evidence` is fixed to a single value to compute the weights. In **bnlearn** this assumption is relaxed: the `event` can take more than one value for each variable. All combinations of values are given the same probability so as not to alter the weights.

```
cpquery(bn, event = (S == "M") & (T == "car"),
  evidence = list(A = c("young", "adult")), method = "lw", n = 10^6)
```

```
[1] 0.337
```

```
cpquery(bn, event = (S == "M") & (T == "car"),
  evidence = list(A = "young"), method = "lw", n = 10^6) * 0.5 +
cpquery(bn, event = (S == "M") & (T == "car"),
  evidence = list(A = "adult"), method = "lw", n = 10^6) * 0.5
```

```
[1] 0.337
```

Sampling and Conditioning

Last but not least, we can also use `cpdist()` to generate random samples conditional on some evidence E. Likelihood weighting works best, and attaches the weights to the samples (for use in later analyses).

```
cpdist(bn, nodes = c("S", "T"), evidence = list(A = "adult"),
  method = "lw", n = 5)
```

```
  S   T
1 M car
2 F car
3 F car
4 M car
5 F car
```

Logic sampling works less well because it often returns far fewer observations than requested.

```
cpdist(bn, nodes = c("S", "T"), evidence = (A == "young"),
  method = "ls", n = 5)
```

```
[1] S T
<0 rows> (or 0-length row.names)
```

The Junction Tree Algorithm

1. **Moralise:** create the moral graph of the BN $\mathcal{B}$.
2. **Triangulate:** break every cycle spanning 4 or more nodes into sub-cycles of exactly 3 nodes by adding arcs to the moral graph, thus obtaining a triangulated graph.
3. **Cliques:** identify the cliques $C_1, \ldots, C_k$ of the triangulated graph, i.e., maximal subsets of nodes in which each element is adjacent to all the others.
4. **Junction Tree:** create a tree in which each clique is a node, and adjacent cliques are linked by arcs. The tree must satisfy the running intersection property: if a node belongs to two cliques C_i and C_j, it must be also included in all the cliques in the (unique) path that connects C_i and C_j.
5. **Parameters:** use the parameters of the local distributions of $\mathcal{B}$ to compute the parameter sets of the compound nodes of the junction tree.

Creating the Moral Graph

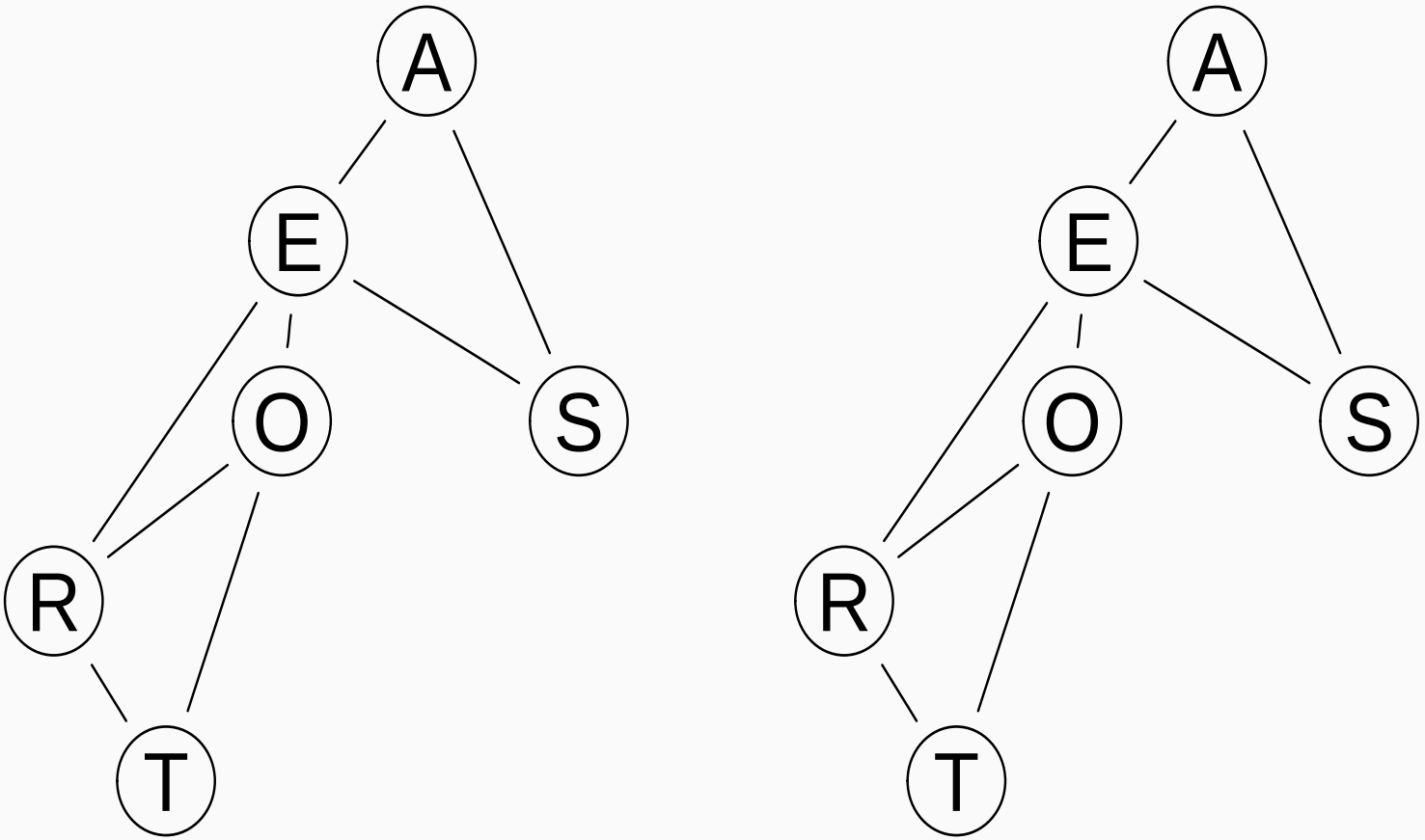
We saw how to create a moral graph earlier when introducing d-separation:

```
survey.dag = model2network("[A][S][E|A:S][O|E][R|E][T|O:R]")
survey.moral = moral(survey.dag)
```

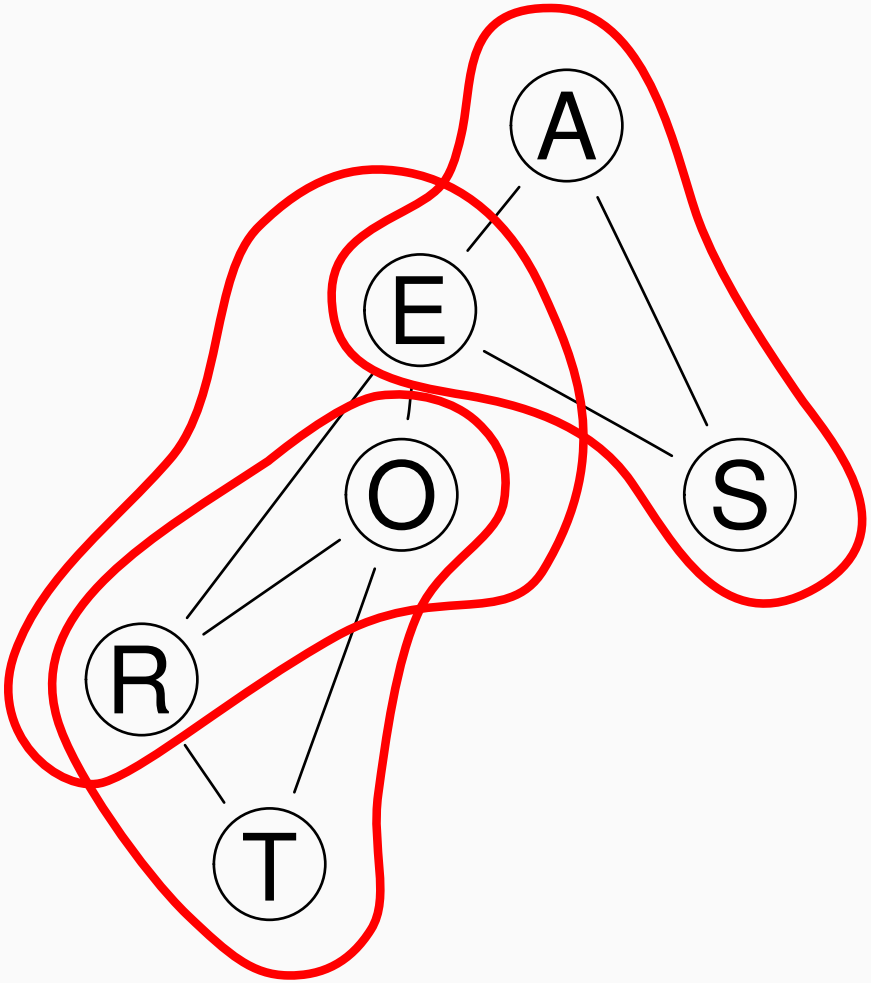
NOTE: different DAGs can express the same set of dependencies and therefore will have the same moral graph. This in turn means that exact inference with junction trees will return the same results for conditional probability and maximum a posteriori queries. They are probabilistically indistinguishable.

Different DAGs, Same Moral Graph

```
survey.dag1 = model2network("[A][S][E|A:S][O|E][R|E][T|O:R]")
survey.dag2 = model2network("[A|E][S|A:E][E|O:R][O|R:T][R|T][T]")
graph.par(list(nodes = list(fontsize = 11)))
par(mfrow = c(1, 2))
graphviz.plot(moral(survey.dag1))
graphviz.plot(moral(survey.dag2))
```



Finding the Cliques



The moral graph is already triangulated, and we can see three cliques:

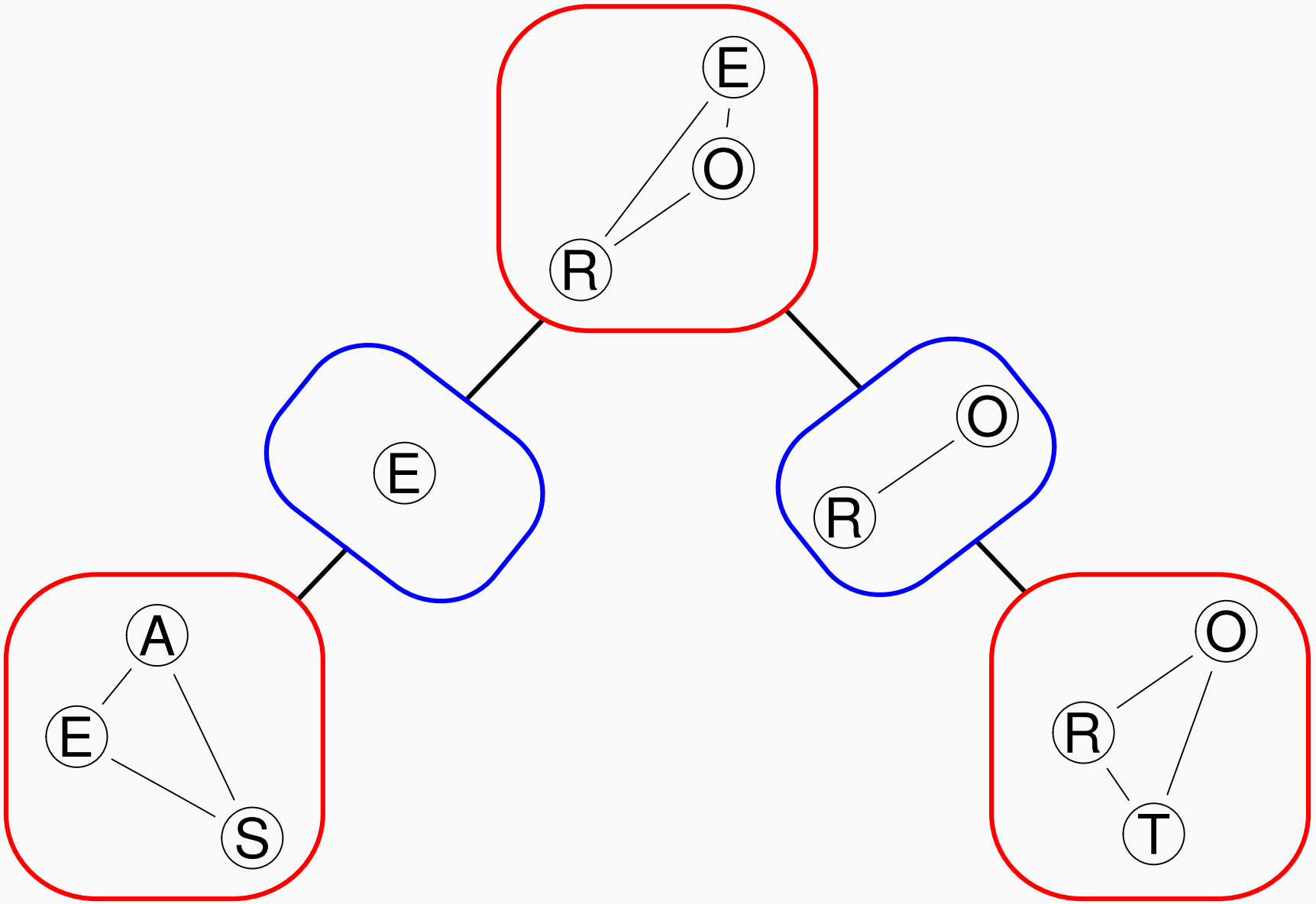
$$C_1 = \{A, E, S\}$$
$$C_2 = \{E, O, R\}$$
$$C_3 = \{O, R, T\}$$

with separators:

$$S_{12} = \{E\}$$
$$S_{23} = \{O, R\}$$

which we can use to build the junction tree.

Building the Junction Tree



In this example on the survey BN, the parameters for the cliques are:

$$\Theta_{C_1} = \mathrm{P}(A, E, S) = \mathrm{P}(A)\,\mathrm{P}(S)\,\mathrm{P}(E \,|\, A, S)$$
$$\Theta_{C_2} = \mathrm{P}(E, O, R) = \mathrm{P}(O \,|\, E)\,\mathrm{P}(R \,|\, E)\,\mathrm{P}(E)$$
$$\Theta_{C_3} = \mathrm{P}(O, R, T) = \mathrm{P}(T|\, O, R)\,\mathrm{P}(O), \mathrm{P}(R)$$

and those for the separators are:

$$\Theta_{S_{12}} = \mathrm{P}(E)$$
$$\Theta_{S_{23}} = \mathrm{P}(O, R)$$

All can be readily computed from the local distributions in the BN.

Estimating the Parameters

```
C1 = coef(bn$E)
for (a in A.lv)
  for (s in S.lv)
    C1[, a, s] = C1[, A = a, S = s] * coef(bn$A)[a] * coef(bn$S)[s]
C1
```

```
, , S = M

      A
E       young  adult    old
  high 0.1350 0.2160 0.1056
  uni  0.0450 0.0840 0.0144

, , S = F

      A
E       young  adult    old
  high 0.0768 0.1400 0.0720
  uni  0.0432 0.0600 0.0080
```

```
S12 = margin.table(C1, 1)
S12
```

```
E
 high   uni
0.745 0.255
```

Estimating the Parameters

```
C2 = array(0, dim = c(2, 2, 2), dimnames = list(O = O.lv, R = R.lv, E = E.lv))
for (o in O.lv)
  for (r in R.lv)
    for (e in E.lv)
      C2[o, r, e] = coef(bn$O)[o, e] * coef(bn$R)[r, e] * S12[e]
C2
```

```
, , E = high

      R
O        small    big
  emp  0.17890 0.5367
  self 0.00745 0.0224

, , E = uni

      R
O        small    big
  emp  0.04685 0.1874
  self 0.00407 0.0163
```

Estimating the Parameters

```
S23 = margin.table(C2, 1:2)
S23
```

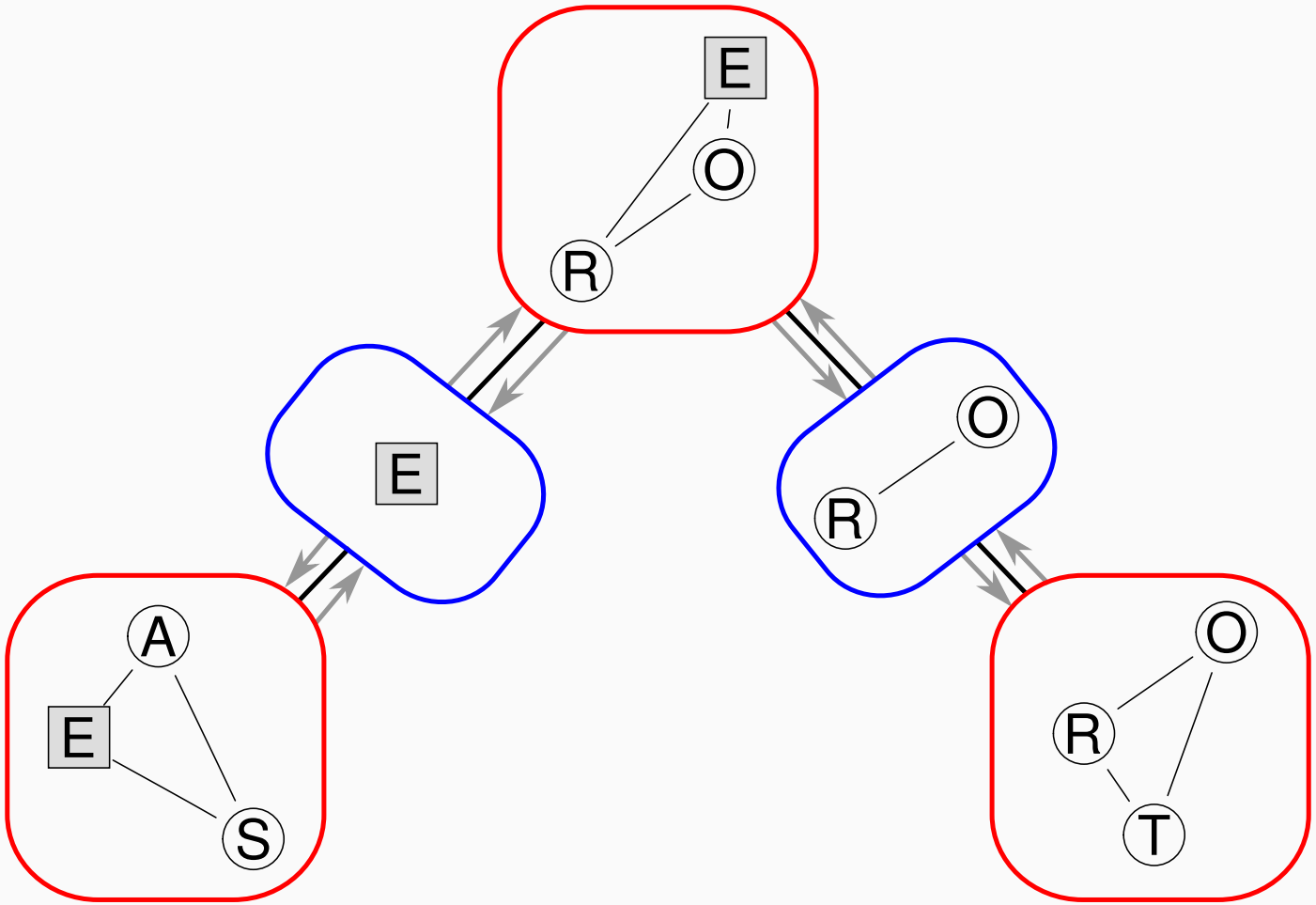
```
        R
 O        small    big
   emp  0.2257 0.7241
   self 0.0115 0.0387
```

```
C3 = coef(bn$T)
for (t in T.lv)
  for (o in O.lv)
    for (r in R.lv)
      C3[t, o, r] = C3[t, o, r] *
                    S23[o, r]
```

```
C3
```

```
, , R = small

       O
T             emp     self
  car    0.108356 0.006455
  train  0.094812 0.004150
  other  0.022574 0.000922

, , R = big

       O
T             emp     self
  car    0.419963 0.027059
  train  0.173778 0.008118
  other  0.130333 0.003479
```

Belief Propagation and Message Passing



Say we set Education to “high school”: we can change it directly in S_{12}, but then we need to propagate the changes to C_1 and C_2; and from C_2 to S_{23} and to C_3. This is called belief propagation by message passing.

Belief Propagation and Message Passing

```
new.S12 = S12
new.S12["high"] = 1
new.S12["uni"] = 0
new.S12
```

```
 high  uni
    1    0
```

```
new.C1 = C1
for (e in E.lv)
  for (a in A.lv)
    for (s in S.lv)
      new.C1[e, a, s] =
        C1[e, a, s] / S12[e] *
                    new.S12[e]
```

```
new.C1
```

```
, , S = M

       A
E        young  adult    old
  high 0.1811 0.2898 0.1417
  uni  0.0000 0.0000 0.0000

, , S = F

       A
E        young  adult    old
  high 0.1030 0.1878 0.0966
  uni  0.0000 0.0000 0.0000
```

```
margin.table(new.C1, 1)
```

```
E
high  uni
   1    0
```

`margin.table(new.C1)` and `new.S12` match as expected.

Belief Propagation and Message Passing

```
new.C2 = C2
for (o in O.lv)
  for (r in R.lv)
    for (e in E.lv)
      new.C2[o, r, e] =
        C2[o, r, e] / S12[e] *
                    new.S12[e]
new.C2
```

```
, , E = high

       R
O        small  big
  emp    0.24 0.72
  self   0.01 0.03

, , E = uni

       R
O        small big
  emp        0   0
  self       0   0
```

```
new.S23 = margin.table(new.C2, 1:2)
new.S23
```

```
         R
 O        small  big
   emp    0.24 0.72
   self   0.01 0.03
```

```
new.C3 = C3
for (t in T.lv)
  for (o in O.lv)
    for (r in R.lv)
      new.C3[t, o, r] =
        C3[t, o, r] / S23[o, r] *
                    new.S23[o, r]
```

Which completes the first iteration of belief propagation.

In more complex graphs and more complex queries we may need more than one iteration, but for this relatively simple network the belief propagation is complete.

Computing $\mathrm{P}(\mathtt{S} = \texttt{"M"}, \mathtt{T} = \texttt{"car"})$ at this point can be done easily by:

```
T = margin.table(new.C3, 1)
S = margin.table(new.C1, 3)
as.numeric(S["M"] * T["car"])
```

```
[1] 0.343
```

because Sex and Transportation are in different cliques and are separated by Education, and therefore independent.

gRain: Exact Inference with Junction Trees

Junction trees and belief propagation are implemented in the **gRain** package. In order to answer our query, we convert the BN from **bnlearn** to its equivalent in **gRain** with `as.grain()` and we construct the junction tree with `compile()`.

```
library(gRain)
junction = compile(as.grain(bn))
```

Then we set the evidence on the node, fixing it to "high school" with probability 1 with `setEvidence()`.

```
jedu = setEvidence(junction, nodes = "E", states = "high")
```

And after that, we can perform our conditional probability query with `querygrain()`, which also takes care of the belief propagation.

```
SxT.cpt = querygrain(jedu, nodes = c("S", "T"), type = "joint")
```

Joint and Marginal Conditional Probabilities

The result of our query is the joint distribution of Sex and Travel given that Education is “high school”.

SxT.cpt

```
    T
S     car train  other
  M 0.343 0.174 0.0962
  F 0.217 0.110 0.0609
```

Similarly, we can use `querygrain()` compute the marginal distributions of Sex and Travel conditional on Education.

```
querygrain(jedu, nodes = c("S", "T"), type = "marginal")
```

```
$S
S
    M     F
0.613 0.387

$T
T
  car train other
0.559 0.283 0.157
```

Interestingly, we can also compute the conditional distribution of Sex given Travel (still conditioning on Education being “high school”), which turns out to be:

```
querygrain(jedu, nodes = c("S", "T"), type = "conditional")
```

```
    T
S      car train other
  M 0.613 0.613 0.613
  F 0.387 0.387 0.387
```

This makes sense in the light of d-separation, which implies conditional independence.

```
dsep(bn, x = "S", y = "T", z = "E")
```

```
[1] TRUE
```

GBNs: Exact and Approximate Inference

Approximate inference works exactly in the same way as for DBNs. Sampling from a linear regression model is easy:

1. plug in the values of the parents;
2. generate the residuals from a normal distribution with mean zero and the specified variance.

The only major difference is that we cannot compute the probability of events that correspond to a point value, because probability is associated with intervals for continuous variables.

Exact inference is much easier than for DBNs. The distribution of some event nodes Q conditional on evidence nodes $E = \mathbf{e}$ has a (multivariate) normal distribution with

$$\widetilde{\boldsymbol{\mu}} = \boldsymbol{\mu}_Q + \Sigma_{QE}\Sigma_{EE}^{-1}(\mathbf{e} - \boldsymbol{\mu}_E) \quad \text{and} \quad \widetilde{\Sigma} = \Sigma_{QQ} - \Sigma_{QE}\Sigma_{EE}^{-1}\Sigma_{EQ}.$$

Use truncated normals in the case of interval evidence.

Marks: Analysis and Algebra

What is the probability that a student will get a distinction mark in algebra after getting a low mark at most in analysis?

```
cpquery(marks.bn, (ALG >= 70), evidence = (ANL <= 50), method = "ls")
```

```
[1] 0.0015
```

```
cpquery(marks.bn, (ALG >= 70), evidence = list(ANL = c(0, 50)), method = "lw")
```

```
[1] 0.00165
```

```
mvn = gbn2mvnorm(marks.bn)
mu.tilde = mvn$mu["ALG"] + mvn$sigma["ANL", "ALG"] / mvn$sigma["ANL", "ANL"] *
  (50 - mvn$mu["ANL"])
sigma.tilde = mvn$sigma["ALG", "ALG"] -
  1/mvn$sigma["ANL", "ANL"] * mvn$sigma["ANL", "ALG"]^2
pnorm(70, mean = mu.tilde, sd = sqrt(sigma.tilde), lower.tail = FALSE)
```

```
[1] 0.00936
```

```
mu.tilde = mvn$mu["ALG"] + mvn$sigma["ANL", "ALG"] / mvn$sigma["ANL", "ANL"] *
  (20 - mvn$mu["ANL"])
pnorm(70, mean = mu.tilde, sd = sqrt(sigma.tilde), lower.tail = FALSE)
```

```
[1] 0.00000614
```

CGBNs: Exact and Approximate Inference

Approximate inference for CGBNs is a combination of that for DBNs and GBNs: all we said earlier applies.

Exact inference, on the other hand, combines the worst of both worlds. We cannot work with the global distribution: like DBNs, it is too large. And we cannot use the junction tree algorithm from above either: there is an adaptation that works with CGBNs in package **BayesNetBP**, but it is slower and more memory intensive.

Rats: Weight Loss Among Females

What are the distributions of weight loss among female rats after one and two weeks?

```
particles =
  cpdist(rats.bn, nodes = c("WL1", "WL2"), evidence = list(SEX = "F"),
    method = "lw", n = 10^6)
summaries = sapply(particles, function(x) c(mean = mean(x), sd = sd(x)))
t(summaries)
```

```
     mean   sd
WL1 9.75 4.14
WL2 8.65 2.79
```

```
library(BayesNetBP)
node.class = sapply(rats.bn, function(ldist) is(ldist, "bn.fit.dnode"))
jtree = Initializer(dag = as.graphNEL(rats.dag), data = rbn(rats.bn, 10^5),
          node.class = node.class, propagate = TRUE)
jtree = AbsorbEvidence(jtree, "SEX", "F")
SummaryMarginals(Marginals(jtree, c("WL1", "WL2")))
```

```
     Mean   SD n
WL1 9.77 4.14 3
WL2 8.66 2.80 3
```

General BNs: Exact and Approximate Inference

Exact inference is impossible: it relies on having closed-form representations of the joint distribution of arbitrary subsets of nodes.

Approximate inference, on the other hand, can take advantage of all the advanced Monte Carlo samples. **rstan** works very well for this.

Reliability: Can We Say We are Reliable?

If I have a somewhat reliable system, what is the probability that my reliability is greater than 99% if I observe 5 failures in the first phase and 20 failure is the second phase?

```
params = list(
  Fp = c(2, 50),
  A1p = 20,
  A2p = 400
)
data = sampling(reliability.bn, algorithm = "Fixed_param", data = params,
        iter = 10^5, seed = 42)
nodes = c("FAILPROB", "ACCESS1", "ACCESS2", "FAILNUM1", "FAILNUM2")
particles = as.data.frame(extract(data)[nodes])
```

```
partE = particles[(particles[, "FAILNUM1"] < 5) && (particles[, "FAILNUM2"] < 20), ]
nE = nrow(partE)
partEQ = partE[partE[, "FAILPROB"] < 0.01, ]
nEQ = nrow(partEQ)
nEQ/nE
```

```
[1] 0.0927
```

Relevant Functions in **bnlearn**

- `as.grain()` to export a fitted BN from **bnlearn** to **gRain**.
- `rbn()` to generate random samples from a BN.
- `cpdist()` to generate random samples from a BN conditional on some evidence.
- `cpquery()` to perform approximate inference with logic sampling and likelihood weighting.

Borrowed from **BayesNetBP**: `Initializer()`, `AbsorbEvidence()`, `Marginals()`.
Borrowed from **gRain**: `compile()`, `setEvidence()`, `querygrain()`.

Summary and Remarks

1. Models in machine learning must be able to decide whether to perform particular actions given evidence on the surrounding environment.
2. The basis of these decisions are the predictions and the conditional probabilities computed after incorporating evidence into the model.
3. In the context of BNs computing these probability is called inference.
4. There are two classes of algorithms to perform inference: approximate and exact algorithms.
5. Approximate algorithms generate random samples to simulate real-world experiments.
6. Exact algorithms automate the mathematical steps we would perform to manipulate the probabilities in the model.

Thanks!

Any questions?

Bayesian Networks in Policy and Society

Day 4: Structure Learning

Marco Scutari

Dalle Molle Institute for
Artificial Intelligence (IDSIA)

August 11, 2022

Learning a Bayesian Networks

Model selection and estimation are collectively known as learning, and are usually performed as a two-step process:

1. structure learning, learning the graph structure from the data;
2. parameter learning, learning the local distributions implied by the graph structure learned in the previous step.

This workflow is implicitly Bayesian: given a data set $\mathcal{D}$ we have

$$\underbrace{\mathrm{P}(\mathcal{M} \mid \mathcal{D})}_{\text{learning}} = \underbrace{\mathrm{P}(\mathcal{G} \mid \mathcal{D})}_{\text{structure learning}} \cdot \underbrace{\mathrm{P}(\Theta \mid \mathcal{G}, \mathcal{D})}_{\text{parameter learning}}$$

and structure learning is done in practice as

$$\mathrm{P}(\mathcal{G} \mid \mathcal{D}) \propto \mathrm{P}(\mathcal{G})\,\mathrm{P}(\mathcal{D} \mid \mathcal{G}) = \mathrm{P}(\mathcal{G}) \int \mathrm{P}(\mathcal{D} \mid \mathcal{G}, \Theta)\,\mathrm{P}(\Theta \mid \mathcal{G}) d\Theta.$$

Local Distributions: Divide and Conquer

Both are computationally hard, but they are still feasible thanks to the decomposition of $\mathbf{X}$ into local distributions. Under some assumptions, we can use local computations and we never need to manipulate more than one at a time.

Structure learning boils down to

$$\begin{aligned} \mathrm{P}(\mathcal{D} \mid \mathcal{G}) &= \int \prod_{i=1}^{N} \left[\mathrm{P}(X_i \mid \Pi_{X_i}, \Theta_{X_i}) \, \mathrm{P}(\Theta_{X_i} \mid \Pi_{X_i}) \right] d\Theta \\ &= \prod_{i=1}^{N} \left[\int \mathrm{P}(X_i \mid \Pi_{X_i}, \Theta_{X_i}) \, \mathrm{P}(\Theta_{X_i} \mid \Pi_{X_i}) d\Theta_{X_i} \right] \end{aligned}$$

and parameter learning boils down to

$$\mathrm{P}(\Theta \mid \mathcal{G}, \mathcal{D}) = \prod_{i=1}^{N} \mathrm{P}(\Theta_{X_i} \mid \Pi_{X_i}, \mathcal{D}).$$

Prior Elicitation versus Data

For both parameter and structure learning, we can rely either on

- eliciting information from experts, drawing on the available prior knowledge on the variables in $\mathbf{X}$;
- using available data and extract the information the contain.

In structure learning, elicitation involves favouring or penalising the inclusion of specific (patterns of) arcs in the DAG. In parameter learning, it means partially or completely specifying the parameters of local distribution, or constraining them in various ways.

There are pros and cons to either approach:

- it maybe difficult to find experts, or it may be difficult to find data, depending on the phenomenon;
- the data may be noisy or may not fit distributional assumptions;
- it is usually difficult for experts to suggest values for the parameters;
- data may be affected by sampling bias, experts may be affected by personal biases.

Assumptions for Structure Learning from Data

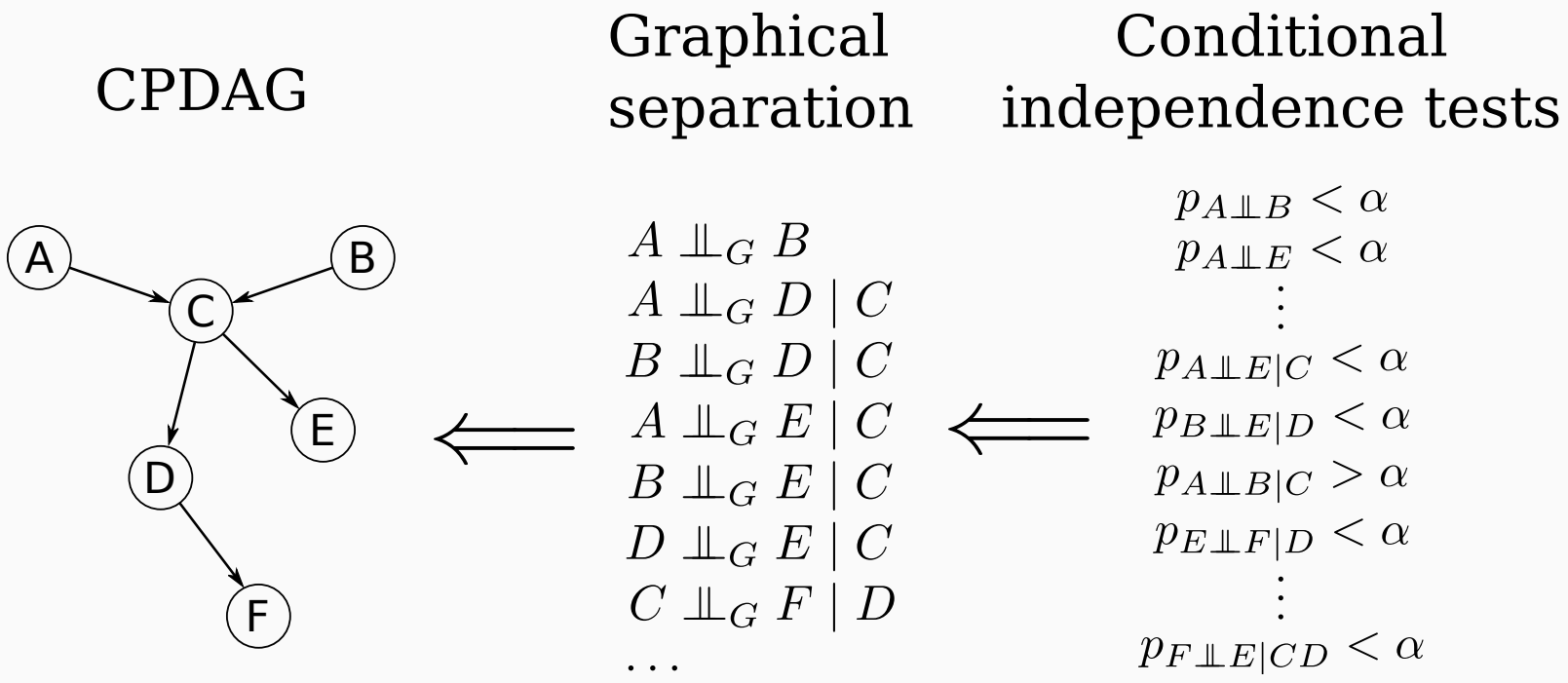
- There must be a one-to-one correspondence between the nodes in the DAG and the random variables in $\mathbf{X}$. There must not be multiple nodes which are deterministic functions of a single variable.
- All the relationships between the variables in $\mathbf{X}$ must be conditional independencies, because they are by definition the only kind of relationships that can be expressed by a BN.
- Every combination of the possible values of the variables in $\mathbf{X}$ must represent a valid, observable (even if really unlikely) event. This assumption implies a strictly positive global distribution, which is needed to a uniquely identifiable model.
- Observations are treated as independent realisations of the set of nodes. If some form of temporal or spatial dependence is present, it must be specifically accounted for in the definition of the network, as in dynamic BNs.

Classes of Structure Learning Algorithms from Data

Despite the (sometimes confusing) variety of theoretical backgrounds and terminology, structure learning algorithms can all be traced to only three approaches:

- Constraint-based algorithms: they use statistical tests to learn conditional independence relationships (called "constraints" in this setting) from the data and assume that the DAG is a perfect map to determine the correct network structure.
- Score-based algorithms: they score each DAG for its goodness of fit, and then find the DAG that maximises that score.
- Hybrid algorithms: conditional independence tests are used to learn at least part of the conditional independence relationships from the data, thus restricting the search space for a subsequent score-based search. The latter determines which edges are actually present in the graph and their direction.

Constraint-Based Structure Learning Algorithms



One way to learn the structure of a BN is to check which conditional independence constraints hold using a suitable conditional independence test. We can identify a single equivalence class in this way.

Assuming a Perfect Map

BNs are defined from graphical separation:

$$\mathbf{A} \perp\!\!\!\perp_G \mathbf{B} \,|\, \mathbf{C} \implies \mathbf{A} \perp\!\!\!\perp_P \mathbf{B} \,|\, \mathbf{C}.$$

However, constraint-based algorithms also imply the reverse:

$$\mathbf{A} \perp\!\!\!\perp_P \mathbf{B} \,|\, \mathbf{C} \iff \mathbf{A} \perp\!\!\!\perp_G \mathbf{B} \,|\, \mathbf{C}.$$

This is a much stronger assumption, which has pros and cons:

- it is impossible to verify;
- but it is a sufficient assumption to uniquely identify Markov blankets, and thus we no longer need to assume that $\mathrm{P}(\mathbf{X})$ is strictly positive everywhere;
- not all $\mathrm{P}(\mathbf{X})$ have a faithful DAG.

The Original: Inductive Causation Algorithm

1. For each pair of variables A and B in $\mathbf{X}$ search for set $\mathbf{S}_{AB} \subset \mathbf{X}$ such that A and B are independent given $\mathbf{S}_{AB}$ and $A, B \notin \mathbf{S}_{AB}$. If there is no such a set, place an undirected arc between A and B.
2. For each pair of non-adjacent variables A and B with a common neighbour C, check whether $C \in \mathbf{S}_{AB}$. If this is not true, set the direction of the arcs $A - C$ and $C - B$ to $A \rightarrow C$ and $C \leftarrow B$.
3. Set the direction of arcs which are still undirected by applying recursively the following two rules:
   - 3.1 if A is adjacent to B and there is a strictly directed path from A to B then set the direction of $A - B$ to $A \rightarrow B$;
   - 3.2 if A and B are not adjacent but $A \rightarrow C$ and $C - B$, then change the latter to $C \rightarrow B$.
4. Return the resulting (partially) directed acyclic graph.

Many newer algorithms: PC, Grow-Shrink, IAMB variants, HITON-PC, HPC.

Conditional Independence Tests: DBNs

Conditional independence tests for $X \perp\!\!\!\perp_P Y | \mathbf{Z}$ are functions of the observed frequencies $\{n_{ijk}, i = 1, \dots, R; j = 1, \dots, C; k = 1, \dots, L\}$.

Classic choices are:

- mutual information/log-likelihood ratio

$$\mathrm{MI}(X, Y | \mathbf{Z}) = \sum_{i=1}^{R} \sum_{j=1}^{C} \sum_{k=1}^{L} \frac{n_{ijk}}{n} \log \frac{n_{ijk} n_{++k}}{n_{i+k} n_{+jk}};$$

- and Pearson's X^2 with a χ^2 distribution

$$\mathrm{X}^2(X, Y | \mathbf{Z}) = \sum_{i=1}^{R} \sum_{j=1}^{C} \sum_{k=1}^{L} \frac{\left(n_{ijk} - m_{ijk}\right)^2}{m_{ijk}}, \quad m_{ijk} = \frac{n_{i+k} n_{+jk}}{n_{++k}}.$$

Both have an asymptotic $\chi^2_{(R-1)(C-1)L}$ null distribution.

Conditional independence tests are functions of the partial correlations $\rho_{XY|\,\mathbf{Z}}$ computed from $\Omega = \Sigma^{-1}_{\{X,Y,\mathbf{Z}\}}$.

Classic choices are:

- the exact t test for Pearson's correlation coefficient, defined as

$$\mathrm{t}(X, Y|\,\mathbf{Z}) = \rho_{XY|\,\mathbf{Z}} \sqrt{\frac{n - |\mathbf{Z}| - 2}{1 - \rho^2_{XY|\,\mathbf{Z}}}} \sim t_{n-|\mathbf{Z}|-2};$$

- the asymptotic Fisher's Z test, defined as

$$\mathrm{Z}(X, Y|\,\mathbf{Z}) = \log\left(\frac{1 + \rho_{XY|\,\mathbf{Z}}}{1 - \rho_{XY|\,\mathbf{Z}}}\right) \frac{\sqrt{n - |\mathbf{Z}| - 3}}{2} \sim N(0, 1)$$

  where n is the number of observations and $|\mathbf{Z}|$ is the number of variables in $\mathbf{Z}$.

Conditional Independence Tests: CGBNs (I)

It is more complicated to specify tests for CLGBNs. Going case by case:

- if X, Y and $\mathbf{Z}$ are all categorical, we can use any test for DBNs;
- if X, Y and $\mathbf{Z}$ are all Gaussian, we can use any test for GBNs;
- if X is categorical and Y is Gaussian (or vice versa), the simple test to use is the mutual information

$$\propto \log \frac{\mathrm{P}(Y|\, X, \mathbf{Z})}{\mathrm{P}(Y|\, \mathbf{Z})}$$

  in which both the numerator and the nominator are linear regressions;
- the same is true if X and Y are Gaussian, regardless of $\mathbf{Z}$ the simple test is again the mutual information.

Conditional Independence Tests: Conditional Gaussian (II)

- if X and Y are categorical, and $\mathbf{Z} = \{Z_{c_1}, \dots, Z_{c_l}, Z_{d_1}, \dots, Z_{d_m}\}$ contains both categorical and Gaussian variables, with several applications of Bayes theorem and the chain rule we get

$$\frac{\mathrm{P}(X \mid Z_{d_1:d_m}, Z_{c_1:c_l})}{\mathrm{P}(X \mid Y, Z_{d_1:d_m}, Z_{c_1:c_l})} = \\ = \frac{\prod_{i=1}^{l-1} \mathrm{P}(Z_{c_i} \mid Z_{c_{i+1}:c_l}, X, Z_{d_1:d_m}) \, \mathrm{P}(X, Z_{d_1:d_m})}{\prod_{i=1}^{l-1} \mathrm{P}(Z_{c_i} \mid Z_{c_{i+1}:c_l}, Z_{d_1:d_m}) \, \mathrm{P}(Z_{d_1:d_m})} \times \\ \frac{\prod_{i=1}^{l-1} \mathrm{P}(Z_{c_i} \mid Z_{c_{i+1}:c_l}, X, Y, Z_{d_1:d_m}) \, \mathrm{P}(X, Y, Z_{d_1:d_m})}{\prod_{i=1}^{l-1} \mathrm{P}(Z_{c_i} \mid Z_{c_{i+1}:c_l}, Y, Z_{d_1:d_m}) \, \mathrm{P}(Y, Z_{d_1:d_m})}$$

which is a chain of log-likelihood ratios that can be treated as a mutual information test.

The ASIA Example, Revisited

The `asia` data set is a small synthetic data set from Lauritzen and Spiegelhalter that tries to implement a diagnostic model for lung diseases (tuberculosis, lung cancer or bronchitis) after a visit to Asia.

- `D`: dyspnoea.
- `T`: tuberculosis.
- `L`: lung cancer.
- `B`: bronchitis.
- `A`: visit to Asia.
- `S`: smoking.
- `X`: chest X-ray.
- `E`: tuberculosis versus lung cancer/bronchitis.

```
head(asia)
```

```
   A   S   T  L   B   E   X   D
1 no yes  no no yes  no  no yes
2 no yes  no no  no  no  no  no
3 no  no yes no  no yes yes yes
4 no  no  no no yes  no  no yes
5 no  no  no no  no  no  no yes
6 no yes  no no  no  no  no yes
```

bnlearn: Functions for Constraint-Based Learning

bnlearn implements several constraint-based algorithms, each with its own function: `gs()`, `iamb()`, `mmpc()`, `si.hiton.pc()`, etc.

```
cpdag = si.hiton.pc(asia, undirected = FALSE)
cpdag
```

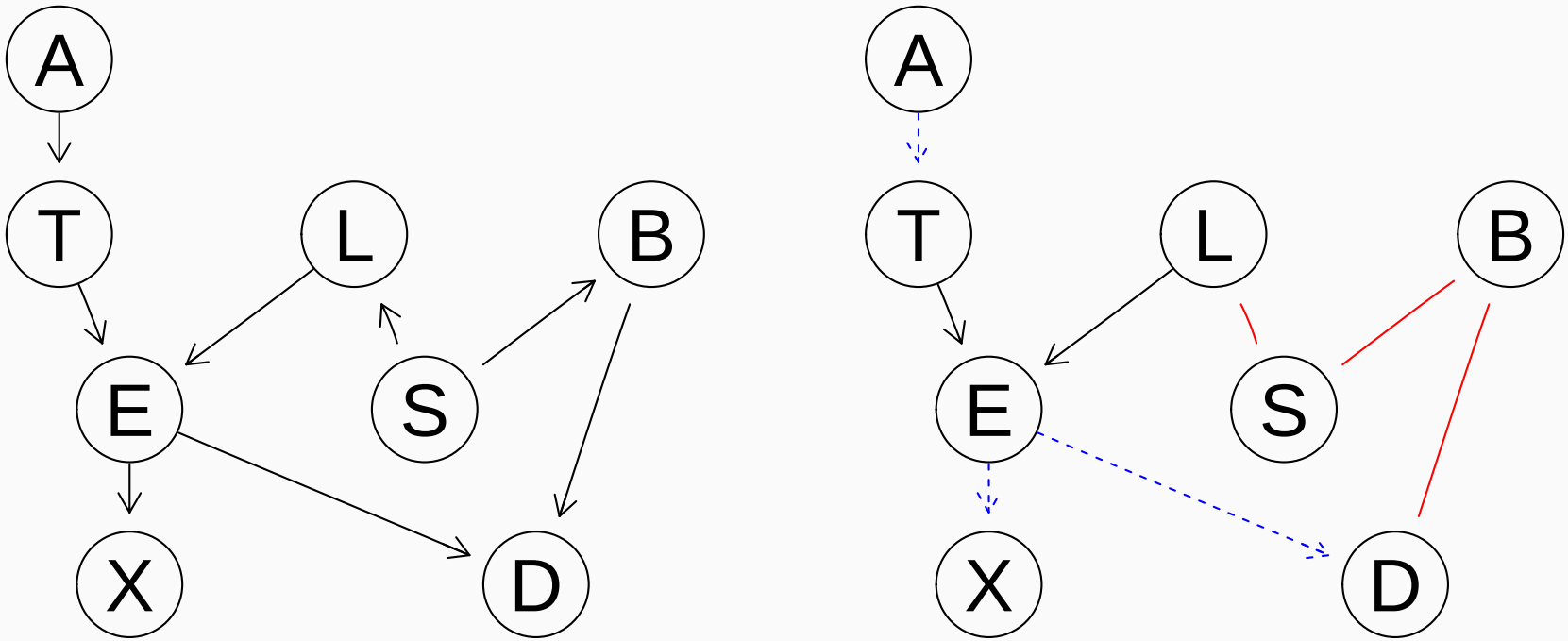
```
  Bayesian network learned via Constraint-based methods

  model:
    [partially directed graph]
  nodes:                                 8
  arcs:                                  5
    undirected arcs:                     3
    directed arcs:                       2
  average markov blanket size:           1.50
  average neighbourhood size:            1.25
  average branching factor:              0.25

  learning algorithm:
                                  Semi-Interleaved HITON-PC
  conditional independence test:
                                 Mutual Information (disc.)
  alpha threshold:                       0.05
  tests used in the learning procedure:  152
```

The arguments for the tuning parameters of constraint-based learning algorithms have the same names in the respective functions:

- the first argument is the data.
- `cluster`: a cluster object from the **parallel** package to perform steps in parallel for different nodes.
- `test`: the label of the test statistic.
- `alpha`: the type-I error threshold for the individual conditional independence tests (i.e. without any multiplicity adjustment).
- `skeleton`: whether to learn just the skeleton instead of the CPDAG.
- `debug`: whether to print out the steps performed by the algorithm.

BNLEARN: Comparing DAGs

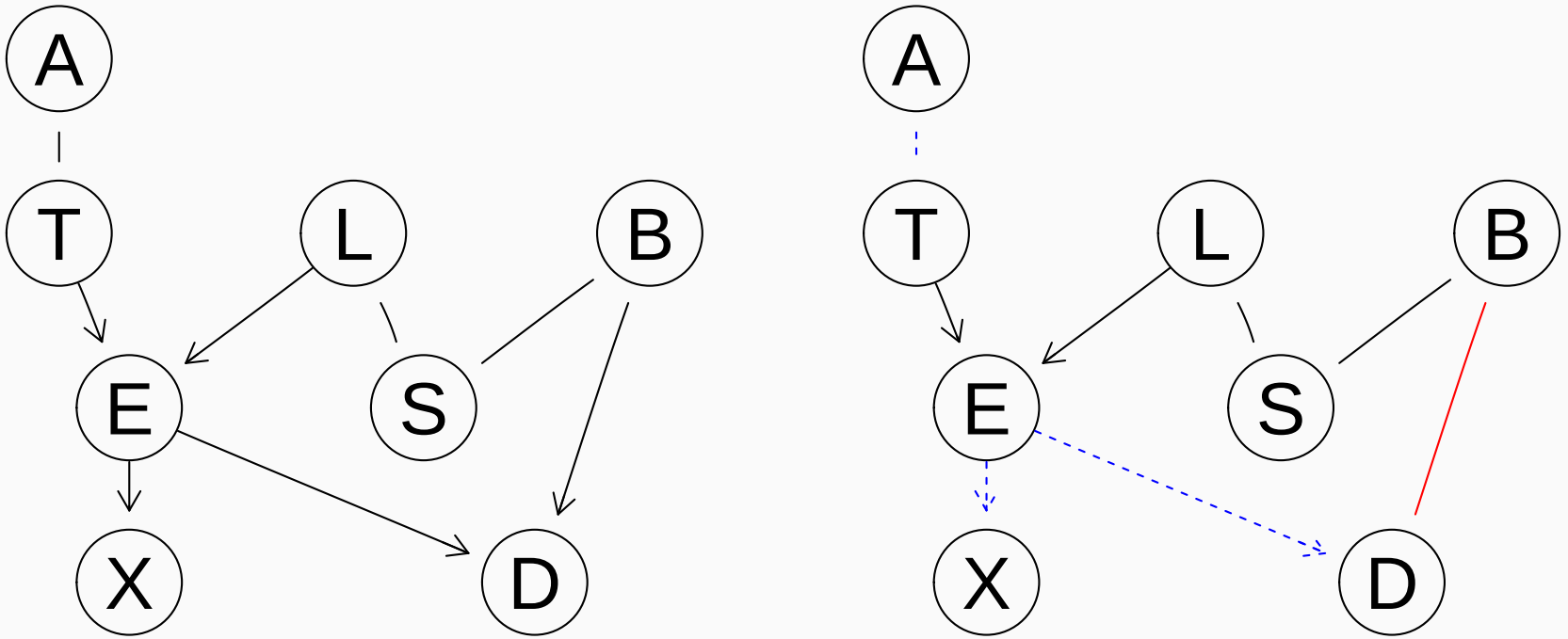
```
asia.dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
cpdag2 = si.hiton.pc(asia, test = "x2", undirected = FALSE)
par(mfrow = c(1, 2))
graph.par(list(nodes = list(fontsize = 10)))
graphviz.compare(asia.dag, cpdag2)
```



Is it really as bad as it looks?

Always Compare CPDAGs

```
par(mfrow = c(1, 2))
graph.par(list(nodes = list(fontsize = 10)))
graphviz.compare(cpdag(asia.dag), cpdag2)
```



It is impossible to uniquely identify the direction of some arcs: `cpdag()` makes that apparent and allows for a fair comparison.

bnlearn: The Debugging Output (I)

```
debugging.output = capture.output(
  si.hiton.pc(asia, test = "mc-mi", undirected = FALSE, debug = TRUE)
)
head(debugging.output, n = 17)
```

```
 [1] "----------------------------------------------------------------"
 [2] "* forward phase for node A ."
 [3] "  * checking nodes for association."
 [4] "  > starting with neighbourhood '  '."
 [5] "  * nodes that are still candidates for inclusion."
 [6] "    > T has p-value 0.0472 ."
 [7] "  * nodes that will be disregarded from now on."
 [8] "    > S has p-value 0.171 ."
 [9] "    > L has p-value 0.516 ."
[10] "    > B has p-value 0.0898 ."
[11] "    > E has p-value 0.132 ."
[12] "    > X has p-value 0.225 ."
[13] "    > D has p-value 0.118 ."
[14] "  @ T accepted as a parent/children candidate ( p-value: 0.0472 )."
[15] "  > current candidates are ' T '."
[16] "----------------------------------------------------------------"
[17] "* forward phase for node S ."
```

bnlearn: The Debugging Output (II)

The debugging output is useful to understand the steps the algorithms perform and to investigate where things go wrong.

```
head(grep("phase", debugging.output, value = TRUE), n = 15)
```

```
 [1] "* forward phase for node A ."
 [2] "* forward phase for node S ."
 [3] "* backward phase for candidate node B ."
 [4] "* backward phase for candidate node E ."
 [5] "* backward phase for candidate node X ."
 [6] "* backward phase for candidate node D ."
 [7] "* forward phase for node T ."
 [8] "* backward phase for candidate node X ."
 [9] "* backward phase for candidate node D ."
[10] "* forward phase for node L ."
[11] "* backward phase for candidate node B ."
[12] "* backward phase for candidate node E ."
[13] "* backward phase for candidate node X ."
[14] "* backward phase for candidate node D ."
[15] "* forward phase for node B ."
```

bnlearn: The Debugging Output (III)

```
head(grep("phase|accepted", debugging.output, value = TRUE), n = 20)
```

```
 [1] "* forward phase for node A ."
 [2] "  @ T accepted as a parent/children candidate ( p-value: 0.0472 )."
 [3] "* forward phase for node S ."
 [4] "  @ L accepted as a parent/children candidate ( p-value: 0 )."
 [5] "* backward phase for candidate node B ."
 [6] "  @ B accepted as a parent/children candidate ( p-value: 0 )."
 [7] "* backward phase for candidate node E ."
 [8] "* backward phase for candidate node X ."
 [9] "* backward phase for candidate node D ."
[10] "* forward phase for node T ."
[11] "  @ E accepted as a parent/children candidate ( p-value: 0 )."
[12] "* backward phase for candidate node X ."
[13] "* backward phase for candidate node D ."
[14] "* forward phase for node L ."
[15] "  @ S accepted as a parent/children candidate ( p-value: 0 )."
[16] "* backward phase for candidate node B ."
[17] "* backward phase for candidate node E ."
[18] "  @ E accepted as a parent/children candidate ( p-value: 0 )."
[19] "* backward phase for candidate node X ."
[20] "* backward phase for candidate node D ."
```

In **bnlearn** we can manually reproduce all the steps performed by constraint-based algorithms, either for debugging purposes or for developing new algorithms.

- We can learn the neighbours of a particular node with any algorithm that learns parents and children (HITON and MMPC).

```
learn.nbr(asia, node = "L", method = "si.hiton.pc", test = "mc-mi")
  [1] "S" "E"
```

- We can learn the Markov blanket of a particular node with any algorithm designed to do that (GS and the IAMB variants).

```
learn.nbr(asia, node = "L", method = "si.hiton.pc", test = "mc-mi")
  [1] "S" "E"
```

BNLEARN: CONDITIONAL INDEPENDENCE TESTS

Another very useful function is `ci.test()`, which performs a single marginal or conditional independence test using the same backends as constraint-based algorithms.

```
options(width = 70)
```

```
ci.test(x = "S", y = "E", z = "L", data = asia, test = "mc-mi")
```

```
        Mutual Information (disc., MC)

data:  S ~ E | L
mc-mi = 0.000004, Monte Carlo samples = 5000, p-value = 1
alternative hypothesis: true value is greater than 0
```

Arguments are much the same as before: `test` specifies the test label, `B` the number of permutations. The test is for x $\perp\!\!\!\perp_P$ y | z where z can be either absent (for marginal tests) or a vector of labels (to condition on one or more variables).

Pros and Cons of Constraint-based Algorithms

- They depend heavily on the quality of the conditional independence tests they use: all proofs of correctness assume tests are always right.
  - Asymptotic tests may make algorithms under-perform.
  - Permutation tests are often too slow, but can be made better with sequential permutations and semi-parametric permutations.
  - Shrinkage tests work better than asymptotic test, but not by much.
- They are consistent, but converge may be slow.
- At any single time they evaluate a small subset of variables, which makes them very memory efficient.
- They do not require multiple testing adjustment, they are self-adjusting (nobody knows why exactly, though).
- They are embarrassingly parallel, so they scale extremely well.

An exhaustive search unfeasible in practice, regardless of the goodness-of-fit measure (called network score) used in the process. However, we can use heuristics in combination with decomposable scores

$$\text{Score}(\mathcal{G}) = \sum_{i=1}^{N} \text{Score}(X_i \,|\, \Pi_{X_i})$$

such as

$$\text{BIC}(\mathcal{G}) = \sum_{i=1}^{N} \log \text{P}(X_i \,|\, \Pi_{X_i}) - \frac{|\Theta_{X_i}|}{2} \log n$$

$$\text{BDe}(\mathcal{G}), \text{BGe}(\mathcal{G}) = \sum_{i=1}^{N} \log \left[\int \text{P}(X_i \,|\, \Pi_{X_i}, \Theta_{X_i}) \, \text{P}(\Theta_{X_i} \,|\, \Pi_{X_i}) d\Theta_{X_i} \right]$$

if we only compare BNs that differ in only one local distribution at a time.

The Hill-Climbing Algorithm

1. Choose an initial network structure $\mathcal{G}$, usually (but not necessarily) empty.
2. Compute the score of $\mathcal{G}$, denoted as $Score_{\mathcal{G}} = \text{Score}(\mathcal{G})$.
3. Set $maxscore = Score_{\mathcal{G}}$.
4. Repeat the following steps as long as $maxscore$ increases:
   - 4.1 for every possible arc addition, deletion or reversal not resulting in a cyclic network:
     - 4.1.1 compute the score of the modified network $\mathcal{G}^*$, $Score_{\mathcal{G}^*} = \text{Score}(\mathcal{G}^*)$:
     - 4.1.2 if $Score_{\mathcal{G}^*} > Score_{\mathcal{G}}$, set $\mathcal{G} = \mathcal{G}^*$ and $Score_{\mathcal{G}} = Score_{\mathcal{G}^*}$.
   - 4.2 update $maxscore$ with the new value of $Score_G$.
5. Return the directed acyclic graph $\mathcal{G}$.

Other optimisation algorithms: tabu search, genetics algorithms, linear programming, constrained optimisation and gradient descent.

If the data $\mathcal{D}$ contain no missing values and assuming:

- a Dirichlet conjugate prior ($X_i \,|\, \Pi_{X_i} \sim Mul\,(\Theta_{X_i} \,|\, \Pi_{X_i})$ and $\Theta_{X_i} \,|\, \Pi_{X_i} \sim Dir\,(\alpha_{ijk})$, $\sum_{jk} \alpha_{ijk} = \alpha_i$ the imaginary sample size);
- positivity (all conditional probabilities $\pi_{ijk} > 0$);
- parameter independence (π_{ijk} for different parent configurations) and modularity (same for π_{ijk} in different $\Theta_{X_i} \,|\, \Pi_{X_i}$);

Heckerman et al. derived a closed form expression for $\mathrm{P}(\mathcal{D} \,|\, \mathcal{G})$:

$$\mathrm{BD}(\mathcal{G}, \mathcal{D}; \boldsymbol{\alpha}) = \prod_{i=1}^{N} \mathrm{BD}(X_i, \Pi_{X_i}; \alpha_i) =$$

$$= \prod_{i=1}^{N} \prod_{j=1}^{q_i} \left[\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + n_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk} + n_{ijk})}{\Gamma(\alpha_{ijk})} \right]$$

where r_i is the number of states of X_i; q_i is the number of configurations of Π_{X_i}; $n_{ij} = \sum_k n_{ijk}$; and $\alpha_{ij} = \sum_k \alpha_{ijk}$.

DBNs: Bayesian Dirichlet Equivalent Uniform (BDeu)

The most common BD score assumes $\alpha_{ijk} = \alpha/(r_i q_i)$, $\alpha_i = \boldsymbol{\alpha}$ and is known as BDeu (Bayesian Dirichlet equivalent uniform). The uniform prior over the parameters was justified by the lack of prior knowledge and widely assumed to be non-informative.

However, there is ample evidence that this is a problematic choice:

- The prior is actually not uninformative.
- MAP DAGs selected using BDeu are highly sensitive to the choice of α and can have markedly different number of arcs even for reasonable α.
- In the limits $\alpha \to 0$ and $\alpha \to \infty$ it is possible to obtain both very simple and very complex DAGs, and model comparison may be inconsistent for small $\mathcal{D}$ and small α.
- The sparseness of the MAP network is determined by a complex interaction between α and $\mathcal{D}$.

Better Than BDeu: Bayesian Dirichlet Sparse (BDs)

If the positivity assumption is violated or the sample size n is small, there may be configurations of some Π_{X_i} that are not observed in $\mathcal{D}$.

$$\mathrm{BDeu}(X_i, \Pi_{X_i}; \alpha) =$$

$$= \prod_{j:n_{ij}=0} \left[\frac{\Gamma(r_i\alpha^*)}{\Gamma(r_i\alpha^*)} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha^*)}{\Gamma(\alpha^*)} \right] \prod_{j:n_{ij}>0} \left[\frac{\Gamma(r_i\alpha^*)}{\Gamma(r_i\alpha^* + n_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha^* + n_{ijk})}{\Gamma(\alpha^*)} \right].$$

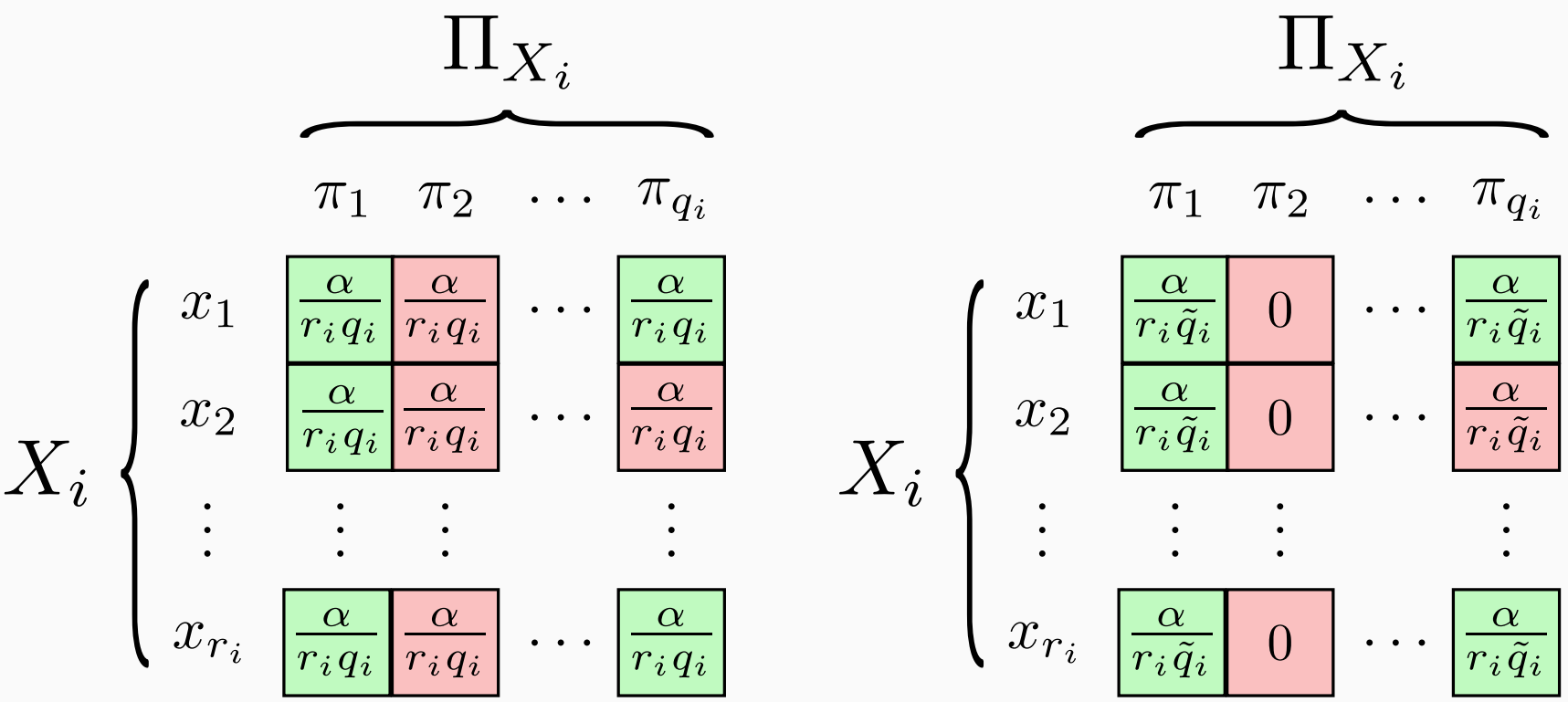
So the effective imaginary sample size decreases as the number of unobserved parents configurations increases, and the MAP estimates of π_{ijk} gradually converge to the ML and favour overfitting.

To address these two undesirable features of BDeu we replace α^* with

$$\tilde{\alpha} = \begin{cases} \alpha/(r_i\tilde{q}_i) & \text{if } n_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}, \quad \tilde{q}_i = \{\text{number of } \Pi_{X_i} \text{ such that } n_{ij} > 0\}$$

and we plug it in BD instead of $\alpha^* = \alpha/(r_i q_i)$ to obtain BDs.

BDeu and BDs Compared



Cells that correspond to $(\mathbf{X}_i, \Pi_{X_i})$ combinations that are not observed in the data are in red, observed combinations are in green.

GBNs: The Bayesian Gaussian Equivalent Score

The Bayesian Gaussian equivalent (BGe) score is defined as the $\mathrm{P}(\mathcal{D} \mid \mathcal{G})$ associated with a normal-Wishart prior $(\boldsymbol{\mu}, W)$ with $\boldsymbol{\mu} \sim N(\boldsymbol{\nu}, \alpha_\mu W)$ and $W \sim Wishart(T, \alpha_w)$:

$$\mathrm{BGe}(X_i, \Pi_{X_i}) = \left(\frac{\alpha_\mu}{N+\alpha_\mu}\right)^{l/2} \frac{\Gamma_l((N+\alpha_w-n+l)/2)}{\pi^{lN/2}\Gamma_l((\alpha_w-n+l)/2)} \frac{|T_{X_i,\Pi_{X_i}}|^{(\alpha_w-n+l)/2}}{|R_{X_i,\Pi_{X_i}}|^{(N+\alpha_w-n+l)/2}}$$

where

$$\Gamma_l\left(\frac{x}{2}\right) = \pi^{l(l-1)/4} \prod_{j=1}^{l} \Gamma\left(\frac{x+1-j}{2}\right),$$

$$R = T + S_N + \frac{N\alpha_w}{N+\alpha_w}(\boldsymbol{\nu} - \bar{\mathbf{x}})(\boldsymbol{\nu} - \bar{\mathbf{x}})^T.$$

(l is defined to be $|X_i \cup \Pi_{X_i}| = |\Pi_{X_i}| + 1$.)

Penalised Likelihoods: AIC and BIC

Penalised likelihoods make for very popular scores. AIC overfits a lot. BIC may under-fit a bit but it is a good default to use. For DBNs, the log-likelihood and the number of parameters associated with a local distribution are:

$$\mathrm{LL}(X_i, \Pi_{X_i}) = \prod_{m=1}^{n} \mathrm{P}(X_i = x_m \,|\, \Pi_{X_i} = \pi_m), \quad |\Theta_{X_i}| = R \times |\Pi_{X_i}|;$$

for GBNs:

$$\mathrm{LL}(X_i, \Pi_{X_i}) = \prod_{m=1}^{n} N(x_m; \boldsymbol{\mu}_{X_i} + \pi_m \boldsymbol{\beta}_{X_i}, \sigma^2_{X_i}), \quad |\Theta_{X_i}| = |\Pi_{X_i}| + 1;$$

for CLGBNS (Δ_{X_i} are the discrete parents, Γ_{X_i} the continuous parents):

$$\mathrm{LL}(X_i, \Pi_{X_i}) = \prod_{m=1}^{n} N(x_m; \boldsymbol{\mu}_{X_i,\delta_m} + \gamma_m \boldsymbol{\beta}_{X_i,\delta_m}, \sigma^2_{X_i,\delta_m}),$$

$$|\Theta_{X_i}| = |\Delta_{X_i}| \times (|\Gamma_{X_i}| + 1).$$

BNLEARN: Hill Climbing with BIC (MARKS)

hc() implements hill-climbing with random restarts, and can use different scores much like functions implementing constraint-based algorithms can use different tests.

```
dag.marks = hc(marks, score = "bic-g")
```

Note that hill-climbing always returns a DAG, not a CPDAG; so the correct way of comparing it with another graph is to take the CPDAG for both.

```
true.dag =
  model2network("[ALG][ANL|ALG][MECH|ALG:VECT][STAT|ALG:ANL][VECT|ALG]")
unlist(compare(dag.marks, true.dag))
```

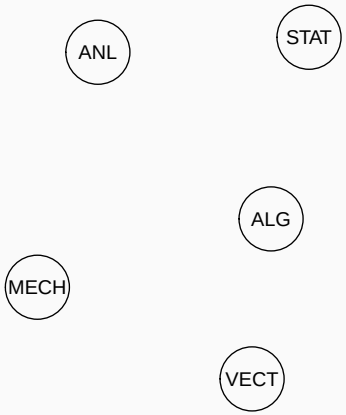
```
 tp fp fn
  3  3  3
```

```
unlist(compare(cpdag(dag.marks), cpdag(true.dag)))
```

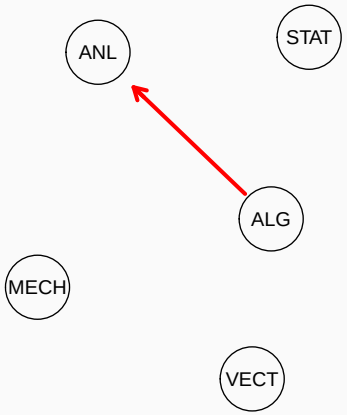
```
 tp fp fn
  6  0  0
```

The Hill-Climbing Algorithm (MARKS)

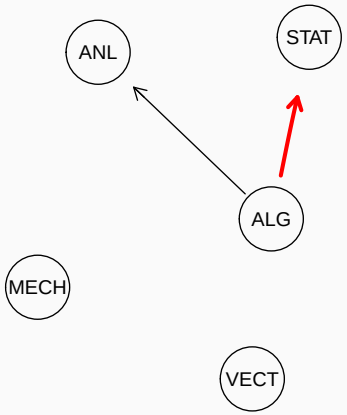
Initial BIC score: −1807.528



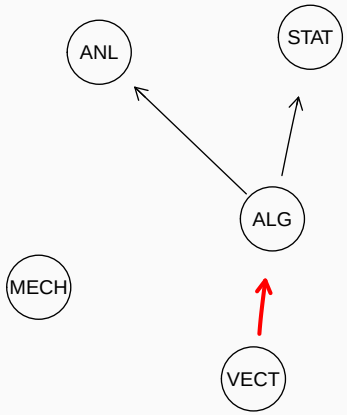
Current BIC score: −1778.804



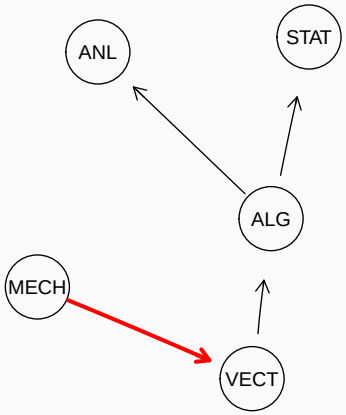
Current BIC score: −1755.383



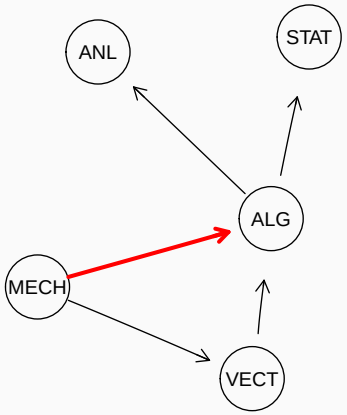
Current BIC score: −1737.176



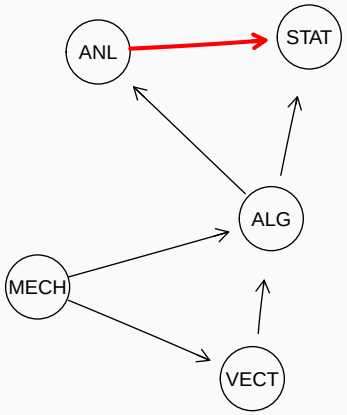
Current BIC score: −1723.325



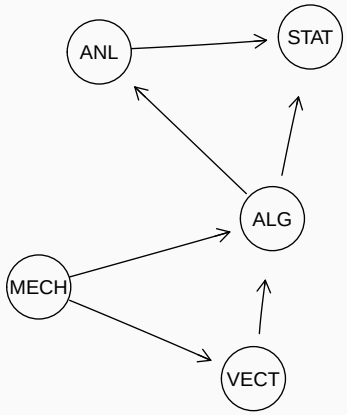
Current BIC score: −1720.901



Current BIC score: −1720.150



Final BIC score: −1720.150



- `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.

```
unlist(compare(dag.marks, true.dag))
```

```
tp fp fn
 3  3  3
```

- `hamming()` computes the Hamming distance between the skeletons of the graphs (zero means a perfect match).

```
hamming(dag.marks, true.dag)
```

```
[1] 0
```

- `shd()` computes the Structural Hamming distance between two CPDAGs, which is similar to the Hamming distance but with a penalty of $1/2$ for directed-undirected arc differences.

```
shd(dag.marks, true.dag)
```

```
[1] 0
```

bnlearn: Hill Climbing with Random Restarts (ASIA)

In addition to scores and their tuning parameters (here `iss` for the imaginary sample size of BDeu), `hc()` has arguments `restart` for the number of random restarts and `perturb` for the number of perturbed arcs in the new starting DAG.

```
asia.restart = hc(asia, score = "bde", iss = 1, restart = 10, perturb = 5)
```

```
debugging.output =
  capture.output(hc(asia, score = "bde", iss = 1, restart = 10,
     perturb = 5, debug = TRUE))
head(grep("^\\* (best|restart)", debugging.output, value = TRUE), n = 10)
```

```
 [1] "* best operation was: adding B -> D ."
 [2] "* best operation was: adding L -> E ."
 [3] "* best operation was: adding E -> X ."
 [4] "* best operation was: adding S -> B ."
 [5] "* best operation was: adding T -> E ."
 [6] "* best operation was: adding E -> D ."
 [7] "* best operation was: adding S -> L ."
 [8] "* best operation was: adding L -> E ."
 [9] "* best operation was: adding E -> X ."
[10] "* best operation was: adding S -> L ."
```

Why Do We Want Random Restarts?

Random restarts reduce the probability of getting stuck in a local maximum by jumping away from it. The DAG we jump to is created by perturbing the DAG that was identified as a local maximum, that is, by changing a number of its arcs to create a new DAG.

```
head(grep("^\\* (current score|doing)", debugging.output, value = TRUE), 14)
```

```
 [1] "* current score: -15225 "
 [2] "* current score: -14043 "
 [3] "* current score: -12955 "
 [4] "* current score: -12026 "
 [5] "* current score: -11579 "
 [6] "* current score: -11348 "
 [7] "* current score: -11217 "
 [8] "* current score: -11096 "
 [9] "* doing a random restart, 9 of 10 left."
[10] "* current score: -12150 "
[11] "* current score: -11220 "
[12] "* current score: -11099 "
[13] "* current score: -11096 "
[14] "* doing a random restart, 8 of 10 left."
```

bnlearn: Hill-Climbing With Preseeded Networks

Another way to avoid getting stuck in local maxima is to start the search from a different network. The default is to start from the empty DAG.

```
capture.output(hc(asia, score = "bde", iss = 1, debug = TRUE))[c(2, 6:7)]
```

```
  [1] "* starting from the following network:"
  [2] "  model:"
  [3] "   [A][S][T][L][B][E][X][D] "
```

However, we can specify an alternative starting DAG with the `start` argument. Here we generate one at random with `random.graph()`.

```
capture.output(hc(asia, score = "bde", iss = 1,
  start = random.graph(names(asia)), debug = TRUE))[c(2, 6:7)]
```

```
  [1] "* starting from the following network:"
  [2] "  model:"
  [3] "   [A][S][T][L|A:S:T][E|A:S][B|S:T:L][X|S:E][D|A:L] "
```

The principle is the same as, say, starting k-means from different sets of centroids and keeping the clustering that fits the data best.

BNLEARN: TABU SEARCH

In addition to `hc()`, **bnlearn** implements `tabu()` with arguments `tabu` (the length of the tabu list) and `max.tabu` (the maximum number of iterations `tabu()` can perform without improving the best network score).

```
debugging.output =
  capture.output(tabu(asia, score = "bde", iss = 1, tabu = 10,
     max.tabu = 5, debug = TRUE))
head(grep("^\\* (best operation|network)", debugging.output, value = TRUE), 10)
```

```
 [1] "* best operation was: adding B -> D ."
 [2] "* best operation was: adding L -> E ."
 [3] "* best operation was: adding E -> X ."
 [4] "* best operation was: adding S -> B ."
 [5] "* best operation was: adding T -> E ."
 [6] "* best operation was: adding E -> D ."
 [7] "* best operation was: adding S -> L ."
 [8] "* network score did not increase (for 1 times), looking for a minimal decreas
 [9] "* best operation was: reversing S -> L ."
[10] "* network score did not increase (for 2 times), looking for a minimal decreas
```

Pros and Cons of Score-based Algorithms

- Convergence to the global maximum (the best structure) is not guaranteed for finite samples, the search may get stuck in a local maximum.
- They are more stable than constraint-based algorithms.
- They require a definition of both the global and the local distributions, and a matching decomposable, network score. This means, for instance, that we cannot use them with ordinal variables because it is difficult to specify the global distribution. On the other hand, there are trend tests to use for conditional independence.
- Most scores have tuning parameters, whereas conditional independence tests (mostly) do not; and algorithms have tuning parameters as well. This usually means a grid of values to be tested under cross-validation to select the optimal learning strategy.

Hybrid Structure Learning Algorithms

Hybrid algorithms combine constraint-based and score-based algorithms to complement the respective strengths and weaknesses; they are considered the state of the art in current literature.

They work by alternating the following two steps:

- learn some conditional independence constraints to restrict the number of candidate networks;
- find the network that maximises some score function and that satisfies those constraints and define a new set of constraints to improve on.

These steps can be repeated several times (until convergence), but one or two times is usually enough.

The Sparse Candidate Algorithm and MMHC

1. Choose a network structure $\mathcal{G}$, usually (but not necessarily) empty.
2. Repeat the following steps until convergence:
   - 2.1 **restrict:** select a set $\mathbf{C}_i$ of candidate parents for each node $X_i \in \mathbf{X}$, which must include the parents of X_i in $\mathcal{G}$;
   - 2.2 **maximise:** find the network structure $\mathcal{G}^*$ that maximises $\mathrm{Score}(\mathcal{G}^*)$ among the networks in which the parents of each node X_i are included in the corresponding set $\mathbf{C}_i$;
   - 2.3 set $\mathcal{G} = \mathcal{G}^*$.
3. Return the directed acyclic graph $\mathcal{G}$.

If we iterate only once, using MMPC for the restrict phase and hill-climbing for the maximise phase we obtain the Max-Min Hill-Climbing (MMHC) algorithm as a particular case.

rsmax2() implements a single step of the Sparse Candidate algorithm: it runs the restrict and maximise phases only once.

```
asia.rsmax2 =
  rsmax2(asia, restrict = "si.hiton.pc",  maximize = "tabu",
    restrict.args = list(test = "x2", alpha = 0.01),
    maximize.args = list(score = "bic", tabu = 10))
```

Its main arguments are:

- restrict: constraint-based algorithm to use in the restrict phase;
- restrict.args: its optional arguments;
- maximize: score-based algorithm to use in the maximise phase;
- maximize.args: its optional arguments.

The following two commands are equivalent:

```
rsmax2(asia, restrict = "mmpc", maximize = "hc")
mmhc(asia)
```

And from the debugging output we can see that is the case:

```
debugging.output = capture.output(print(mmhc(asia, debug = TRUE)))
grep("restrict|maximize|method:", debugging.output, value = TRUE)
```

```
[1] "* restrict phase, using the Max-Min Parent Children algorithm."
[2] "* maximize phase, using the Hill-Climbing algorithm."
[3] "  constraint-based method:                  "
[4] "  score-based method:                       Hill-Climbing "
```

```
debugging.output =
  capture.output(print(rsmax2(asia, restrict = "mmpc", maximize = "hc",
    debug = TRUE)))
grep("restrict|maximize|method:", debugging.output, value = TRUE)
```

```
[1] "* restrict phase, using the Max-Min Parent Children algorithm."
[2] "* maximize phase, using the Hill-Climbing algorithm."
[3] "  constraint-based method:                  "
[4] "  score-based method:                       Hill-Climbing "
```

Pros and Cons of Hybrid Algorithms

- You can mix and match conditional independence tests and network scores with structure learning algorithms, since the latter do not depend on the nature of the data. They can range from frequentist to Bayesian to information-theoretic and anything in between (within reason).
- Constraint-based algorithms are usually faster, score-based algorithms are more stable. Hybrid algorithms are at least as good as score-based algorithms, and often a bit faster.
- Tuning parameters can be difficult to tune for some configurations of algorithms, tests and scores.

A Final Comparison

In this particular case, hill-climbing with random restarts wins the day.

```
true.dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
unlist(compare(cpdag(asia.rsmax2), cpdag(true.dag)))
```

```
  tp fp fn
   4  4  1
```

```
shd(asia.rsmax2, true.dag)
```

```
  [1] 4
```

```
unlist(compare(cpdag(asia.restart), cpdag(true.dag)))
```

```
  tp fp fn
   7  1  0
```

```
shd(asia.restart, true.dag)
```

```
  [1] 1
```

```
unlist(compare(cpdag(cpdag2), cpdag(true.dag)))
```

```
  tp fp fn
   4  4  1
```

```
shd(cpdag2, true.dag)
```

```
  [1] 4
```

The most common choice for $\mathrm{P}(\mathcal{G})$ is the uniform distribution because it is feels like it is uninformative. However, it is problematic because:

- Score-based structure learning algorithms typically generate new candidate DAGs by a single arc additions, deletions or reversals:

$$\frac{\mathrm{P}(\mathcal{G} \cup \{X_j \to X_i\} \mid \mathcal{D})}{\mathrm{P}(\mathcal{G} \mid \mathcal{D})} = \frac{\mathrm{P}(\mathcal{G} \cup \{X_j \to X_i\})}{\mathrm{P}(\mathcal{G})} \frac{\mathrm{P}(\mathcal{D} \mid \mathcal{G} \cup \{X_j \to X_i\})}{\mathrm{P}(\mathcal{D} \mid \mathcal{G})}.$$

  The prior always simplifies, and that implies all operations have a prior probability of $1/3$. The probability that two nodes up connected is then $2/3$.

- Two arcs are correlated in the prior if they are incident on a common node, so false positives and false negatives can potentially propagate through $\mathrm{P}(\mathcal{G})$ and lead to further errors in learning $\mathcal{G}$.

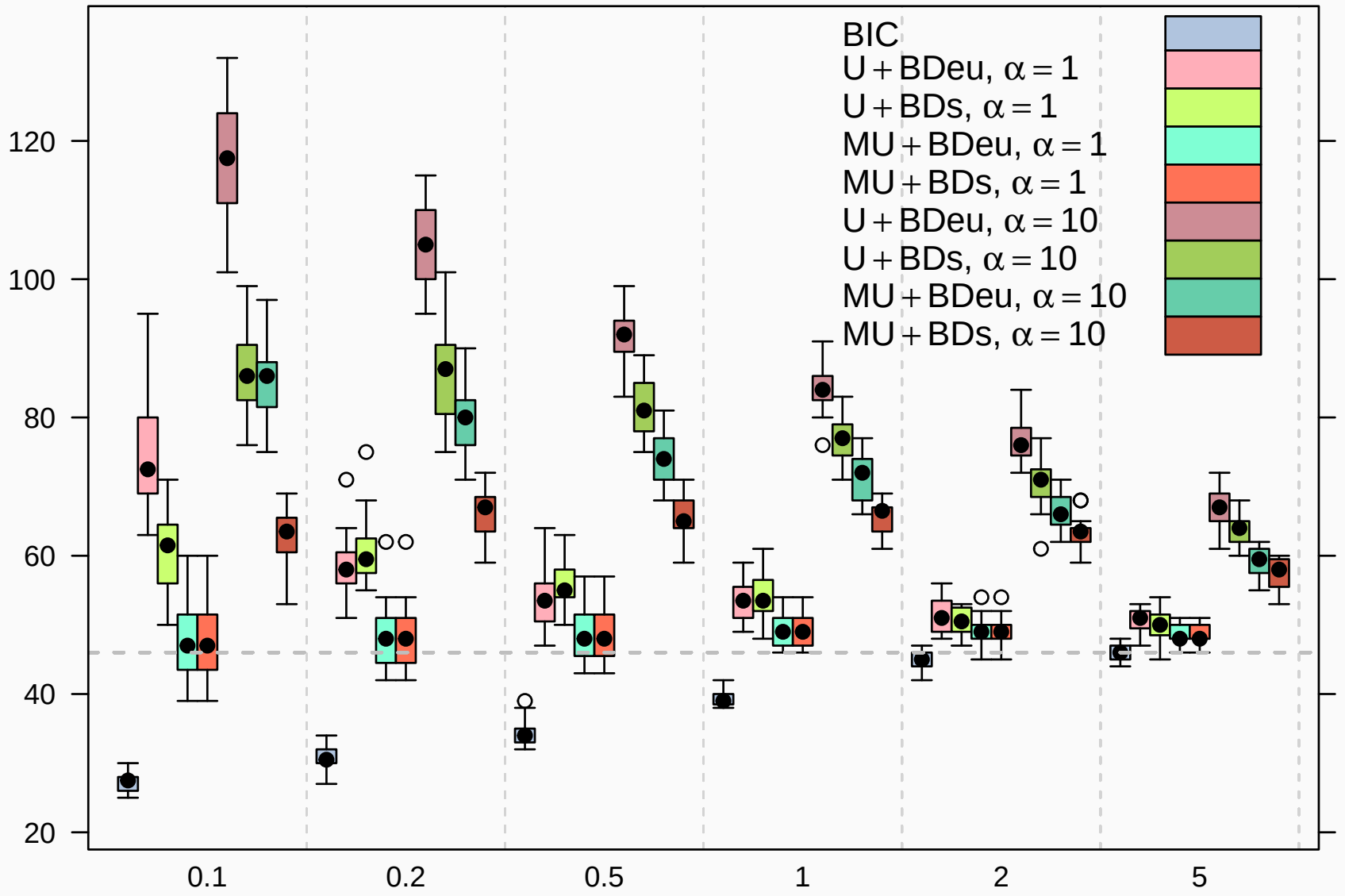
- DAGs that are completely unsupported by the data have most of the probability mass for large enough N.

We want our BNs to be sparse: we should express this fact in $\mathrm{P}(\mathcal{G})$. The simplest option is to use a marginal uniform prior that:

- Does not favour arc inclusion, which should have probability $\leqslant {}^1/_2$.
- Does not favour the propagation of errors in structure learning because arcs are independent from each other.
- Is computationally trivial to use: including an arc with probability $\leqslant {}^1/_4$, its reverse with the same probability, and not including an arc with probability $\geqslant {}^1/_2$.

A DAG can contain $\frac{N(N-1)}{2}$ arcs. We can set the probability of inclusion to $c\frac{2}{N-1}, c \in [1, 3]$ to have $O(cN)$ expected arcs in the prior, which often works even better.

BDe and BDs Scores, Uniform and Marginal Uniform Priors



Informative Graphical Priors

When we have reliable information elicited from experts, we may also want to use an informative prior to force structure learning to include (or not) specific arcs patterns. Some examples:

- limiting the number of parents for some or all nodes;
- setting different probabilities for inclusion for different arcs (the Castelo & Siebes prior);
- whitelisting or blacklisting specific arcs;
- setting the topological ordering of the topological ordering, effectively allowing only one direction for each arc.

All these informative priors concentrate the prior probability mass on DAGs we know from experts to be the most sensible. They can also rule out a large portion of the possible DAGs (giving them a zero probability), reducing the search space by orders of magnitude.

bnlearn: Graphical Priors with Hill-Climbing

```
hc(marks, maxp = 3)
```

```
wl = data.frame(
  from = c("ANL", "ANL"),
  to = c("ALG", "MECH")
)
bl = data.frame(
  from = c("MECH", "MECH"),
  to = c("ALG", "VECT")
)
hc(marks, whitelist = wl, blacklist = wl)
```

```
hc(marks, score = "bge", prior = "marginal", beta = 2 / (ncol(marks) - 1))
beta = data.frame(from = c("MECH", "ALG"), to = c("ALG", "MECH"),
          prob = c(0.2, 0.6))
hc(marks, score = "bge", prior = "cs", beta = beta)
```

```
bl = tiers2blacklist(list(c("ANL", "ALG"), c("VECT", "STAT"), "MECH"))
hc(marks, blacklist = bl)
bl = ordering2blacklist(c("ANL", "ALG", "VECT", "STAT", "MECH"))
hc(marks, blacklist = bl)
```

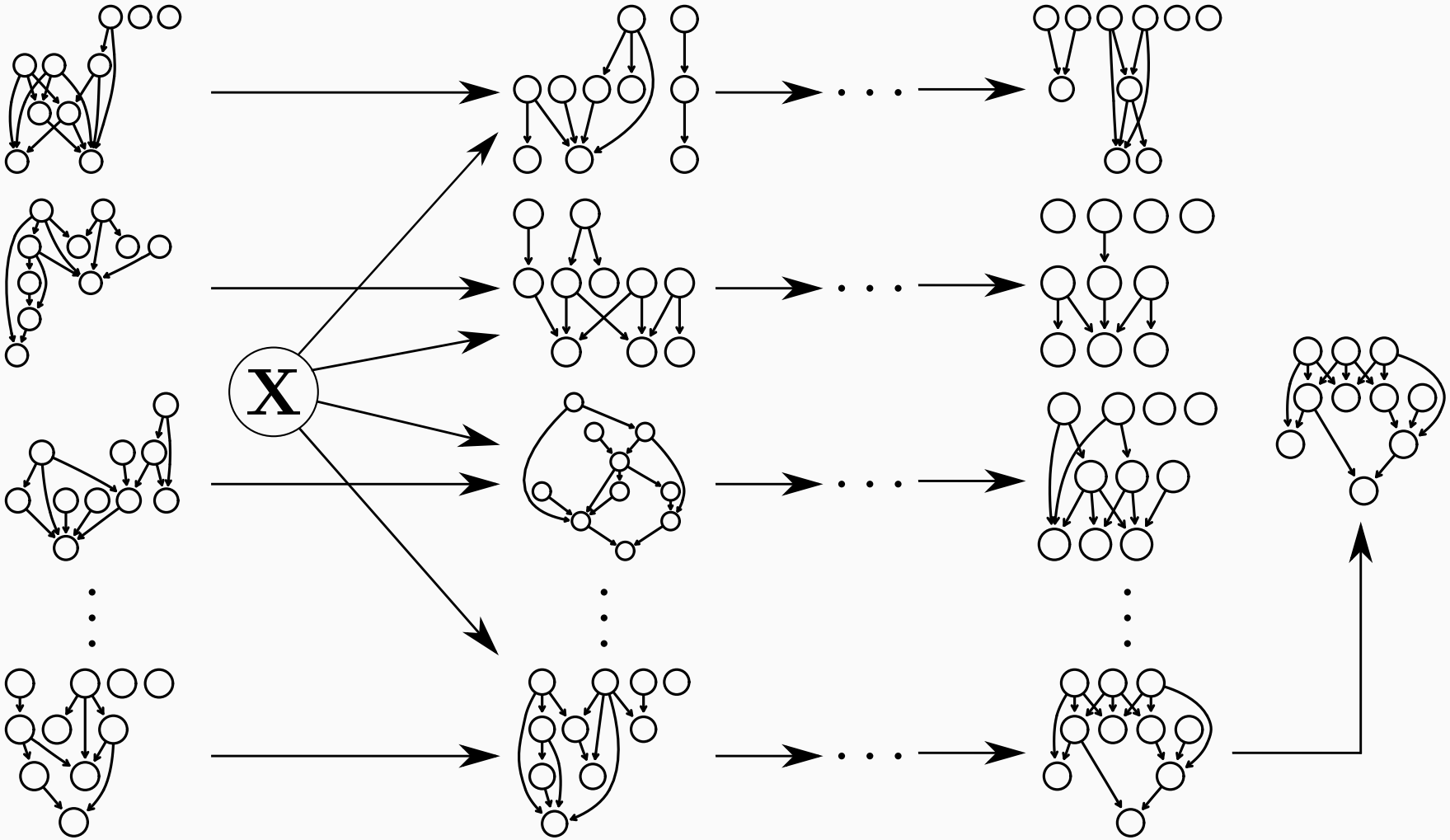
Model Averaging: We Should Always Do That!

Structure learning from limited amounts of data is inherently noisy. Perturbing the data, learning multiple DAGs and then averaging them is most effective in removing that noise. There are two main ways of doing that:

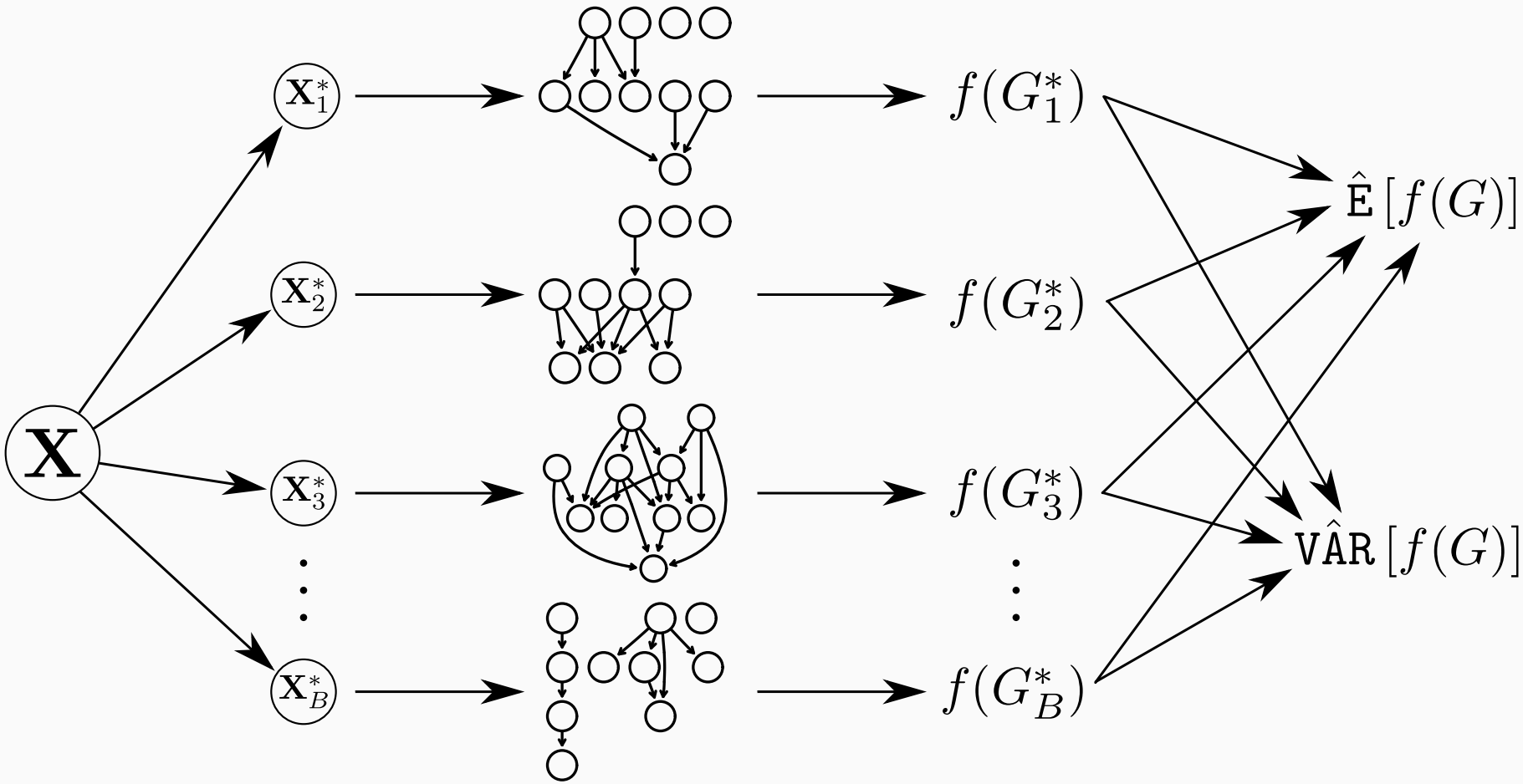
1. Searching from different starting points increases our coverage of the space of the possible DAGs and uses all observations.
2. Learning multiple DAGs from bootstrap samples (that is, bagging) perturbs the data reducing the impact of outliers.
3. Both.

The frequency with which an arc appears is a measure of the strength of the dependence. In other words, it quantifies our confidence that the arc is real. This is separate from the magnitude or the sign of the statistical effect the arcs represents.

Multiple Starting Points and the Same Data



Same Starting Point and Bootstrapped Data



BNLEARN: Model Averaging (I)

```
nodes = names(marks)
start = random.graph(nodes = nodes, method = "ic-dag", num = 500, every = 50)
netlist = lapply(start,
  function(dag) hc(marks, start = dag)
)
str = custom.strength(netlist, nodes = nodes)
head(str)
```

```
    from   to strength direction
1 MECH VECT    1.000       0.5
2 MECH  ALG    1.000       0.5
3 MECH  ANL    0.086       0.5
4 MECH STAT    0.066       0.5
5 VECT MECH    1.000       0.5
6 VECT  ALG    1.000       0.5
```

```
str = boot.strength(marks, algorithm = "hc")
head(str)
```

```
    from   to strength direction
1 MECH VECT    0.830     0.521
2 MECH  ALG    0.825     0.509
3 MECH  ANL    0.095     0.737
4 MECH STAT    0.030     0.583
5 VECT MECH    0.830     0.479
6 VECT  ALG    0.940     0.500
```

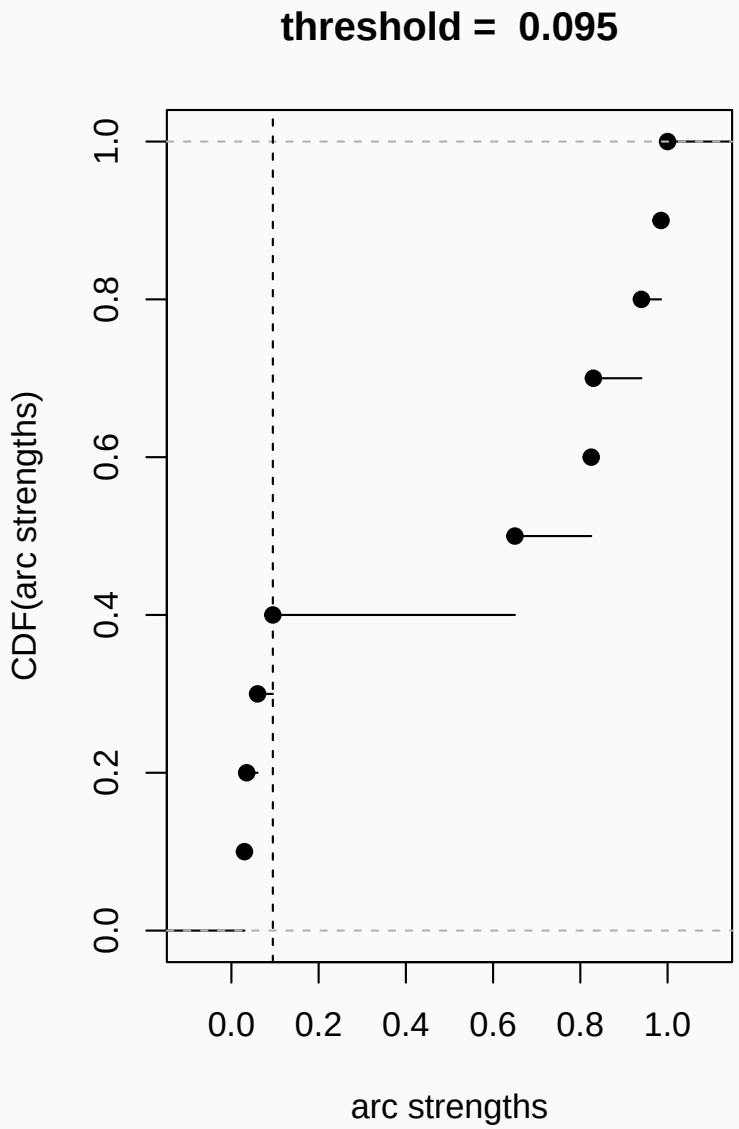
BNLEARN: Model Averaging (II)

We need a threshold to decide which arcs are strong enough to be included in the consensus DAG that is the output of model averaging: we can either estimate it from the data

```
averaged.network(str)
```

or pick one ourselves.

```
averaged.network(str, threshold = 0.90)
```



Who Learns the Best Network Structures

Bayesian network Structure learning is defined by the combination of a statistical criterion and an algorithm that determines how the criterion is applied to the data. What we can say about the algorithms?

Q1 *Which of constraint-based and score-based algorithms provide the most accurate structural reconstruction?*

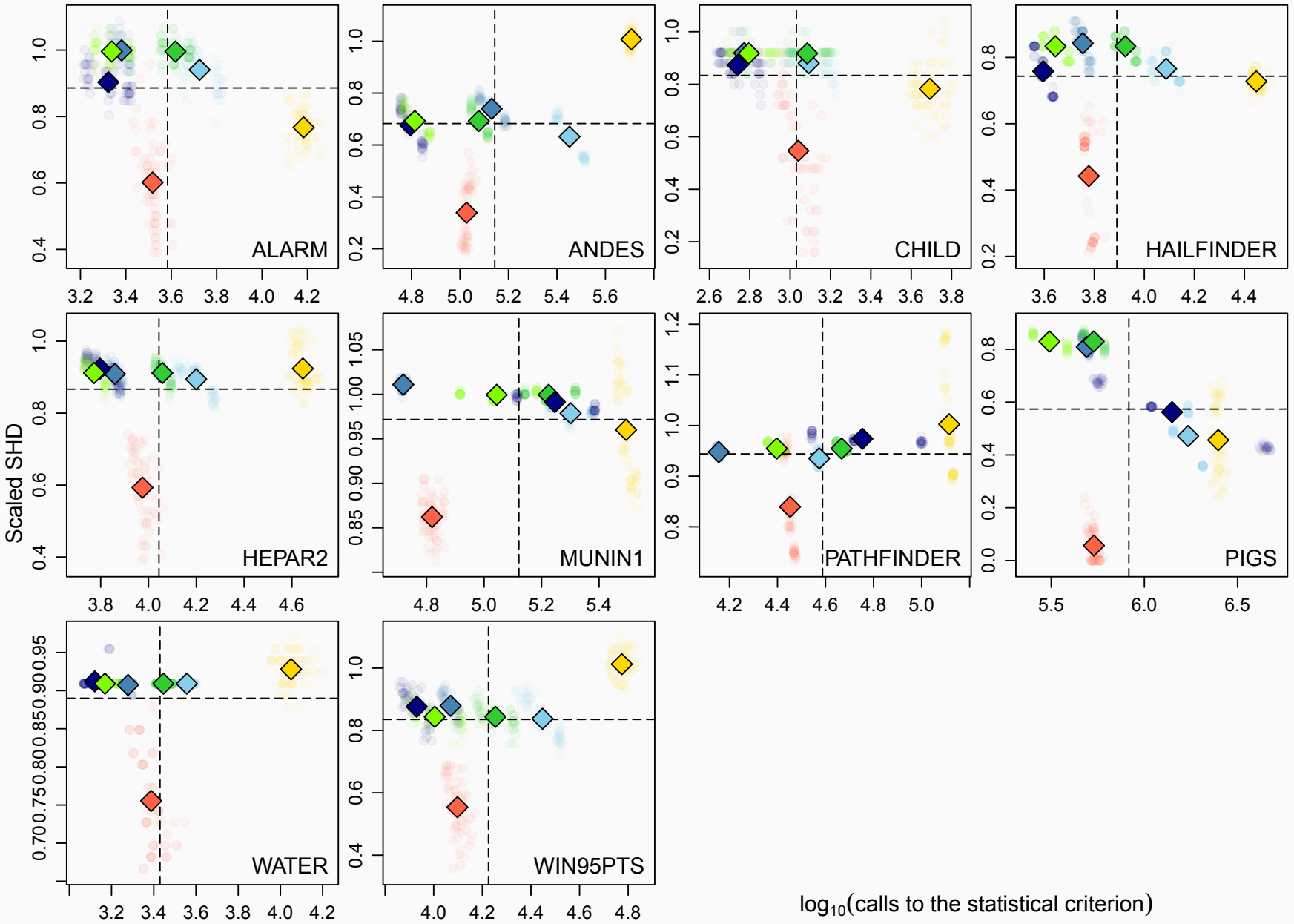
Q2 *Are hybrid algorithms more accurate than constraint-based or score-based algorithms?*

Q3 *Are score-based algorithms slower than constraint-based and hybrid algorithms?*

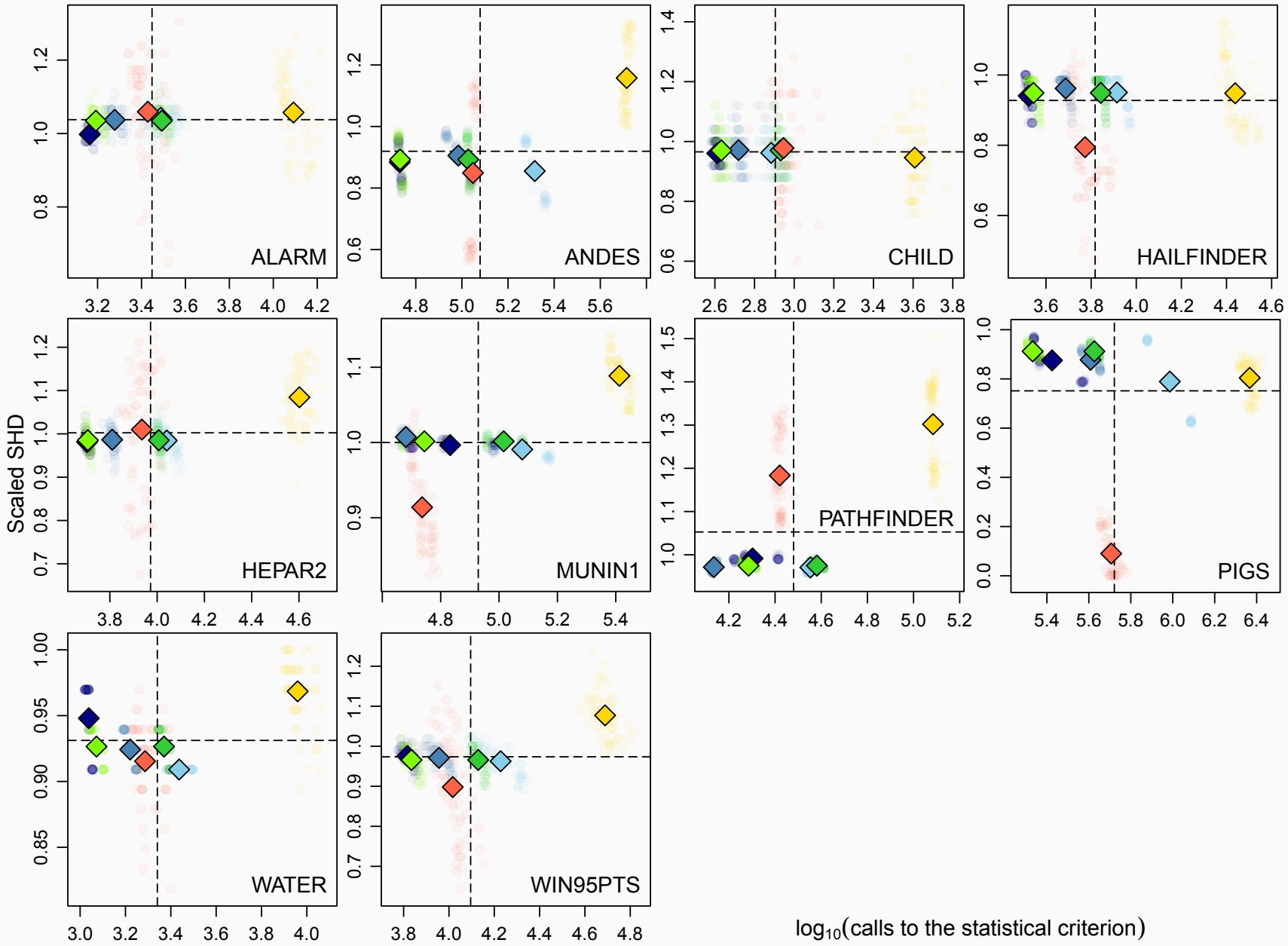
Q4 *Are hybrid algorithms faster than constraint-based or score-based algorithms?*

Q5 *Do the different classes of algorithms present any systematic difference in either speed or accuracy when learning small networks and large networks?*

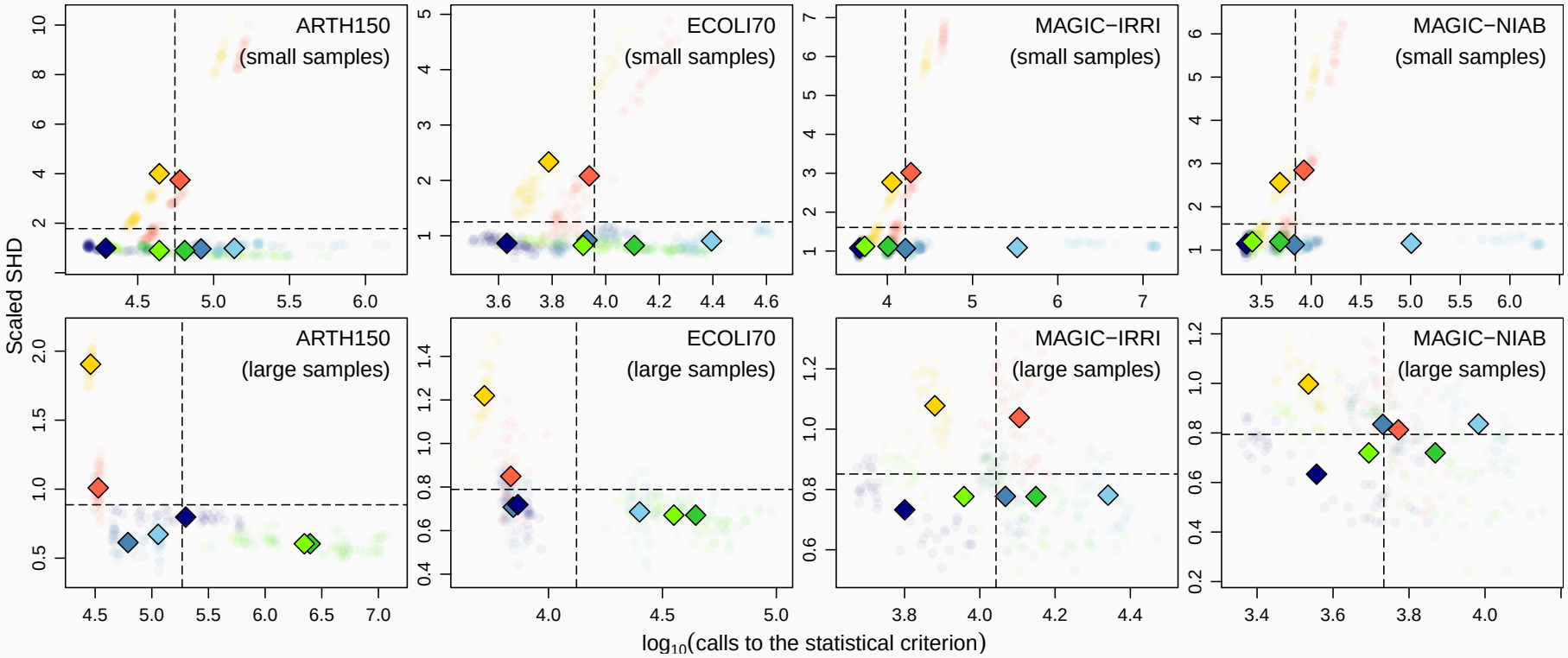
Discrete Bayesian Networks (Large Samples)



Discrete Bayesian Networks (Small Samples)



Gaussian Bayesian Networks



Constraint-based algorithms are in blue shades, hybrid algorithms in green shades, score-based in warm colours. Tabu search is red, the PC algorithm is navy blue.

Conclusions

We can say that, broadly speaking:

Q1 Constraint-based algorithms are more accurate than score-based algorithms for small sample sizes.

Q2 They are as accurate as hybrid algorithms.

Q3 Tabu search, as a score-based algorithm, is faster than constraint-based algorithms more often than not.

Q4 Hybrid algorithms are not faster overall than constraint-based or score-based algorithms. In fact, there is no consistent ordering of the algorithms from these classes.

Q5 No systematic difference in the ranking of different classes of algorithms in terms of speed and accuracy was observed for any class of algorithms for small networks compared to large networks.

This in contrast with the general view in the older literature that only studied trivially small problems.

Relevant Functions in bnlearn

- structure learning algorithms: `pc.stable()`, `hc()`, `tabu()`, `rsmax2()`, etc.
- graphical distances: `shd()`, `hamming()` and `compare()`.
- graphical comparisons: `graphviz.compare()`
- model averaging: `boot.strength()`, `custom.strength()` and `averaged.network()`.

Summary

- Learning the structure of a BN is the first and most crucial step in learning a BN, whether from data or from expert knowledge.
- There are three classes of algorithms to learn the structure of a BN from data: constraint-based, score-based and hybrid.
- The algorithms in these three classes are defined without requiring any specific type of data, which means that it is possible to mix and match tests and scores with algorithms.
- Different classes of algorithms have different strengths and weaknesses. Score-based algorithms are in more common use in practice.
- Scores, tests and algorithms all have tuning parameters and it is usually not clear how their choice impacts the learned networks and how much.
- There is no "best" algorithm: different algorithms will be "best" with different data sets and for different tasks.

Thanks!

Any questions?

Bayesian Networks in Policy and Society

Day 5: Causal Inference

Marco Scutari

Dalle Molle Institute for
Artificial Intelligence (IDSIA)

August 12, 2022

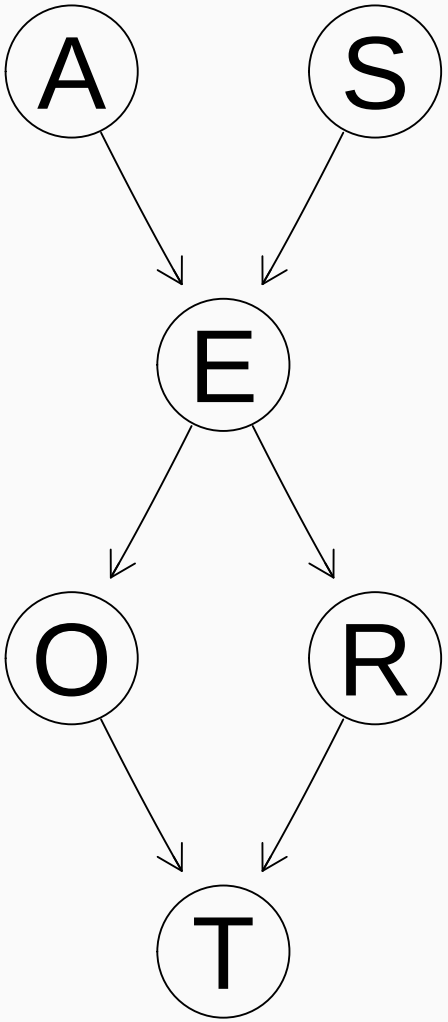
Bayesian Networks are not Necessarily Causal

In the previous lectures, we have defined BNs in terms of conditional independence relationships and probabilistic properties, without any implication that arcs should represent cause-and-effect relationships.

The existence of equivalence classes of BNs that are indistinguishable from a probabilistic point of view provides an immediate proof that arc directions are not indicative of causal effects. The fact that are prognostic and diagnostic formulations of the same BN are identical in terms of inference is another strong hint.

Therefore, while it is appealing to interpret the direction of arcs in causal terms, please do not do it lightly, especially when they have been learned from observational data.

The Train Use Survey as a Prognostic BN

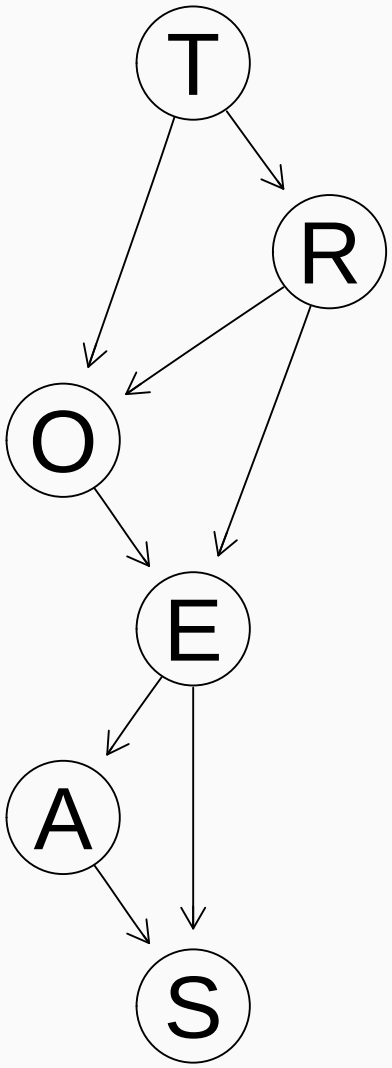


That is a prognostic view of the survey as a BN:

1. the blocks in the experimental design on top (say, stuff from the registry office);
2. the variables of interest in the middle (say, socio-economic indicators);
3. the object of the survey at the bottom (say, means of transport).

Variables that can be thought as “causes” are above variables that can be considered their “effect”, and confounders are above everything else.

The Train Use Survey as a Diagnostic BN



That is a diagnostic view of the survey as a BN: it encodes the same dependence relationships as the prognostic view but is laid out to have "effects" on top and "causes" at the bottom.

Depending on the phenomenon and the goals of the survey, one may have a graph that makes more sense than the other; but they are equivalent for any subsequent inference.

However, from an intuitive point of view it can be argued that a "good" BN should represent the causal structure of the data it is describing. Such BN are usually fairly sparse and their interpretation is at the same time clear and meaningful, as explained by Judea Pearl in his book on causality:

> *It seems that if conditional independence judgments are byproducts of stored causal relationships, then tapping and representing those relationships directly would be a more natural and more reliable way of expressing what we know or believe about the world. This is indeed the philosophy behind causal BNs.*

This is the reason why building a BN from expert knowledge in practice codifies known and expected causal relationships for a given phenomenon.

What Additional Assumptions Do We Need For Causality?

We need three additional assumptions:

- Each variable X_i is conditionally independent of its non-effects, both direct and indirect, given its direct causes (the causal Markov assumption, much like the original but causal);
- There must exist a DAG that is faithful to $\mathrm{P}(\mathbf{X})$: the only dependencies in $\mathrm{P}(\mathbf{X})$ are those arising from d-separation.
- There must be no latent variables (unobserved variables influencing the variables in the network) acting as confounding factors. Such variables may induce spurious correlations between the observed variables, thus introducing bias in the causal network.

What Additional Assumptions Do We Need For Causality?

The third assumption descends from the first two:

- the presence of unobserved variables violates the faithfulness assumption, because the network structure does not include them;
- and possibly the causal Markov property, because an arc may be wrongly added between two observed variables due to the influence of the latent one.

These assumptions are difficult to verify in real-world settings because the set of the potential confounding factors is not usually known. At best, we can address this issue, along with selection bias, by implementing a carefully planned experimental design in which we use blocking to screen out confounding.

Causality and Equivalence Classes

Even when dealing with interventional data collected from a scientific experiment (where we can control at least some variables and observe the resulting changes), there are usually multiple equivalent BNs that represent reasonable causal models. Many arcs may not have a definite direction, resulting in substantially different DAG. When the sample size is small there may also be several non-equivalent BN fitting the data equally well.

Therefore, in general we are not able to identify a single, “best”, causal BN but rather a small set of likely causal BN that fit our knowledge of the data.

The MARKS Example, Revisited

An example of the bias introduced by the presence of a latent variable was illustrated by Edwards ("*Introduction to Graphical Modelling*") using the `marks` data. This data set was originally investigated by Mardia ("*Multivariate Analysis*") and subsequently in Whittaker ("*Graphical Models in Applied Multivariate Statistics*").

`marks` contains the exam scores between 0 and 100 for 88 students across 5 different topics, namely: mechanics (`MECH`), vectors (`VECT`), algebra (`ALG`), analysis (`ANL`) and statistics (`STAT`).

```
library(bnlearn)
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
```

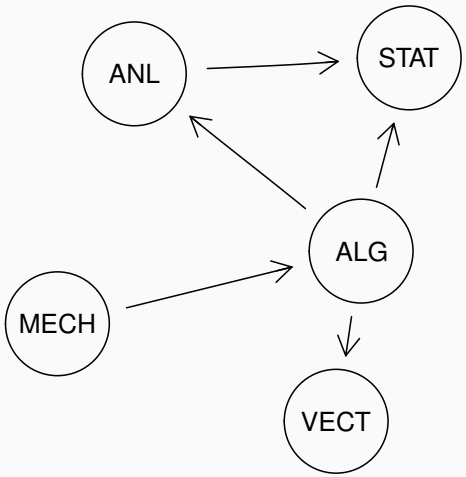
Add Latent Grouping...

Edwards noted that the students apparently belonged to two groups (which we will call "A" and "B") with substantially different academic profiles. He then assigned each student to one of those two groups using the EM algorithm to impute group membership as a latent variable (say, LAT). The EM algorithm assigned the first 52 students (with the exception of number 45) to group "A", and the rest to group "B".

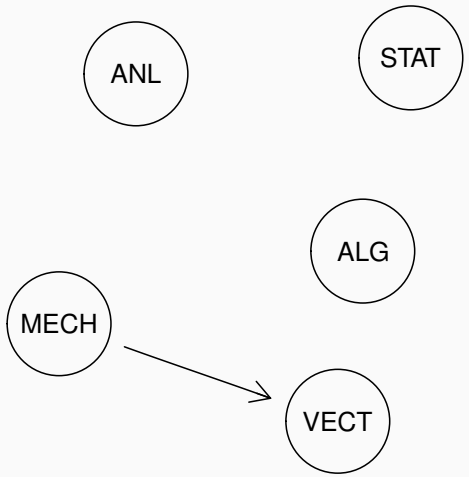
```
latent = factor(c(rep("A", 44), "B", rep("A", 7), rep("B", 36)))
modelstring(hc(marks[latent == "A", ]))
  [1] "[MECH][ALG|MECH][VECT|ALG][ANL|ALG][STAT|ALG:ANL]"
modelstring(hc(marks[latent == "B", ]))
  [1] "[MECH][ALG][ANL][STAT][VECT|MECH]"
modelstring(hc(marks))
  [1] "[MECH][VECT|MECH][ALG|MECH:VECT][ANL|ALG][STAT|ALG:ANL]"
```

... And the Models Look Nothing Alike

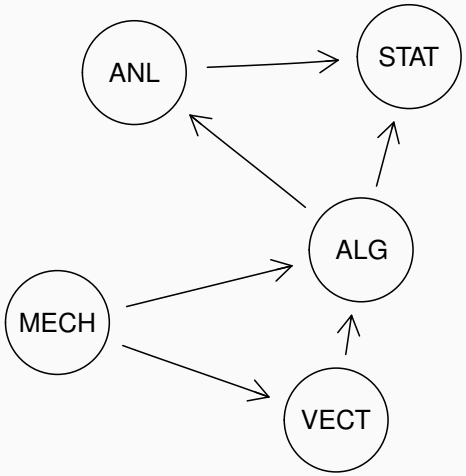
Group A



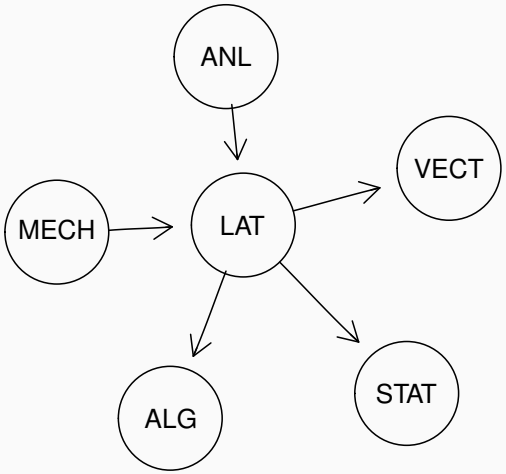
Group B



BN without Latent Grouping



BN with Latent Grouping



The BNs learned from group "A" and group "B" are completely different.

Furthermore, they are both different from the BN learned from the whole data set.

And finally, learning the BN including LAT gives a completely different DAG again.

Distributional Assumptions also Matter

We can choose to discretise the `marks` data and include `LAT` when learning the structure of the discrete BN. Again, we obtain a BN whose DAG is completely different from the rest.

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

```
[1] "[MECH][ANL][LAT|MECH:ANL][VECT|LAT][ALG|LAT][STAT|LAT]"
```

This BN seems to provide a simple interpretation of the relationships between the topics: the grades in mechanics and analysis can be used to infer which group a student belongs to, and that in turn influences the grades in the remaining topics.

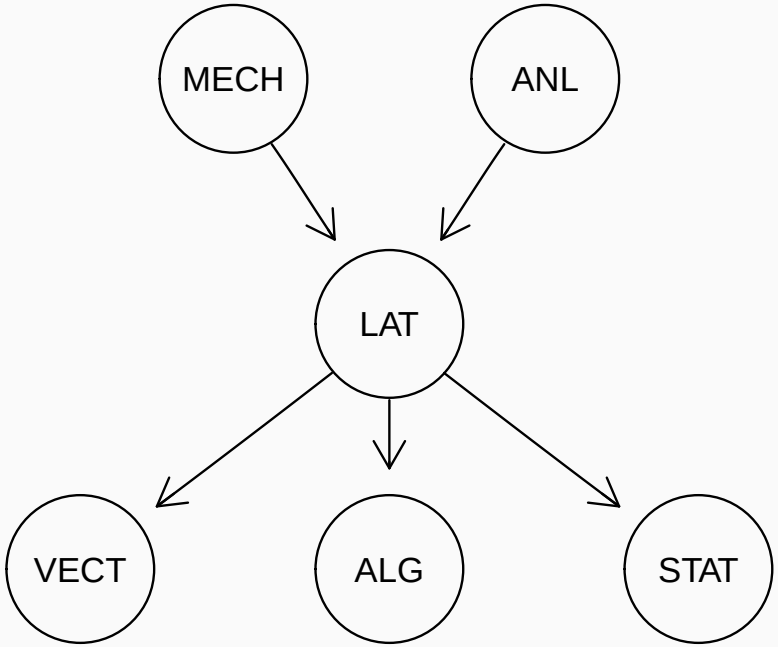
However, if we choose not to discretise:

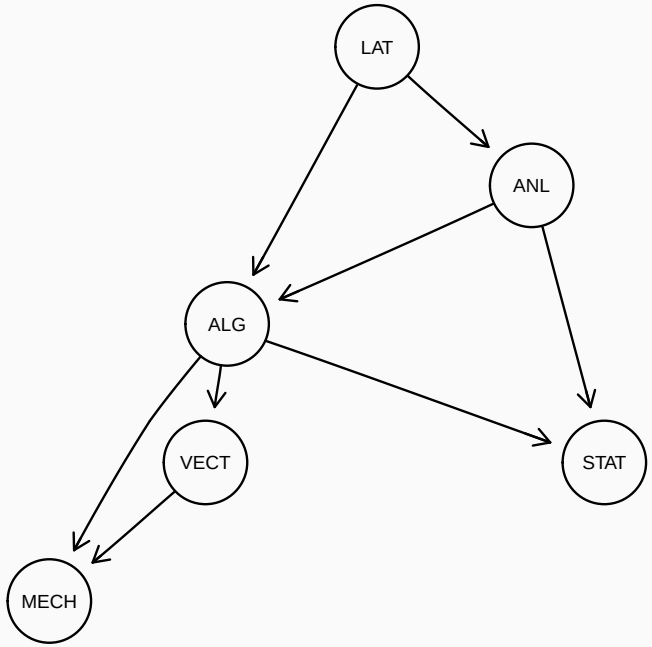
```
modelstring(hc(data.frame(marks, LAT = latent)))
```

```
[1] "[LAT][ANL|LAT][ALG|ANL:LAT][VECT|ALG][STAT|ALG:ANL][MECH|VECT:ALG]"
```

With Discretisation, Without Discretisation

```
par(mfrow = c(1, 2))
graph.par(list(nodes = list(fontsize = 11)))
graphviz.plot(hc(cbind(dmarks, LAT = latent)))
graphviz.plot(hc(cbind(marks, LAT = latent)))
```





We can clearly see that any causal relationship we would have inferred from a DAG learned without taking LAT into account would be potentially spurious. Even after including LAT the situation is not necessarily clear.

Where Things Go Wrong (I)

Suppose that we have a simple GBN of the form $\mathtt{B} \leftarrow \mathtt{A} \rightarrow \mathtt{C}$:

```
complete.bn = custom.fit(model2network("[A][B|A][C|A]"),
  list(A = list(coef = c("(Intercept)" = 0), sd = 1),
       B = list(coef = c("(Intercept)" = 0, A = 3), sd = 0.5),
       C = list(coef = c("(Intercept)" = 0, A = 2), sd = 0.5))
)
```

In this model we have that B is not adjacent to C but $\mathtt{B} \not\perp\!\!\!\perp_G \mathtt{C}$ since they are both children of A:

```
dsep(complete.bn, "B", "C")
```

```
[1] FALSE
```

However, B and C are d-separated by A, and this implies $\mathtt{B} \perp\!\!\!\perp_P \mathtt{C} \mid \mathtt{A}$.

```
dsep(complete.bn, "B", "C", "A")
```

```
[1] TRUE
```

Where Things Go Wrong (II)

If we generate 100 observations from the complete data we can learn the correct DAG from the data.

```
complete.data = rbn(complete.bn, 100)
modelstring(hc(complete.data))
```

```
[1] "[A][B|A][C|A]"
```

Now, assume we do not observe A; that is, A is a latent variable. As a result, B and C are adjacent in the DAG we learn from the incomplete data.

```
modelstring(hc(complete.data[, c("B", "C")]))
```

```
[1] "[B][C|B]"
```

If we do not include A in the model, there is no way to d-separate B and C! As a result they end up being linked in this second DAG, as that is the closest we can get to the set of conditional independencies expressed by the true DAG.

Sometimes Things Do Not Go Wrong (I)

However, consider now a GBN of the form $\mathsf{A} \longrightarrow \mathsf{B} \longrightarrow \mathsf{C}$:

```
complete.bn = custom.fit(model2network("[A][B|A][C|B]"),
  list(A = list(coef = c("(Intercept)" = 0), sd = 1),
       B = list(coef = c("(Intercept)" = 0, A = 3), sd = 0.5),
       C = list(coef = c("(Intercept)" = 0, B = 2), sd = 0.5))
)
```

Now, B depends on A and C depends on B, so by transitivity $\mathsf{A} \not\perp\!\!\!\perp_G \mathsf{C}$ unless we use B to d-separate them.

```
dsep(complete.bn, "B", "A")
  [1] FALSE
dsep(complete.bn, "C", "A")
  [1] FALSE
dsep(complete.bn, "C", "A", "B")
  [1] TRUE
```

Sometimes Things Do Not Go Wrong (II)

Again, if we generate 100 observations from the complete data we can learn the correct DAG from the data.

```
complete.data = rbn(complete.bn, 100)
modelstring(hc(complete.data))
```

```
[1] "[A][B|A][C|B]"
```

The DAG we learn from the incomplete data (omitting B) is still consistent with the true DAG as there is still a path leading from A to C.

```
modelstring(hc(complete.data[, c("A", "C")]))
```

```
[1] "[A][C|A]"
```

The fact that we do not observe the intermediate node B in the causal chain of nodes means that it is now impossible to d-separate A and C and that A appear to be a direct cause of C. The DAG simple glosses over the unobserved B.

Sometimes Things Do Not Go Wrong (III)

Another situation in which latent variables can have a smaller impact when learning the DAG from the data is for v-structures.

```
complete.bn = custom.fit(model2network("[A][B][C|B:A]"),
  list(A = list(coef = c("(Intercept)" = 0), sd = 1),
       B = list(coef = c("(Intercept)" = 0), sd = 0.5),
       C = list(coef = c("(Intercept)" = 0, A = 3, B = 2), sd = 0.5))
)
complete.data = rbn(complete.bn, 100)
modelstring(hc(complete.data[, c("A", "C")]))
```

```
[1] "[A][C|A]"
```

```
modelstring(hc(complete.data[, c("A", "B")]))
```

```
[1] "[A][B]"
```

In this case:

- if one of the parents is a latent variable, we still learn the arc from the other parent correctly;
- if the common child is the latent variable, the parents are not linked by a (spurious) arc.

In Conclusion

- The robustness of causal networks rests on the assumption that there are no latent variables.
- Learning a DAG from data in the presence of latent variables is likely to result in a DAG that is causally wrong, especially when the DAG includes more than 2-3 nodes or encodes a large set of (in)dependence statements.
- Some patterns of latent variables are more problematic than others: a latent variable that is a common cause for two or more observed nodes represents a confounders and as such always leads to wrong causal networks. Other patterns may be less problematic.
- Latent variables and wrong parametric assumptions interact in determining how wrong the learned DAG is, and it is impossible in practice to determine which is causing a missing/spurious arc.

Causal Inference

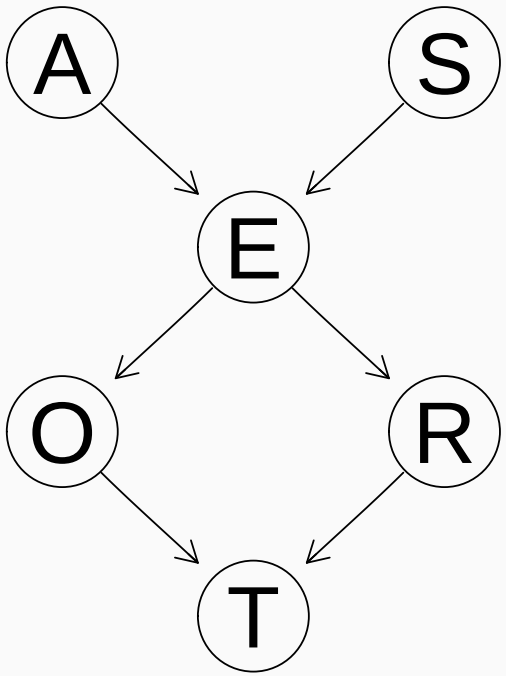
Once we have a causal BN we are happy with, we can again focus on using it to answer relevant questions. In the context of causal networks, we call this causal inference. Compared to the posterior inference we have seen in the previous lecture:

- in probabilistic inference we compute posterior probabilities for events of interest for the observed network;
- in causal inference we compute the effects of interventions for events of interest on a modified network that reflects the interventions.

So in probabilistic inference we are working in an observational setting (look but do not touch), in causal inference we are working in an experimental setting (tweak and see what happens). As a result, causal and probabilistic inference answer different questions; and they will give different probabilities for the same event given the same evidence in general.

The Train Use Survey Revisited

Say that in the original train survey example we collect the data by handing out forms to people chosen at random from the general population: this gives us an observational data set which we can use to learn the BN (from the next lecture).



Say that we are interested in the effect that the residence (R) has on occupation (O), in particular how occupation changes for people living in big cities. The conditional distribution that describes this is:

$$\mathrm{P}(\mathtt{O} \,|\, \mathtt{R} = \texttt{"big"}).$$

The Train Use Survey Revisited (Posterior)

We can compute the posterior distribution of O given R = "big".

```
prop.table(table(cpdist(survey.bn, "O", evidence = (R == "big"))))
```

```
O
   emp   self
0.9548 0.0452
```

This gives us the conditional distribution of the occupation in the part of the general population that lives in a big city. If we compare this with the marginal distribution of O

```
prop.table(table(cpdist(survey.bn, "O", evidence = TRUE)))
```

```
O
   emp   self
0.9478 0.0522
```

we see a $\approx 0.07\%$ increase in employees, so the difference from the overall general population is not very big from a practical perspective.

The Train Use Survey Revisited (Causal, I)

Now, we can wonder: if we allow everybody to live in a big city by starting a public housing program, how will that affect the occupation status? In other words, we will perform an intervention to alter the characteristics of the population from

```
coef(survey.bn$R)
```

```
        E
R        high  uni
  small 0.25 0.20
  big   0.75 0.80
```

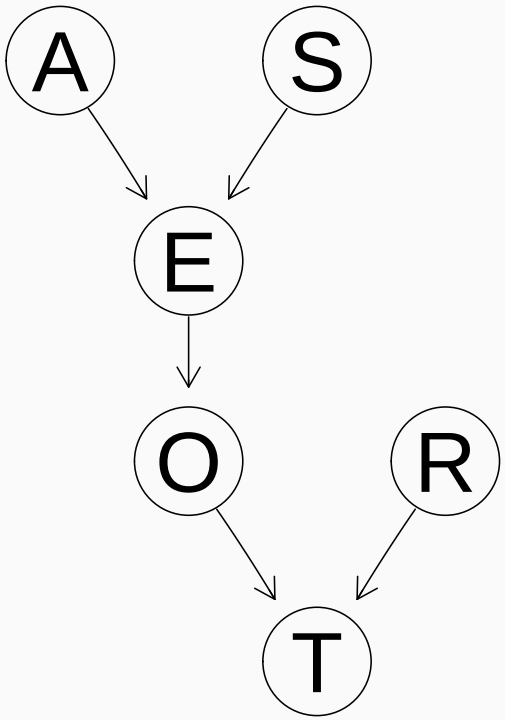
to

```
mut.bn = mutilated(survey.bn, evidence = list(R = "big"))
coef(mut.bn$R)
```

```
  small   big
      0     1
```

because we give everybody a house regardless of their education E.

The Train Use Survey Revisited (Causal, II)



We can compute the effect of this policy on occupation using the mutilated network that incorporates the intervention.

```
prop.table(table(cpdist(mut.bn, "O",
  evidence = TRUE)))
```

```
O
   emp   self
0.9508 0.0492
```

The difference from before the intervention is minimal: this suggests that providing public housing is not an effective policy if the goal is to alter the composition of the workforce.

This the do-calculus: it rests on the idea that we take complete control of the nodes that are subject to intervention and therefore we remove all their parents from the DAG.

The Train Use Survey Revisited (Causal, III)

It is important to note that interventions need not to be hard interventions (e.g. like hard evidence) but can also be soft interventions (e.g. like soft evidence). For instance, we can consider an alternative housing policy that makes the population spread out to small cities with probability 0.5.

```
mut.bn$R = array(c(0.50, 0.50), dim = 2,
             dimnames = list(R = c("small", "big")))
prop.table(table(cpdist(mut.bn, "O",
  evidence = TRUE)))
```

```
O
   emp   self
0.9486 0.0514
```

Again, not much effect on O. Which should not be a surprise since O is d-separated from R in the mutilated network.

```
dsep(mut.bn, "O", "R")
```

```
[1] TRUE
```

There are three key benefit in this approach to causal inference:

- We can simulate the effect of interventions without the need to carry out a real-world experiment, which can be expensive and/or impossible.
- We can use d-separation to identify which variables produce a change in a target variable if we intervene on them.
- We can re-purpose posterior inference to quantify the effects of (possibly complex) causal interventions.

In situation in which designed experiments are possible, causal inference provides a more intuitive representations of classic experimental design:

- We take control of experimental and blocking factors, which then have no parents in the DAG.
- Randomisation is equivalent to a soft causal intervention.
- Since randomised variables have no parents, causality necessarily flows from them to the target variables

Consider: a sufficient condition for randomised controlled trials to protect against confounding is strong ignorability: $E \perp\!\!\!\perp_P T \,|\, V$ where E is the treatment effects, T is the treatment and V are the pre-treatment evidence on the patients.

Pearl's backdoor principle (what we just did) implies ignorability.

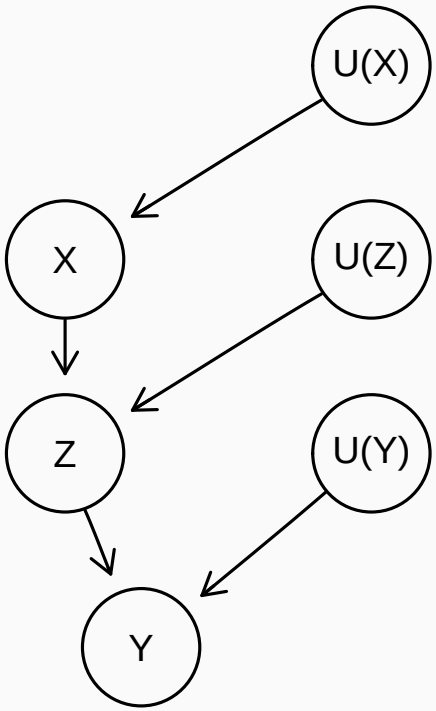
Consider: the S-ignorability assumption roughly says that if we can find a set Z of pre-treatment covariates such that cross-population differences disappear in every stratum $Z = z$, then the problem can perform potential outcomes analyses by averaging over those strata.

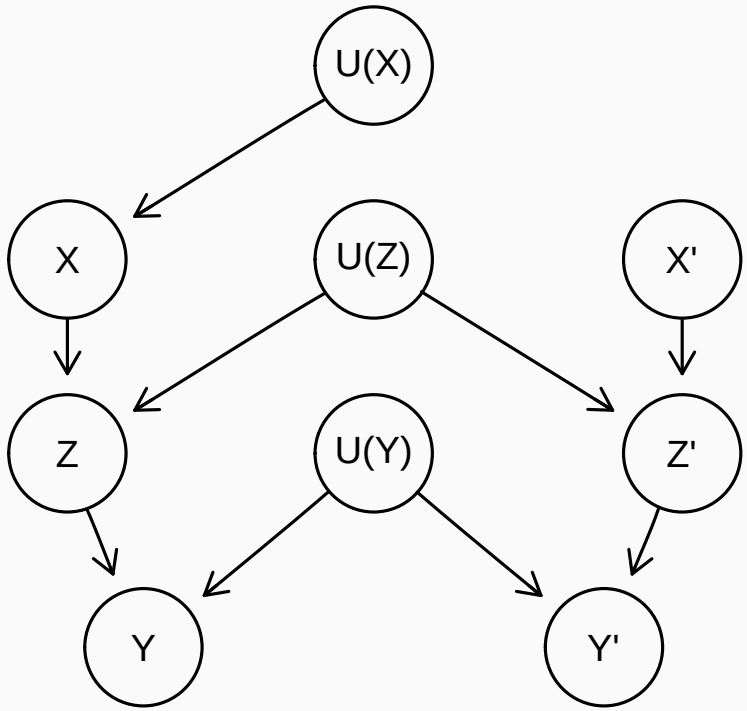
Pearl's backdoor principle can express S-admissibility which is equivalent to S-ignorability if Z are pre-treatment variables. Furthermore, S-ignorability can be verified using counterfactuals...

More Causal Inference: Counterfactuals

A counterfactual is an "if" statement in which the "if" portion is untrue or unrealised. The "if" portion of a counterfactual is called the hypothetical condition, or more often, the antecedent. We use counterfactuals to emphasise our wish to compare two outcomes under the exact same conditions, differing only in one aspect: the antecedent.

We can express both outcomes simultaneously using a twin network:





If we wish to compute the counterfactual probability $\mathrm{P}(\mathsf{Y}_x = y \,|\, \mathsf{Z} = z)$, meaning "what if $\mathsf{X} = x$", we should:

1. Augment the BN into the twin network by adding the counterfactual node X', without any parents.
2. Add counterfactual nodes Y' and Z' for the other variables in the query, with the same parents as Y and Z, respectively.
3. Compute the query $\mathrm{P}(\mathsf{Y}' = y \,|\, \mathsf{Z} = z)$ in the augmented BN.

Automating this construction to general BNs is not trivial: it is usually done manually on BNs with a known DAG. However, using BNs for counterfactuals has the great advantage of allowing us to use d-separation, inference, etc. which are all automated.

Counterfactual Fairness

Counterfactual Fairness: a sensitive attribute A a should not be a cause of a target variable in any (statistical) individual. In other words, changing A while holding things which are not causally dependent on A constant will not change the distribution of Y.

Formally: a predictor $\hat{Y}$ of Y is counterfactually fair given the sensitive attribute $A = a$ and any observed variables $\mathbf{X}$ if

$$\mathrm{P}(\hat{Y}_{A \leftarrow a} = y \,|\, \mathbf{X} = \mathbf{x}, A = a) = \mathrm{P}(\hat{Y}_{A \leftarrow a'} = y \,|\, \mathbf{X} = \mathbf{x}, A = a)$$

for all y and $a' \neq a$.

In graphical terms: $\hat{Y}$ is counterfactually fair if it is a function of the non-descendants of A in causal network.
This condition is too strong for most practical applications because it requires that the distributions are exactly the same for all values of A.

Approximate Counterfactual Fairness

Relaxing the definition of counterfactual fairness by introducing some tolerance in the comparison of the distributions of $\hat{Y}$ under different values of A gives:

Approximate Counterfactual Fairness: a predictor $f(\mathbf{X}, A)$ satisfies (ε, δ)-approximate counterfactual fairness if, given the sensitive attribute $A = a$ and any instantiation $\mathbf{X} = \mathbf{x}$, we have that:

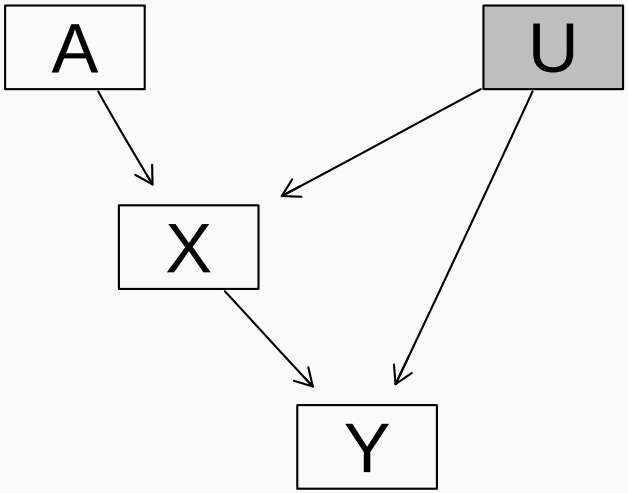
$$\mathrm{P}\left(|f(\mathbf{X}_{A \leftarrow a}, a) - f(\mathbf{X}_{A \leftarrow a'}, a')| \leqslant \varepsilon \,|\, \mathbf{X} = \mathbf{x}, A = a\right) > 1 - \delta$$

for all $a' \neq a$.

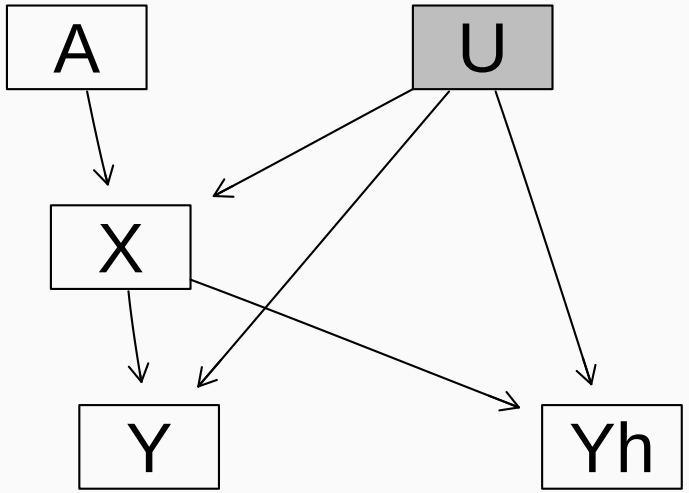
In other words, we allow some degree of unfairness but we assume that the predictor $f(\mathbf{X}, A)$ is mostly fair most of the time. This is common in all fairness literature, regardless of the statistical model.

Consider: an Example

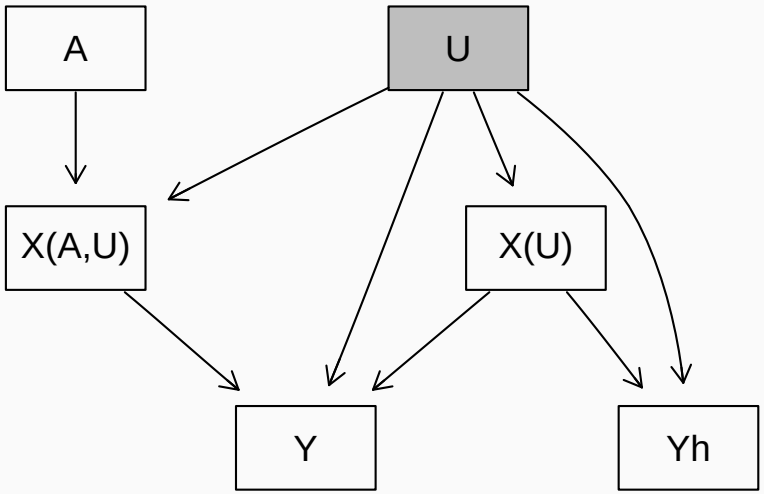
original model



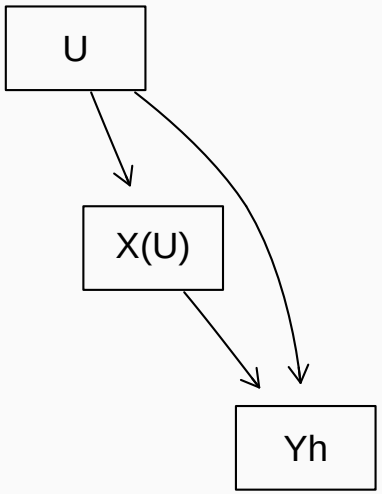
with predictions



split fair and unfair



fair prediction



Missing Data

Latent variables are just on kind of missing data:

- A latent variable is a variable which we know nothing about, either its position in the BN or its distribution.
- An unobserved variable is a variable we do not observe, but which we know the position and the distribution of.
- A partially observed variable is a variable for which we observe some but not all the samples (the rest are denoted as `NA`).

The main problems that arise with missing data are:

- How do we learn the structure of BN from the data?
- Given a DAG, how do we estimate the parameters of the local distributions?

The answers to both questions is the Expectation-Maximisation (EM) and algorithm. (Data Augmentation is an option as well, but we will not cover it in this course).

Classes of Missing Data

There are three classes of missing data:

- Missing completely at random (MCAR): there is no relationship between the missingness of the data and any values, observed or missing. Those missing data points are a random subset of the data.
- Missing at Random (MAR): there is a systematic relationship between the propensity of missing values and the observed data, but not the missing data.
- Missing Not at Random (MNAR): there is a relationship between the propensity of a value to be missing and its values.

MNAR is non-ignorable because the missing data mechanism itself has to be modelled (why the data are missing and what the likely values are). MCAR and MAR are both considered ignorable because we don't have to include any information about the missing data itself when we deal with the missing data.

Representing the Missingness Mechanism

In the context of BNs, each variable has a local distribution $X_i \sim \mathrm{P}(X_i \mid \Pi_{X_i})$ if the data are complete. If X_i has missing data, in the MCAR case

$$X_i \sim \begin{cases} \mathrm{P}(X_i \mid \Pi_{X_i}) & \text{for observed data } X_i^{(O)} \\ \mathrm{P}(X_i \mid \Pi_{X_i}) & \text{for missing data } X_i^{(M)}. \end{cases}$$

The same happens in the MAR case, since the missingness depends on Π_{X_i}. On the other hand, in the MNAR case

$$X_i \sim \begin{cases} \mathrm{P}(X_i^{(O)} \mid \Pi_{X_i}, M) & \text{for observed data } X_i^{(O)} \\ \mathrm{P}(X_i^{(M)} \mid \Pi_{X_i}, M) & \text{for missing data } X_i^{(M)} \end{cases}$$

where M is the missingness mechanism. M is non-ignorable because we cannot estimate the local distribution of X_i properly without knowing the missing values in the first place.

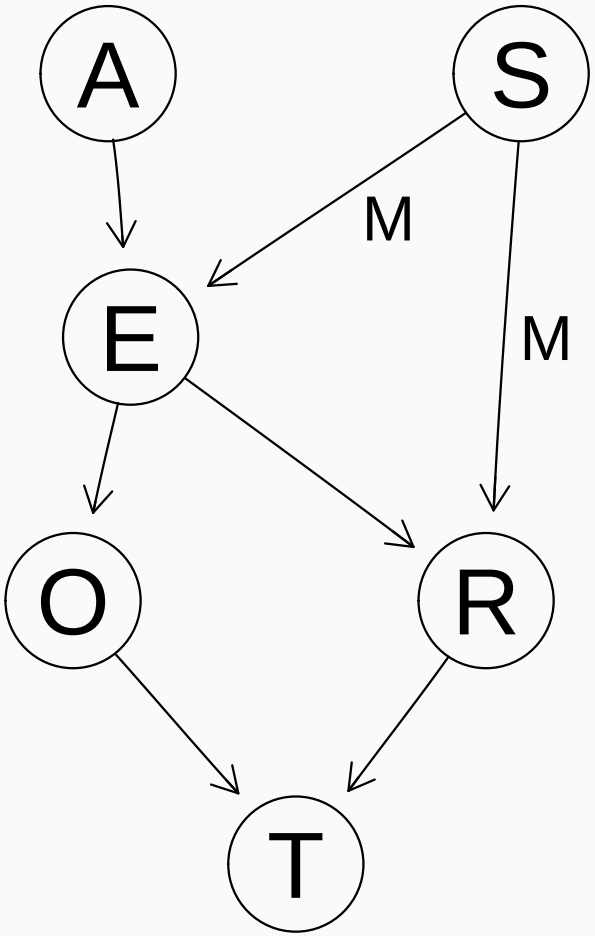
Examples with the Train Use Survey (I)

Since the survey data are collected through a questionnaire, there will be a positive non-response rate for various questions and for the whole questionnaire.

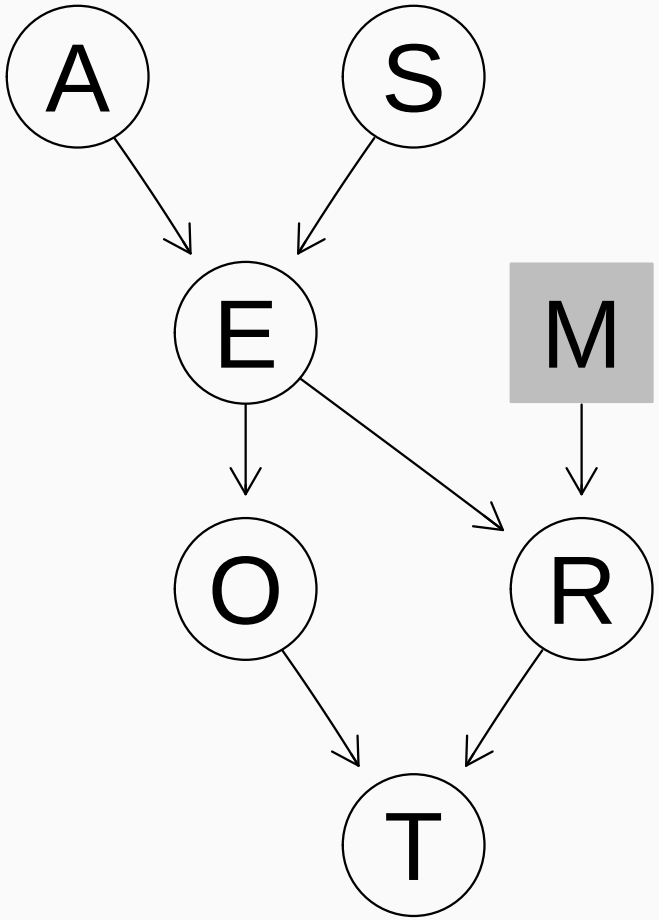
- A MCAR situation may arise when questionnaires are lost in the post – the missingness does not depend on the characteristics of the individual.
- A MAR situation may arise if women refuse to answer some questions in the questionnaire in rates significant higher than men – that is fine since S is observed.
- A MNAR situation may arise if all people in a specific big city do not answer or people of certain social groups do not answer all or part of the questionnaire – we need to introduce M to identify the non-responders.

Examples with the Train Use Survey (II)

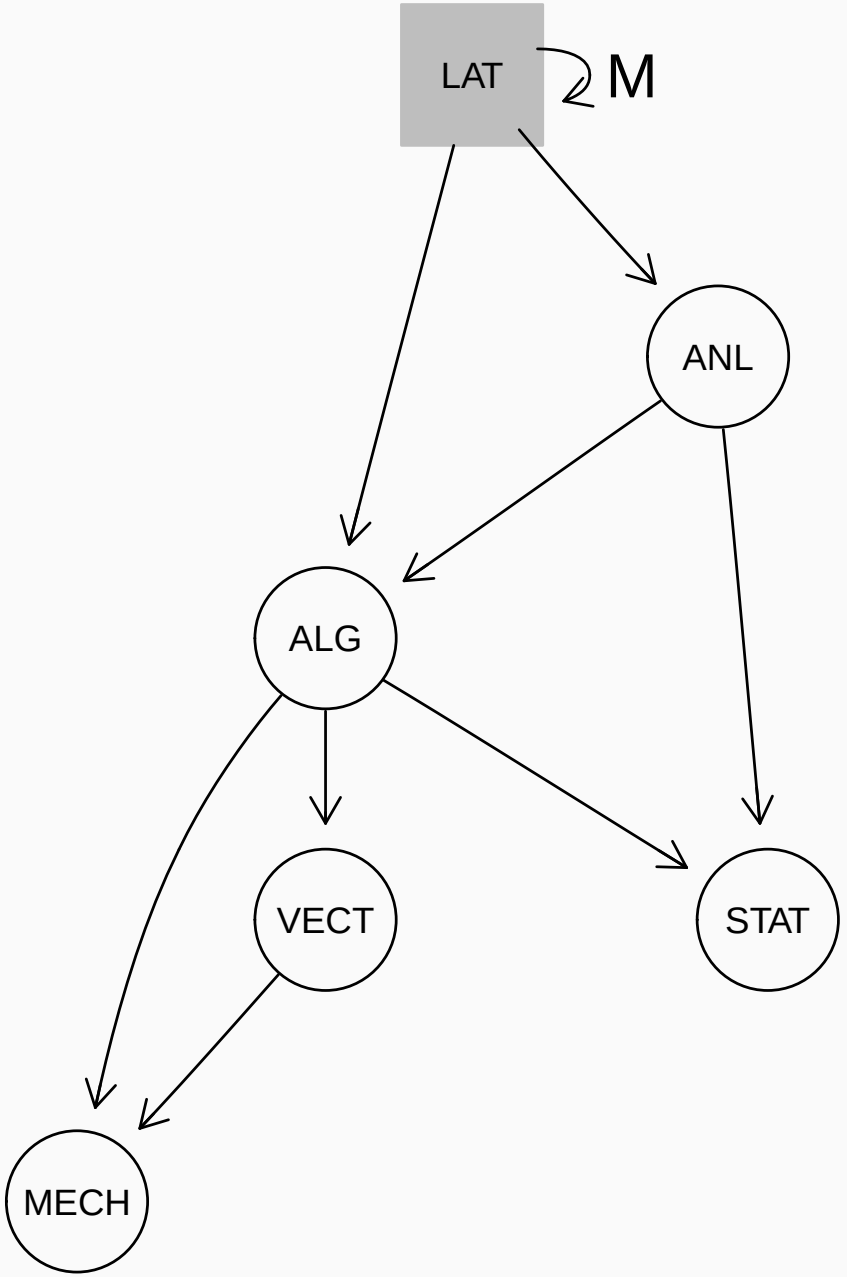
Missing at Random (MAR)



Missing Not at Random (MNAR)



The MARKS Example, Revisited



The latent variable in the MARKS example is MCAR, since all the data are missing the missingness mechanism is simply $\mathrm{P}(\mathtt{M} \mid \mathtt{LAT}) = 1$.

Which shows that MCAR missingness is not necessarily any less problematic than MAR or MNAR, especially for causal inference!

The Expectation-Maximisation (EM) Algorithm

For a generic statistical quantity θ:

---

1. Choose an initial value $\hat{\theta}_0$ for θ.
2. While $|\hat{\theta}_{j-1} - \hat{\theta}_j| < \varepsilon$, increasing j:
   - 2.1 $\hat{\theta}_j = \hat{\theta}_{j-1}$
   - 2.2 Expectation step: compute the probability distribution over the missing values,

$$\mathrm{P}(X_i^{(M)} \mid X_i^{(O)}, \hat{\theta}_j) = \frac{\mathrm{P}(X_i^{(O)} \mid X_i^{(M)}, \hat{\theta}_j)\,\mathrm{P}(X_i^{(M)} \mid \hat{\theta}_j)}{\int_{X_i^{(M)}} \mathrm{P}(X_i^{(O)} \mid X_i^{(M)}, \hat{\theta}_j)\,\mathrm{P}(X_i^{(M)} \mid \hat{\theta}_j)}$$

   - 2.3 Maximisation step: Compute the new estimate $\hat{\theta}_j$ given $\mathrm{P}(X_i^{(M)} \mid X_i^{(O)}, \hat{\theta}_j)$.
3. Estimate θ with the last $\hat{\theta}_j$.

Properties of the EM Algorithm

- There are both Bayesian and frequentist implementations of EM; the former estimates by maximum posterior and the latter by maximum likelihood.

- EM is guaranteed to converge but
  - it may converge to a local maximum and
  - the convergence can be arbitrarily slow.

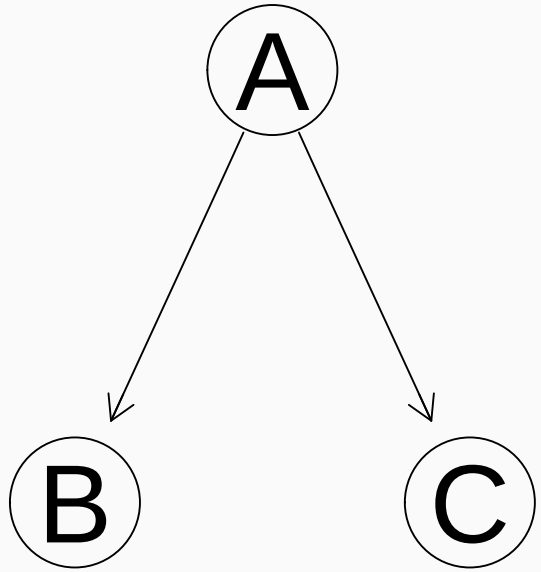
- For BNs, convergence is guaranteed only if all steps are carried out with exact inference; the additional variability introduced by approximate inference can derail convergence.

Manual Approach: a Numeric Example

Consider the following simple numeric example.

```
inc = data.frame(
  A = c(NA, "a1", "a1", "a1", NA),
  B = c("b1", "b2", "b1", "b1", "b2"),
  C = c("c1", "c1", NA, "c2", NA)
)
inc
```

```
      A  B    C
1 <NA> b1   c1
2   a1 b2   c1
3   a1 b1 <NA>
4   a1 b1   c2
5 <NA> b2 <NA>
```



If the data were complete, we could estimate the conditional probabilities in the BN simply by counting the frequencies of the various configurations of values.

Manual Approach: a Numeric Example

Frequentist probability estimates would look like:

$$\mathrm{P}(\mathtt{A} = \mathtt{a1}) = \frac{n_{\mathtt{a1}}}{n}$$

$$\mathrm{P}(\mathtt{A} = \mathtt{a2}) = 1 - \mathrm{P}(\mathtt{A} = \mathtt{a1})$$

$$\mathrm{P}(\mathtt{B} = \mathtt{b1} \,|\, \mathtt{A} = \mathtt{a1}) = \frac{n_{\mathtt{b1,a1}}}{n_{\mathtt{a1}}}$$

$$\mathrm{P}(\mathtt{B} = \mathtt{b2} \,|\, \mathtt{A} = \mathtt{a1}) = 1 - \mathrm{P}(\mathtt{B} = \mathtt{b1} \,|\, \mathtt{A} = \mathtt{a1})$$

$$\mathrm{P}(\mathtt{B} = \mathtt{b1} \,|\, \mathtt{A} = \mathtt{a2}) = \frac{n_{\mathtt{b1,a2}}}{n_{\mathtt{a2}}}$$

$$\mathrm{P}(\mathtt{B} = \mathtt{b2} \,|\, \mathtt{A} = \mathtt{a2}) = 1 - \mathrm{P}(\mathtt{B} = \mathtt{b1} \,|\, \mathtt{A} = \mathtt{a2})$$

$$\mathrm{P}(\mathtt{C} = \mathtt{c1} \,|\, \mathtt{A} = \mathtt{a2}) = \frac{n_{\mathtt{c1,a2}}}{n_{\mathtt{a2}}}$$

$$\mathrm{P}(\mathtt{C} = \mathtt{c2} \,|\, \mathtt{A} = \mathtt{a1}) = 1 - \mathrm{P}(\mathtt{C} = \mathtt{c1} \,|\, \mathtt{A} = \mathtt{a1})$$

$$\mathrm{P}(\mathtt{C} = \mathtt{c1} \,|\, \mathtt{A} = \mathtt{a2}) = \frac{n_{\mathtt{c1,a2}}}{n_{\mathtt{a2}}}$$

$$\mathrm{P}(\mathtt{C} = \mathtt{c2} \,|\, \mathtt{A} = \mathtt{a2}) = 1 - \mathrm{P}(\mathtt{C} = \mathtt{c1} \,|\, \mathtt{A} = \mathtt{a2})$$

Manual Approach: a Numeric Example

Looking at A first, the count we need is

$$n_{\mathtt{a1}} = \mathbb{1}(\text{1st observation is a1}) + \mathbb{1}(\text{2nd observation is a1}) + \mathbb{1}(\text{3rd observation is a1}) + \mathbb{1}(\text{4th observation is a1}) + \mathbb{1}(\text{5th observation is a1}),$$

where $\mathbb{1}()$ is equal to 1 if its argument is true and 0 otherwise.

For the 2nd, 3rd and 4th observations we know that A = a1, so we can write

$$n_{\mathtt{a1}} = \mathbb{1}(\text{1st observation is a1}) + 1 + 1 + 1 + \mathbb{1}(\text{5th observation is a1}).$$

From a different perspective, what we are saying is that we know that those observations take value a1 with probability 1.

Manual Approach: a Numeric Example

If we take this new perspective further, we can then write:

$$n_{\texttt{a1}} = \mathrm{P}(\text{1st observation is a1}) + \mathrm{P}(\text{2nd observation is a1}) + \mathrm{P}(\text{3rd observation is a1}) + \mathrm{P}(\text{4th observation is a1}) + \mathrm{P}(\text{5th observation is a1}).$$

If we had no missing values, each of those probabilities would be equal to either 0 or 1 and tallying them up would give us the usual empirical frequency $n_{\texttt{a1}}$.

But since have missing values, all we can do is to say

$$n_{\texttt{a1}} = \mathrm{P}(\text{1st observation is a1}) + 1 + 1 + 1 + \mathrm{P}(\text{5th observation is a1}).$$

This is easier to work with, because now we can use the axioms of probability to borrow information from the other variables to fill the missing values.

Manual Approach: a Numeric Example

Consider that our BN can be written as

$$\mathrm{P}(\mathsf{A}, \mathsf{B}, \mathsf{C}) = \mathrm{P}(\mathsf{A})\,\mathrm{P}(\mathsf{B} \mid \mathsf{A})\,\mathrm{P}(\mathsf{C} \mid \mathsf{A}).$$

We cannot use $\mathrm{P}(\mathsf{A})$ to fill in the missing values because from the data we would guess $\mathrm{P}(\mathsf{A} = \mathsf{a1}) = 1$ and $\mathrm{P}(\mathsf{A} = \mathsf{a2}) = 0$ since a2 is never observed; but that is not desirable if we assume a2 can actually happen.

What we can do is to reverse the dependencies in the model to get

$$\mathrm{P}(\mathsf{A})\,\mathrm{P}(\mathsf{B} \mid \mathsf{A})\,\mathrm{P}(\mathsf{C} \mid \mathsf{A}) = \mathrm{P}(\mathsf{A})\,\mathrm{P}(\mathsf{B}, \mathsf{C} \mid \mathsf{A}) = \mathrm{P}(\mathsf{A} \mid \mathsf{B}, \mathsf{C})\,\mathrm{P}(\mathsf{B}, \mathsf{C})$$

an use $\mathrm{P}(\mathsf{A} \mid \mathsf{B}, \mathsf{C})$ instead of $\mathrm{P}(\mathsf{A})$ to borrow the information from the other variables.

Manual Approach: a Numeric Example

This gets us to

$$n_{\mathsf{a1}}^{(0)} = \mathrm{P}_0(\mathsf{A} = \mathsf{a1} \,|\, \mathsf{B} = \mathsf{b1}, \mathsf{C} = \mathsf{c1}) + 1 + 1 + 1 + \mathrm{P}_0(\mathsf{A} = \mathsf{a1} \,|\, \mathsf{B} = \mathsf{b1})$$

and if we assume as a starting point that

$$\mathrm{P}_0(\mathsf{A}) = \begin{cases} 0.5 & \text{for a1} \\ 0.5 & \text{for a2} \end{cases} \qquad \mathrm{P}_0(\mathsf{A} \,|\, \mathsf{B}, \mathsf{C}) = \begin{cases} 0.5 & \text{for a1 for all B, C} \\ 0.5 & \text{for a2 for all B, C} \end{cases}$$

$$\mathrm{P}_0(\mathsf{A} \,|\, \mathsf{B}) = \begin{cases} 0.5 & \text{for a1 given b1} \\ 0.5 & \text{for a2 given b2} \end{cases}$$

we can now compute

$$n_{\mathsf{a1}}^{(0)} = \frac{1}{2} + 1 + 1 + 1 + \frac{1}{2} = 4$$

as an initial estimate for n_{a1}.

Manual Approach: a Numeric Example

Replacing $n_{\mathtt{a1}}$ with $n_{\mathtt{a1}}^{(0)}$, we can estimate

$$\mathrm{P}_1(\mathtt{A} = \mathtt{a1}) = \frac{4}{5} = 0.8, \qquad \mathrm{P}_1(\mathtt{A} = \mathtt{a2}) = 1 - \mathrm{P}(\mathtt{A} = \mathtt{a1}) = 0.2.$$

Moving to B, what we need to estimate is

$$\mathrm{P}(\mathtt{B} = \mathtt{b1} \,|\, \mathtt{A} = \mathtt{a1}) = \frac{n_{\mathtt{b1,a1}}}{n_{\mathtt{a1}}} = \frac{n_{\mathtt{b1,a1}}}{4}$$

where

$$\begin{aligned} n_{\mathtt{b1,a1}} &= \mathrm{P}(\text{1st observation is b1, a1}) + \mathrm{P}(\text{2nd observation is b1, a1}) + \\ &\quad \mathrm{P}(\text{3rd observation is b1, a1}) + \mathrm{P}(\text{4th observation is b1, a1}) + \\ &\quad \mathrm{P}(\text{5th observation is b1, a1}) \\ &= \mathrm{P}(\text{1st observation is b1, a1}) + 0 + 1 + 1 + 0 \end{aligned}$$

since only for the first observation B = b1 we do not know the value of A.

Manual Approach: a Numeric Example

We are working with the joint frequencies of A and B; hence we choose as starting probabilities the joint uniform

$$\mathrm{P}_0(\mathtt{B}, \mathtt{A} \,|\, \mathtt{C}) = \begin{cases} 0.25 & \text{for b1, a1 for both c1, c2} \\ 0.25 & \text{for b1, a2 for both c1, c2} \\ 0.25 & \text{for b2, a1 for both c1, c2} \\ 0.25 & \text{for b2, a2 for both c1, c2} \end{cases}$$

so that the probability of the first observation is

$$\begin{aligned} &\mathrm{P}(\text{1st observation is b1, a1}) \\ &= \mathrm{P}_0(\mathtt{A} = \mathtt{a1} \,|\, \mathtt{B} = \mathtt{b1}, \mathtt{C} = \mathtt{c1}) \\ &= \frac{\mathrm{P}_0(\mathtt{A} = \mathtt{a1}, \mathtt{B} = \mathtt{b1} \,|\, \mathtt{C} = \mathtt{c1})}{\mathrm{P}_0(\mathtt{B} = \mathtt{b1} \,|\, \mathtt{C} = \mathtt{c1})} \\ &= \frac{\mathrm{P}_0(\mathtt{A} = \mathtt{a1}, \mathtt{B} = \mathtt{b1} \,|\, \mathtt{C} = \mathtt{c1})}{\mathrm{P}_0(\mathtt{A} = \mathtt{a1}, \mathtt{B} = \mathtt{b1} \,|\, \mathtt{C} = \mathtt{c1}) + \mathrm{P}_0(\mathtt{A} = \mathtt{a2}, \mathtt{B} = \mathtt{b1} \,|\, \mathtt{C} = \mathtt{c1})} \\ &= \frac{0.25}{0.25 + 0.25} = 0.5. \end{aligned}$$

Manual Approach: a Numeric Example

We can now compute

$$n^{(0)}_{\mathtt{b1},\mathtt{a1}} = 0.5 + 0 + 1 + 1 + 0 = 2.5$$

and in turn

$$\mathrm{P}(\mathtt{B} = \mathtt{b1} \,|\, \mathtt{A} = \mathtt{a1}) = \frac{n^{(0)}_{\mathtt{b1},\mathtt{a1}}}{n^{(0)}_{\mathtt{a1}}} = \frac{2.5}{4} = 0.625$$

$$\mathrm{P}(\mathtt{B} = \mathtt{b2} \,|\, \mathtt{A} = \mathtt{a1}) = 1 - \mathrm{P}(\mathtt{B} = \mathtt{b1} \,|\, \mathtt{A} = \mathtt{a1}) = 0.375$$

which is the first of the two conditional distributions of B.

The second of the two conditional distributions of B is computed in the same way, starting from

$$\mathrm{P}(\mathtt{B} = \mathtt{b1} \,|\, \mathtt{A} = \mathtt{a2}) = \frac{n_{\mathtt{b1},\mathtt{a2}}}{n_{\mathtt{a2}}} = \frac{n_{\mathtt{b1},\mathtt{a2}}}{n - n_{\mathtt{a1}}} = \frac{n_{\mathtt{b1},\mathtt{a2}}}{1}.$$

Manual Approach: a Numeric Example

Using the same $\mathrm{P}_0(\mathtt{A} \mid \mathtt{B}, \mathtt{C})$ as before,

$$\begin{aligned} n_{\mathtt{b1},\mathtt{a2}} &= \mathrm{P}(\text{1st observation is } \mathtt{b1}, \mathtt{a2}) + \mathrm{P}(\text{2nd observation is } \mathtt{b1}, \mathtt{a2}) + \\ &\quad \mathrm{P}(\text{3rd observation is } \mathtt{b1}, \mathtt{a2}) + \mathrm{P}(\text{4th observation is } \mathtt{b1}, \mathtt{a2}) + \\ &\quad \mathrm{P}(\text{5th observation is } \mathtt{b1}, \mathtt{a2}) \\ &= \mathrm{P}(\text{1st observation is } \mathtt{b1}, \mathtt{a2}) + 0 + 0 + 0 + 0, \end{aligned}$$

leading to $n^{(0)}_{\mathtt{b1},\mathtt{a2}} = 0.5 + 0 + 0 + 0 + 0 = 0.5$.

If we replace $n^{(0)}_{\mathtt{b1},\mathtt{a2}}$ in the expression above we get

$$\mathrm{P}_1(\mathtt{B} = \mathtt{b1} \mid \mathtt{A} = \mathtt{a2}) = \frac{n^{(0)}_{\mathtt{b1},\mathtt{a2}}}{n - n^{(0)}_{\mathtt{a1}}} = \frac{0.5}{5 - 4} = 0.5,$$

$$\mathrm{P}_1(\mathtt{B} = \mathtt{b2} \mid \mathtt{A} = \mathtt{a2}) = 1 - \mathrm{P}(\mathtt{B} = \mathtt{b1} \mid \mathtt{A} = \mathtt{a2}) = 0.5.$$

This makes complete sense: we never observe the combination of values b1, a2 so there is no information to learn from the data.

Manual Approach: a Numeric Example

As for C, what we need to estimate is

$$\mathrm{P}(\mathtt{C} = \mathtt{c1} \,|\, \mathtt{A} = \mathtt{a1}) = \frac{n_{\mathtt{c1,a1}}}{n_{\mathtt{a1}}}$$

where we know that $n_{\mathtt{a1}}^{(0)} = 4$ from before and

$$\begin{aligned} n_{\mathtt{c1,a1}} = {} & \mathrm{P}(\text{1st observation is } \mathtt{c1}, \mathtt{a1}) + \mathrm{P}(\text{2nd observation is } \mathtt{c1}, \mathtt{a1}) + \\ & \mathrm{P}(\text{3rd observation is } \mathtt{c1}, \mathtt{a1}) + \mathrm{P}(\text{4th observation is } \mathtt{c1}, \mathtt{a1}) + \\ & \mathrm{P}(\text{5th observation is } \mathtt{c1}, \mathtt{a1}). \end{aligned}$$

The starting probabilities are the joint uniform distribution over C and A

$$\mathrm{P}_0(\mathtt{C}, \mathtt{A} \,|\, \mathtt{B}) = \begin{cases} 0.25 & \text{for c1, a1 given b1, b2} \\ 0.25 & \text{for c1, a2 given b1, b2} \\ 0.25 & \text{for c2, a1 given b1, b2} \\ 0.25 & \text{for c2, a2 given b1, b2} \end{cases}.$$

Manual Approach: a Numeric Example

Considering the partial observations we have for A and C, we can rewrite the above as:

$$n^{(0)}_{\mathsf{c1},\mathsf{a1}} = \mathrm{P}_0(\mathsf{A} = \mathsf{a1} \,|\, \mathsf{C} = \mathsf{c1}, \mathsf{B} = \mathsf{b1}) + 1 + \mathrm{P}_0(\mathsf{C} = \mathsf{c1} \,|\, \mathsf{A} = \mathsf{a1}, \mathsf{B} = \mathsf{b1}) + 0 + \mathrm{P}_0(\mathsf{C} = \mathsf{c1}, \mathsf{A} = \mathsf{a1} \,|\, \mathsf{B} = \mathsf{b2}).$$

By the axioms of probability,

$$\mathrm{P}_0(\mathsf{A} \,|\, \mathsf{C}, \mathsf{B}) = \frac{\mathrm{P}_0(\mathsf{C}, \mathsf{A} \,|\, \mathsf{B})}{\mathrm{P}_0(\mathsf{C} \,|\, \mathsf{B})} = \frac{\mathrm{P}_0(\mathsf{C}, \mathsf{A} \,|\, \mathsf{B})}{\mathrm{P}_0(\mathsf{C}, \mathsf{A} = \mathsf{a1} \,|\, \mathsf{B}) + \mathrm{P}_0(\mathsf{C}, \mathsf{A} = \mathsf{a2} \,|\, \mathsf{B})} = \frac{0.25}{0.25 + 0.25} = 0.5$$

$$\mathrm{P}_0(\mathsf{C} \,|\, \mathsf{A}, \mathsf{B}) = \frac{\mathrm{P}_0(\mathsf{C}, \mathsf{A} \,|\, \mathsf{B})}{\mathrm{P}_0(\mathsf{A} \,|\, \mathsf{B})} = \frac{\mathrm{P}_0(\mathsf{A}, \mathsf{C} \,|\, \mathsf{B})}{\mathrm{P}_0(\mathsf{C} = \mathsf{c1}, \mathsf{A} \,|\, \mathsf{B}) + \mathrm{P}_0(\mathsf{C} = \mathsf{c2}, \mathsf{A} \,|\, \mathsf{B})} = \frac{0.25}{0.25 + 0.25} = 0.5$$

Manual Approach: a Numeric Example

As a result, $n^{(0)}_{\mathsf{c1,a1}} = 0.5 + 1 + 0.5 + 0 + 0.25 = 2.25$ and

$$\mathrm{P}_1(\mathsf{C} = \mathsf{c1} \,|\, \mathsf{A} = \mathsf{a1}) = \frac{n^{(0)}_{\mathsf{c1,a1}}}{n^{(0)}_{\mathsf{a1}}} = \frac{2.25}{4} = 0.56,$$

$$\mathrm{P}_1(\mathsf{C} = \mathsf{c2} \,|\, \mathsf{A} = \mathsf{a1}) = 1 - \mathrm{P}(\mathsf{C} = \mathsf{c1} \,|\, \mathsf{A} = \mathsf{a1}) = 0.44.$$

The second conditional distribution of C,

$$\mathrm{P}(\mathsf{C} = \mathsf{c1} \,|\, \mathsf{A} = \mathsf{a2}) = \frac{n_{\mathsf{c1,a2}}}{n_{\mathsf{a2}}},$$

requires $n_{\mathsf{a2}} = 1$ from before and

$$\begin{aligned} n_{\mathsf{c1,a2}} &= \mathrm{P}(\text{1st observation is c1, a2}) + \mathrm{P}(\text{2nd observation is c1, a2}) + \\ &\quad \mathrm{P}(\text{3rd observation is c1, a2}) + \mathrm{P}(\text{4th observation is c1, a2}) + \\ &\quad \mathrm{P}(\text{5th observation is c1, a2}) \\ &= \mathrm{P}_0(\mathsf{A} = \mathsf{a2} \,|\, \mathsf{C} = \mathsf{c1}, \mathsf{B} = \mathsf{b1}) + 0 + 0 + 0 + \\ &\quad \mathrm{P}_0(\mathsf{C} = \mathsf{c1}, \mathsf{A} = \mathsf{a2} \,|\, \mathsf{B} = \mathsf{b2}) = 0.5 + 0 + 0 + 0 + 0.25 = 0.75. \end{aligned}$$

Manual Approach: Summary

This gives the last conditional probability distribution:

$$\mathrm{P}_1(\mathsf{C} = \mathsf{c1} \,|\, \mathsf{A} = \mathsf{a2}) = \frac{n^{(0)}_{\mathsf{c1},\mathsf{a2}}}{n^{(0)}_{\mathsf{a2}}} = \frac{0.75}{1} = 0.75,$$

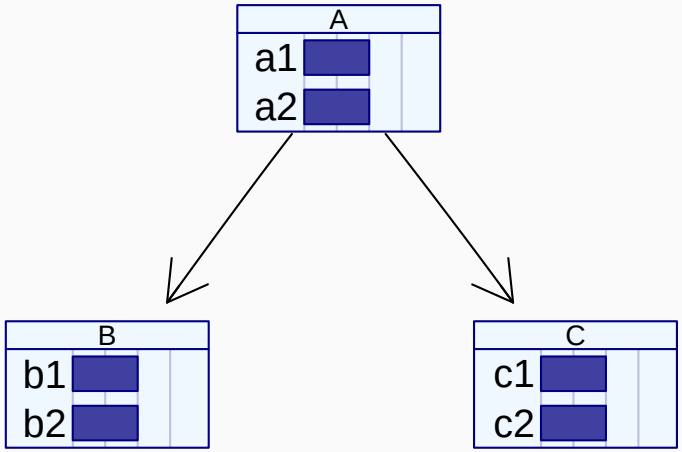
$$\mathrm{P}_1(\mathsf{C} = \mathsf{c2} \,|\, \mathsf{A} = \mathsf{a2}) = 1 - \mathrm{P}(\mathsf{C} = \mathsf{c1} \,|\, \mathsf{A} = \mathsf{a2}) = 0.25.$$

What did we do?

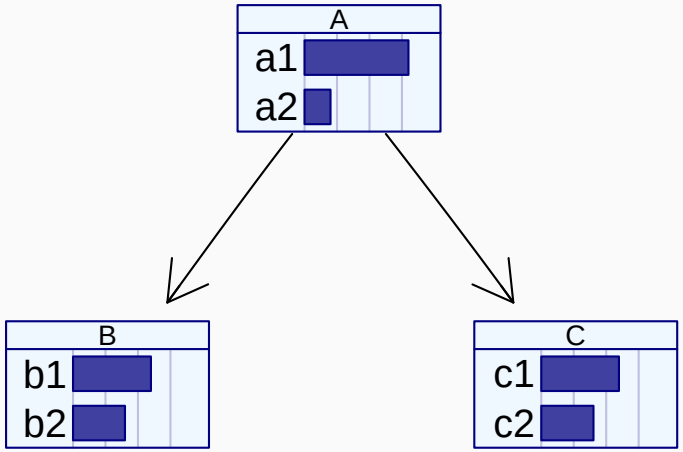
1. We could not estimate the conditional probabilities due to the missing values in the data.
2. We assumed all distributions were uniform as a starting point.
3. In the frequencies we needed to estimate the conditional probabilities, we replaced the missing values with the probabilities of observing corresponding values.
4. We computed the frequencies, and used them to compute the conditional probabilities in the BN.

Manual Approach: Summary

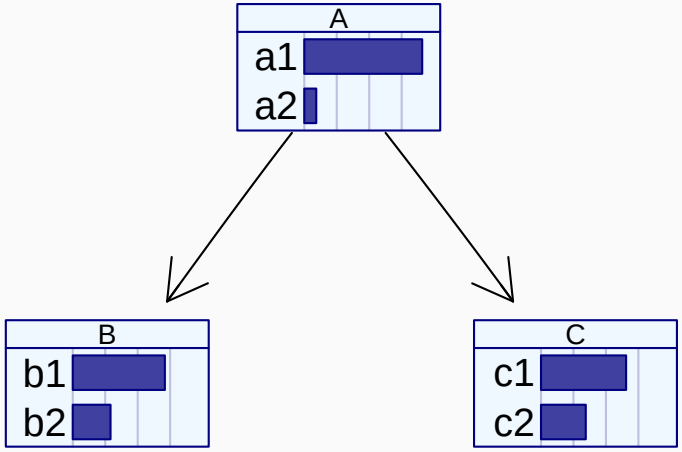
Prior



Step 1



Step 2



... and so on, and so forth...

The EM Algorithm, Unknown Graph Structure

Learning the (CP)DAG of a BN in the presence of missing data (in addition to the parameters) is a problem that is challenging from both a statistical and a computational point of view. Friedman extended the EM algorithm to work for this task, and called the resulting algorithm Structural EM:

---

1. Start with a BN $\mathcal{B}_0$ with an empty DAG $\mathcal{G}_0$ (with no arcs).
2. As long as $\mathcal{B}_i$ is different from B_{i-1}:
   - 2.1 Expectation step: impute the missing data with their posterior expectations or their maximum likelihood estimates using the current BN.
   - 2.2 Maximisation step: learn an updated BN from the completed data.

The MARKS Example, Revisited (I)

```
ldmarks = data.frame(dmarks, LAT = factor(rep(NA, nrow(dmarks)),
            levels = c("A", "B")))
imputed = ldmarks
imputed$LAT = sample(factor(c("A", "B")), nrow(dmarks), replace = TRUE)

# initialise an empty BN that includes LAT.
bn = bn.fit(empty.graph(names(ldmarks)), imputed)
bn$LAT = array(c(0.5, 0.5), dim = 2, dimnames = list(c("A", "B")))

# three manual iterations of structural EM.
for (i in 1:3) {

  # expectation step.
  imputed = impute(bn, ldmarks, method = "bayes-lw")
  # maximisation step (forcing LAT to be connected to the other nodes).
  dag = hc(imputed, whitelist = data.frame(from = "LAT", to = names(dmarks)))
  bn = bn.fit(dag, imputed, method = "bayes")

}#FOR

# same, but automatically.
bn = structural.em(ldmarks, maximize = "hc",
  maximize.args = list(whitelist = data.frame(from = "LAT", to = names(dmarks))),
  impute = "bayes-lw", fit = "bayes", start = bn, return.all = TRUE)
```

The MARKS Example, Revisited (II)

From Structural EM we get putative class assignments for the students,

```
table(bn$imputed$LAT)
```

```
 A  B
70 18
```

and parameters for the CPTs conditional on class.

```
coef(bn$fitted$ANL)
```

```
, , LAT = A

           ALG
ANL         [15,47.5] (47.5,80]
  [9,39.5]     0.6132    0.0895
  (39.5,70]    0.3868    0.9105

, , LAT = B

           ALG
ANL         [15,47.5] (47.5,80]
  [9,39.5]     0.6644    0.5000
  (39.5,70]    0.3356    0.5000
```

Imputing Missing Data

Imputing missing values in an incomplete data set implies:

- replacing them with their posterior expectations or maximum a posteriori estimates in a Bayesian setting;
- replacing them with their maximum likelihood estimates, possibly using their parents, in a frequentist setting.

In both cases:

- we need a fully specified BN to do it;
- it is preferable to learn the BN in a Bayesian/frequentist way to perform imputation in a Bayesian/frequentist way;
- all the information needed to make inference on each node is included in its Markov blanket, so we do not need the rest of the BN to impute missing values for that node.

Relevant Functions in bnlearn

- inference: `cpquery()` and `cpdist()`.
- missing data: `impute()` and `structural.em()`.

Summary

- BNs are defined as probabilistic models, but with some care it is possible to use them as causal models. We need additional assumptions and latent variables are a constant source of difficult-to-debug problems.
- Inference is different for causal BNs: it focuses on simulating interventions and measuring their effects as opposed to compute conditional probabilities of events for the original BN.
- A related problem in learning BNs and performing inference is dealing with missing data by applying algorithms such as EM to these tasks.
- BNs provide a nice way to represent and reason about different patterns of missingness.
- BNs can also be used for counterfactuals, which are the key to achieving fairness in our models.

Thanks!

Any questions?