Directed Graphical Models

Gunwoong Park

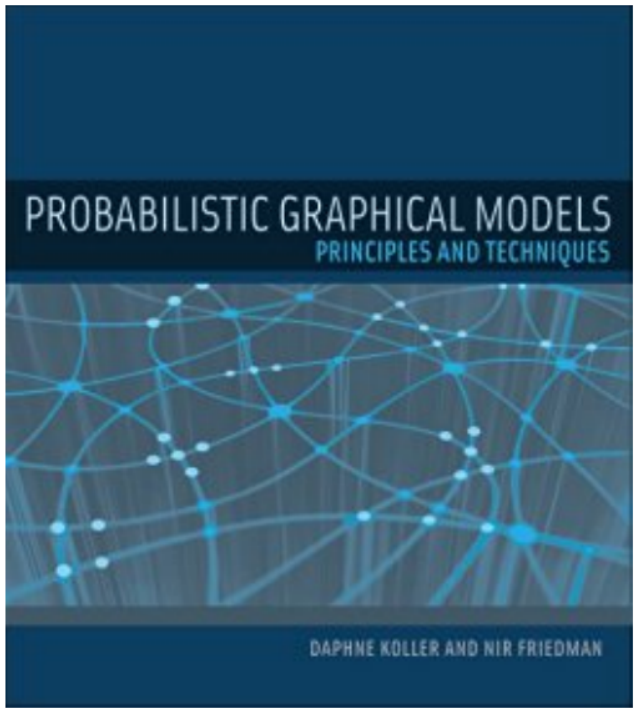
University of Seoul

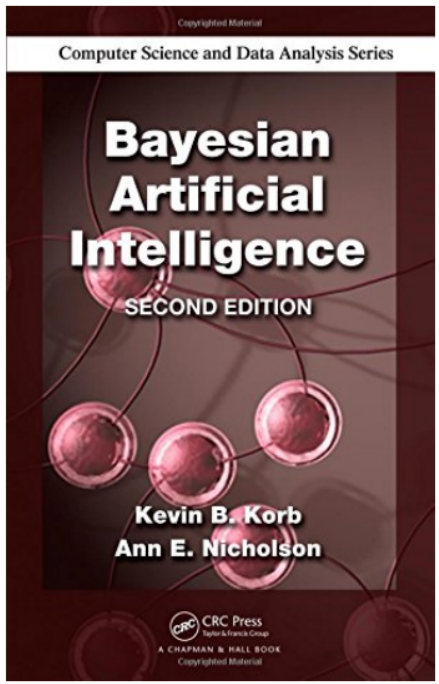
Lecture Note

- Introduction
- Definitions and Useful Notations
- Causal Inference
- Structure Learning
    - Exponential Family DAG Models
    - GHD DAG Models
    - Identifiable Gaussian SEMs

Introductions

Where to Look: Book References





How to Use: Software References

DISCLAIMER: I am the author of the **bnlearn** R package
and I will use it for the most part in this course.

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

For displaying graphs, I will use the **Rgraphviz** from BioConductor:

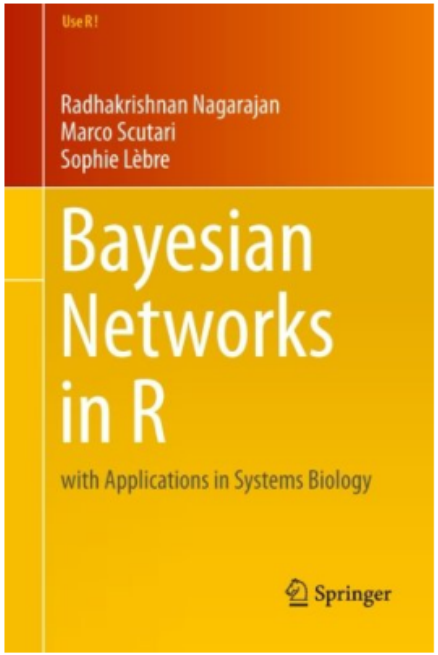
```
source("http://bioconductor.org/biocLite.R")
biocLite(c("graph", "Rgraphviz"))
```

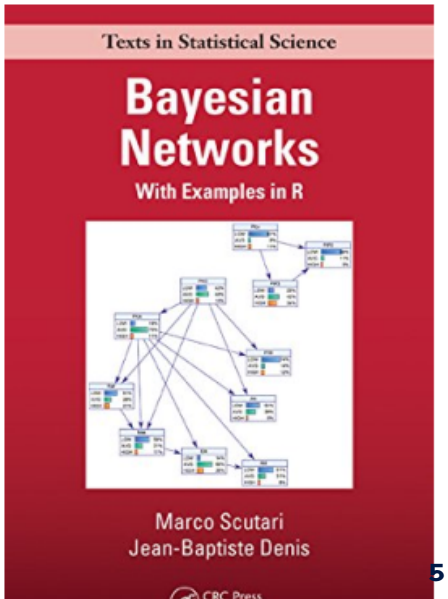
For exact inference on discrete Bayesian networks:

```
source("http://bioconductor.org/biocLite.R")
biocLite(c("graph", "Rgraphviz", "RBGL"))
install.packages("gRain")
```

Other packages from CRAN:

```
install.packages(c("pcalg", "catnet", "abn"))
```





Definitions and Useful Notations

Graph and a Probability Distribution

Bayesian networks (BNs) are defined by:

- a network structure, a directed acyclic graph $G = (V, E)$, in which each node $i \in V$ corresponds to a random variable X_i
- a global probability distribution $X = (X_1, X_2, ..., X_p)$ with parameters Θ, which can be factorized into smaller local probability distributions according to the edges $(i, j) \in E$ 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 factorization of the global distribution:

$$P(X) = \prod_{j=1}^{p} P(X_j \mid X_{\mathrm{Pa}(j)}; \Theta_{X_j}).$$

Graphs

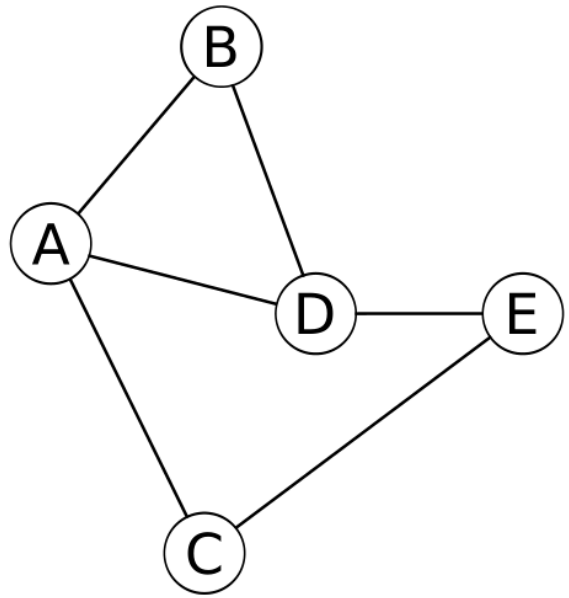
The first component of a BN is a graph. A graph G is a mathematical object with:

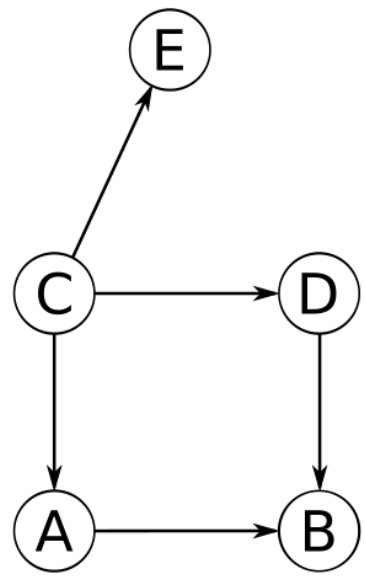
- set of nodes $V = \{v_1, ..., v_N\}$;
- a set of edges E which are identified by

pairs for nodes in V, e.g. $a_{ij} = (v_i, v_j)$. Given V, a graph is uniquely identified by E. The edges in E can be:

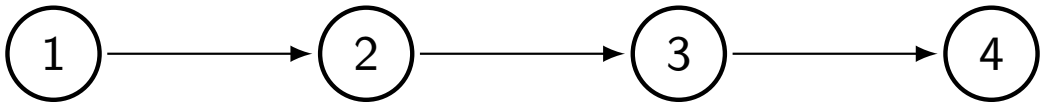
- undirected if (v_i, v_j) is an unordered pair and the edge $v_i - v_j$ has no direction;
- directed if $(v_i, v_j) \neq (v_j, v_i)$is an ordered pair and the edge has a specific direction $v_i \rightarrow v_j$.

The assumption is that there is at most one edge between a pair of nodes.





Graphs Notations

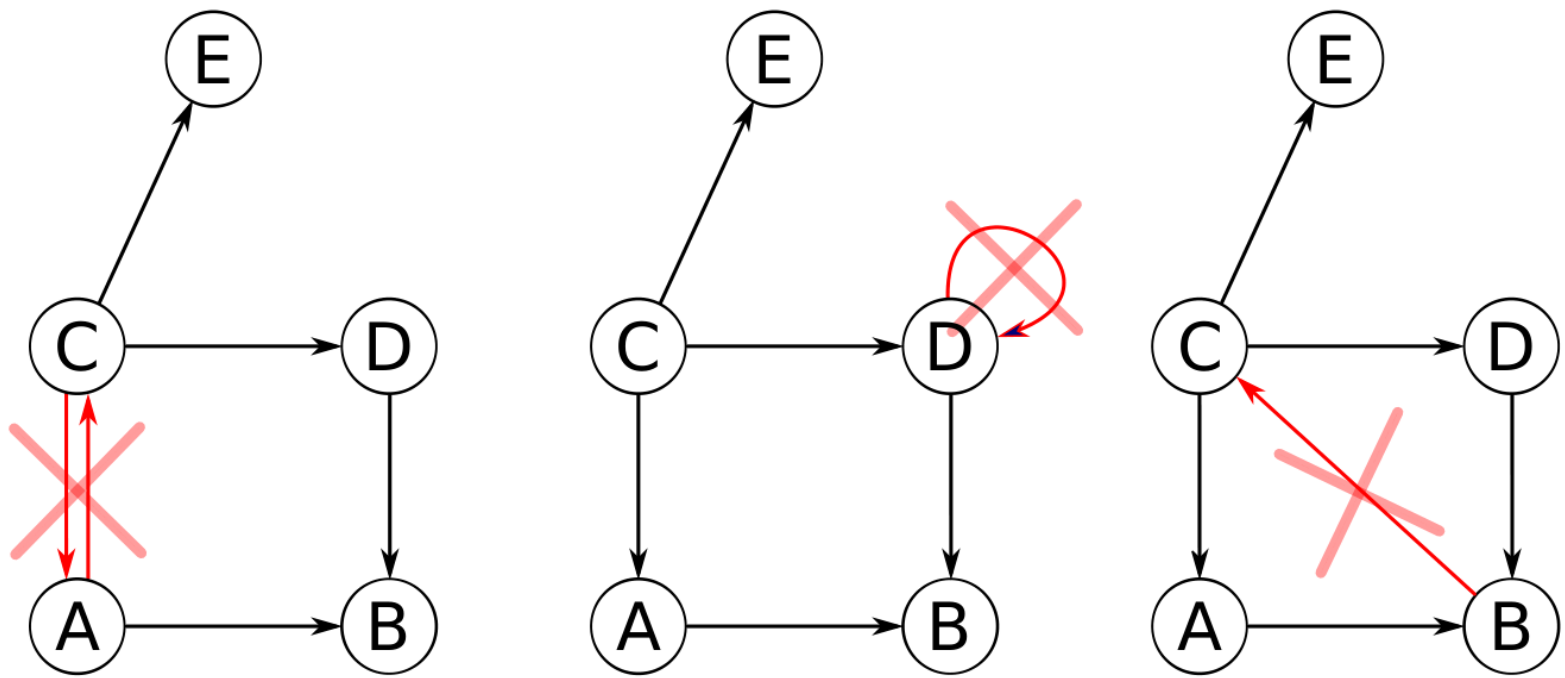


- The set of *parents* of node k, denoted by $\mathrm{Pa}(k)$, consists of all nodes j such that $(j,k) \in E$, e.g., $Pa(2) = \{1\}$.
- If there is a directed path $j \to \cdots \to k$, then k is called a *descendant* of j, and j is an *ancestor* of k. The set $\mathrm{De}(k)$ denotes the set of all descendants of node k and the set $\mathrm{An}(k)$ denotes the set of all ancestors of node k, e.g., $De(2) = \{3,4\}$
- The *non-descendants* of node k are $\mathrm{Nd}(k) := V \setminus (\{k\} \cup \mathrm{De}(k))$, e.g., $Nd(2) = \{1\}$.

Directed Acyclic Graphs

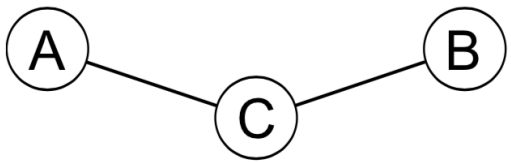
BNs use a specific kind of graph called a directed acyclic graph, that:

- contains only directed edges;
- does not contain any loop (e.g. an edge $v_i \rightarrow v_i$ from a node to itself);
- does not contain any cycle (e.g. a sequence of edges $v_i \rightarrow v_j \rightarrow ... \rightarrow v_k \rightarrow v_i$ that starts and ends in the same node).



Graphical Separation (Fundamental Connections)

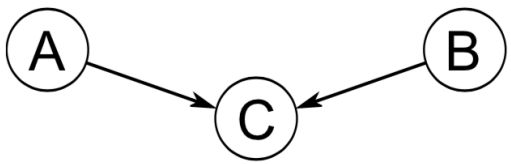
separation (undirected graphs)



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

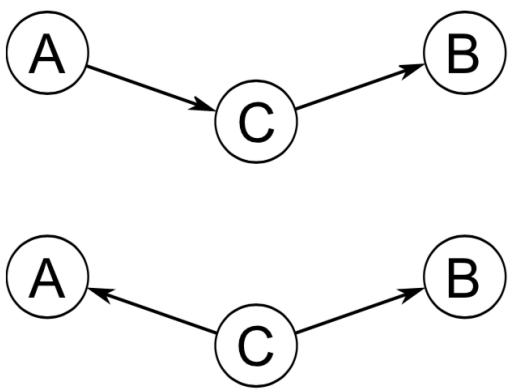
$$\mathrm{P}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \mathrm{P}(\mathbf{A} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{B} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{C})$$

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 A and B for a given C; this is how d-separation is defined.

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

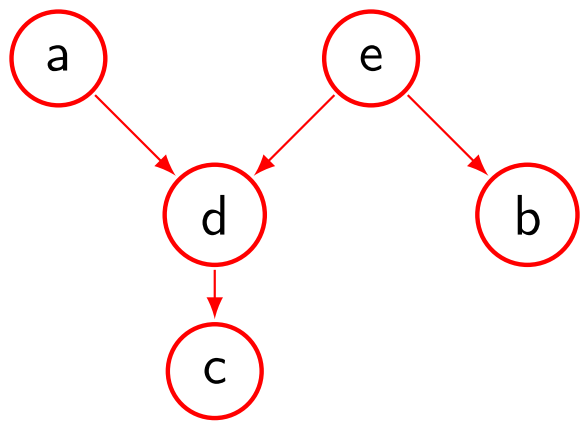
This definition clearly does not provide a computationally feasible approach to assess d-separation; but there are other ways.

Summary of D-separation

If all paths are blocked, then A is said to be d-separated from B by C. Otherwise, A is said to be d-connected to B given C.

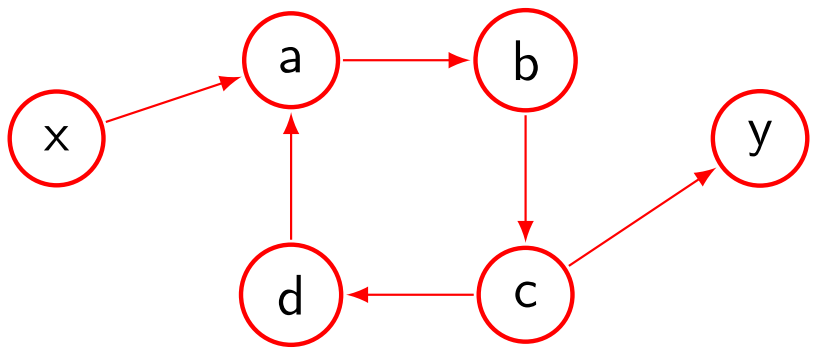
D-Separation Example

- The path from a to b is blocked.
- The path from a to b is not blocked by d.
- The path from a to b is not blocked by c.
- The path from a to b is blocked by e.

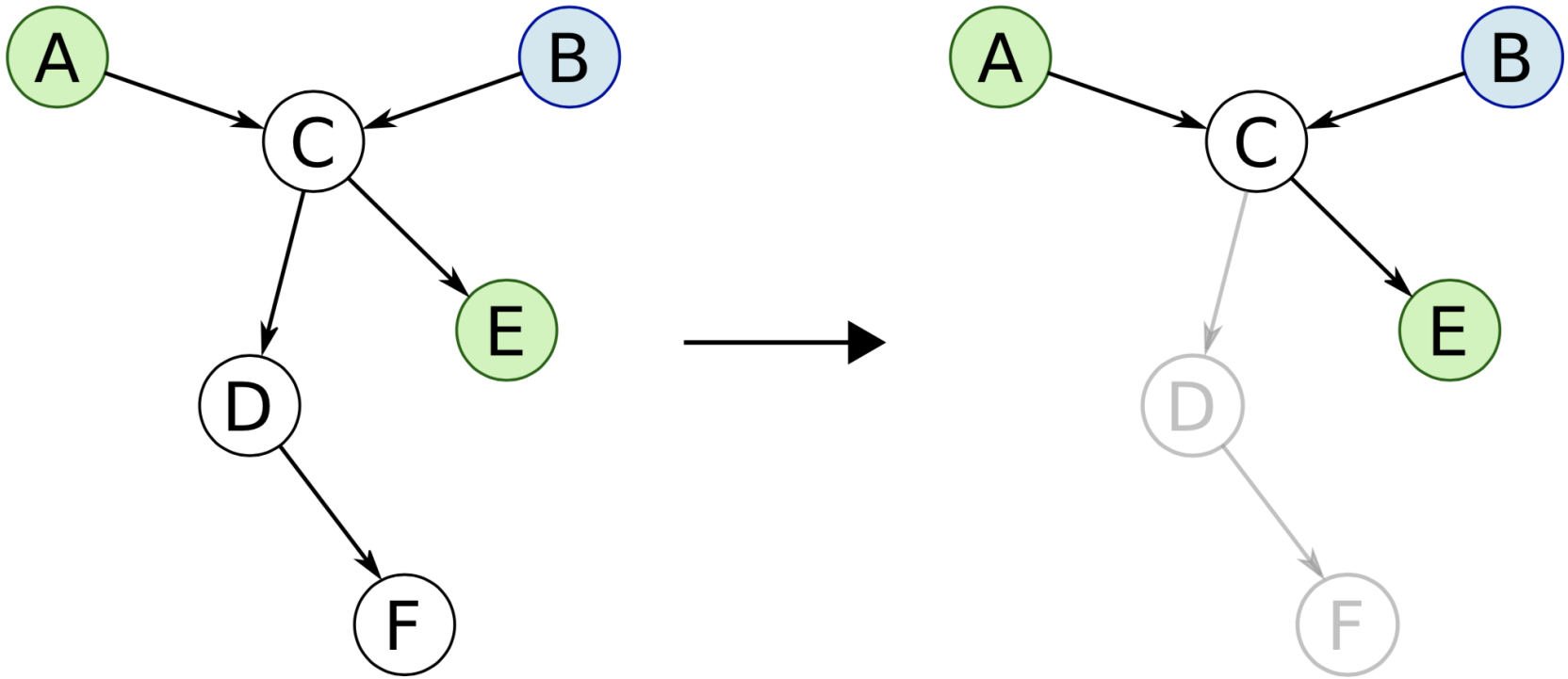


D-Separation Example in Directed Cyclic Graphs

- The path from x to d is not blocked by a.
- The path from x to d is not blocked by b.
- The path from x to d is (not) blocked.

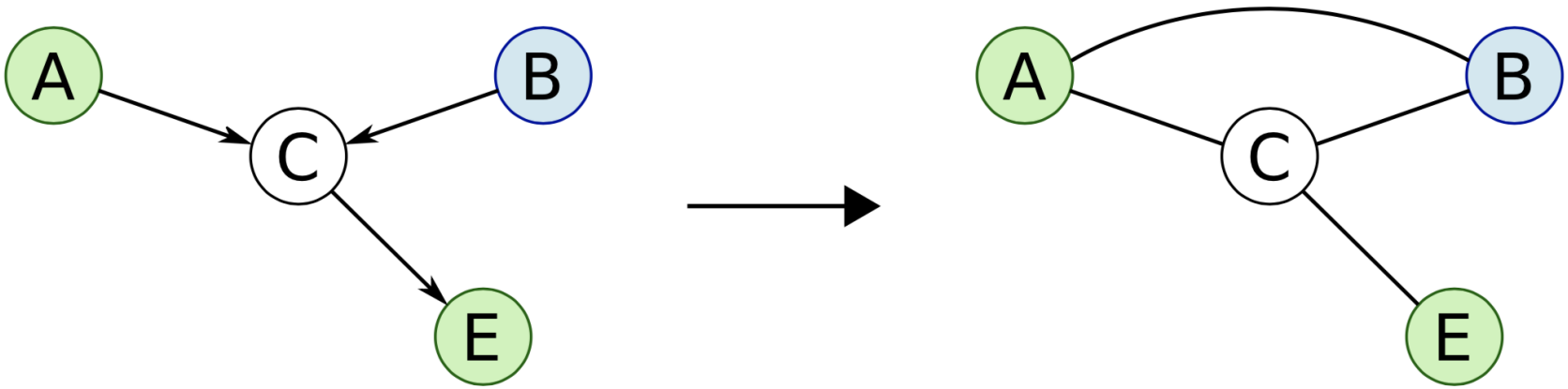


A Simple Algorithm to Check D-Separation (I)



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

A Simple Algorithm to Check D-Separation (II)

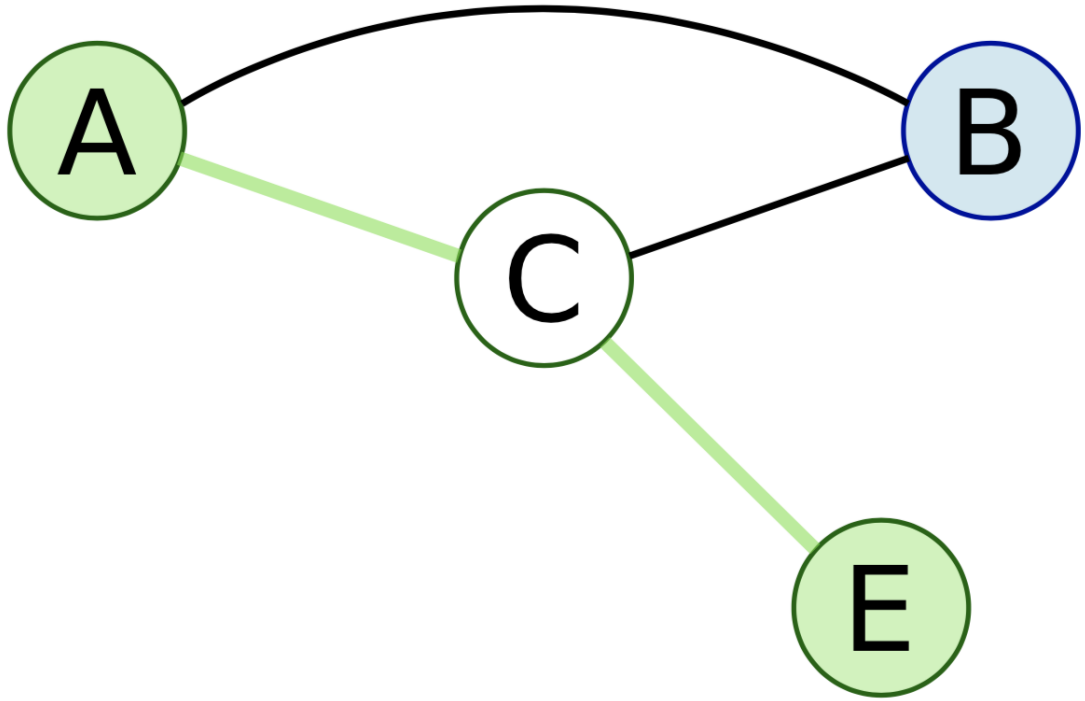


Transform the subgraph into its moral graph by

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

This transformation has the double effect of making the dependence between parents explicit by "marrying" them and of allowing us to use the classic definition of graphical separation.

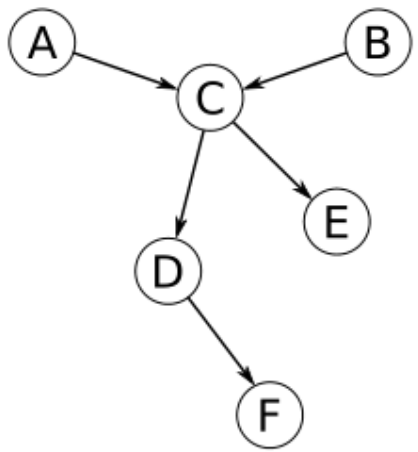
A Simple Algorithm to Check D-Separation (III)



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

How to connect a DAG and the Probability Distribution

DAG



$\Longrightarrow$

Graphical separation

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

$\Longrightarrow$

Probabilistic independence

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

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

Maps

Let M be the dependence structure of the probability distribution P of X, that is, the set of conditional independence relationships linking any triplet A, B, C of subsets of X. A graph G is a dependency map (or D-map) of M if there is a one-to-one correspondence between the random variables in X and the nodes V of G such that for all disjoint subsets A, B, C of X we have

$$A \perp\!\!\!\perp_P B \mid C \longrightarrow A \perp\!\!\!\perp_G B \mid C.$$

Similarly, G is an independence map (or I-map) of M if

$$A \perp\!\!\!\perp_P B \mid C \longleftarrow A \perp\!\!\!\perp_G B \mid C.$$

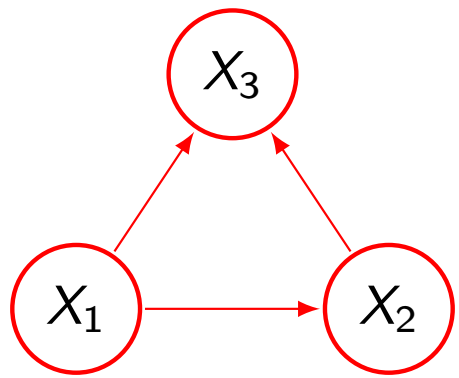
G is said to be a perfect map of M if it is both a D-map and an I-map, that is

$$A \perp\!\!\!\perp_P B \mid C \quad \Longleftrightarrow \quad A \perp\!\!\!\perp_G B \mid C.$$

and in this case G is said to be faithful or isomorphic to M.

Faithfulness Assumption

- The faithfulness assumption may not be satisfied even in population. It implies that the sample version of faithfulness assumption is extremely strong in finite sample settings.



It can be explained by partial correlations.

$$X_1 = \epsilon_1, \quad X_2 = X_1 + \epsilon_2, \quad X_3 = X_1 + X_2 + \epsilon_3$$

where $(\epsilon_i)_{i=1}^3 \sim N(0, 1)$.

The Local Markov Property (I)

If we use d-separation as our definition of graphical separation, assuming that the DAG is an I-map leads to the general formulation of the decomposition of the global distribution $P(X)$:

$$P(X) = \prod_{j=1}^{N} P(X_j \mid X_{\mathrm{Pa}(j)})$$

into the local distributions for the X_j given their parents $X_{\mathrm{Pa}(j)}$.

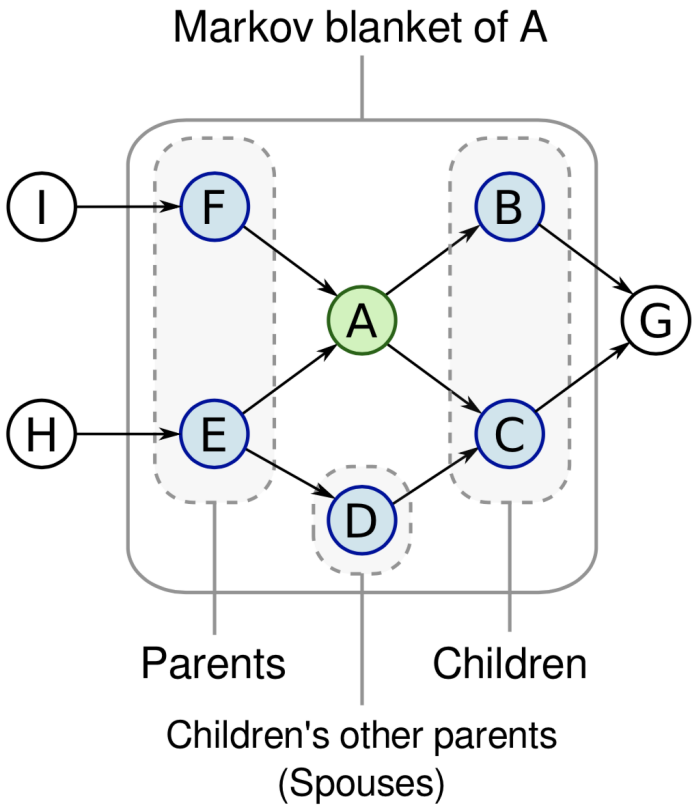
The Local Markov Property (II)

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

- Each node X_j is conditionally independent of its non-descendants (e.g., nodes X_k for which there is no path from X_j to X_k) given its parents.

Compared to the previous decomposition, it highlights the fact that parents are not completely independent from their children in the BN

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.

Different DAGs, Same Distribution: Topological Ordering

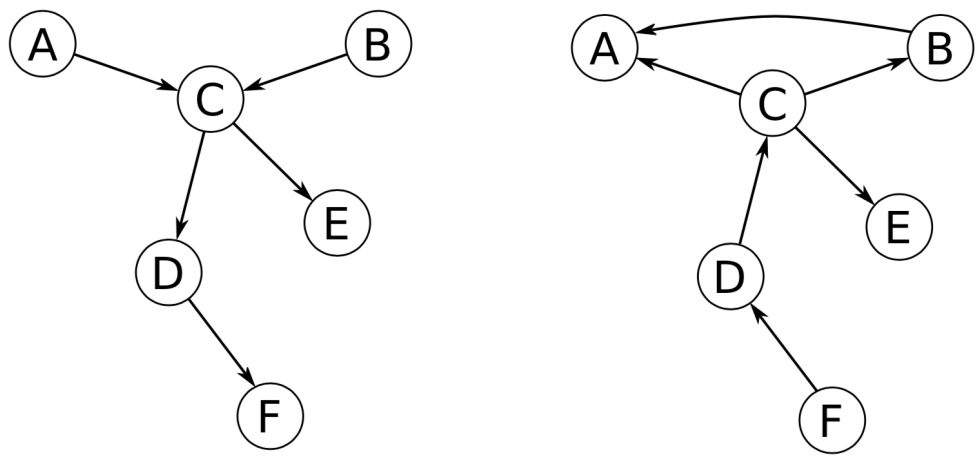
A DAG uniquely identifies a factorization of $P(X)$; the converse is Not necessarily true. Consider again the DAG on the left:

$$P(X) = P(A)P(B)P(C \mid A, B)P(D \mid C)P(E \mid C)P(F \mid D).$$

We can rearrange the dependencies using Bayes theorem to obtain:

$$P(X) = P(A \mid B, C)P(B \mid C)P(C \mid D)P(D \mid F)P(E \mid C)P(F),$$

which gives the DAG on the right, with a different topological ordering.



Different DAGs, Same Distribution: Equivalence Classes

On a smaller scale, even keeping the same underlying undirected graph we can reverse a number of edges without changing the dependence structure of X. Since the triplets $A \to B \to C$ and $A \leftarrow B \to C$ are probabilistically equivalent, we can reverse the directions of their edges as we like as long as we do not create any new v-structure ($A \to B \leftarrow C$, with no edge between A and C).

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 edges can be either:

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

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

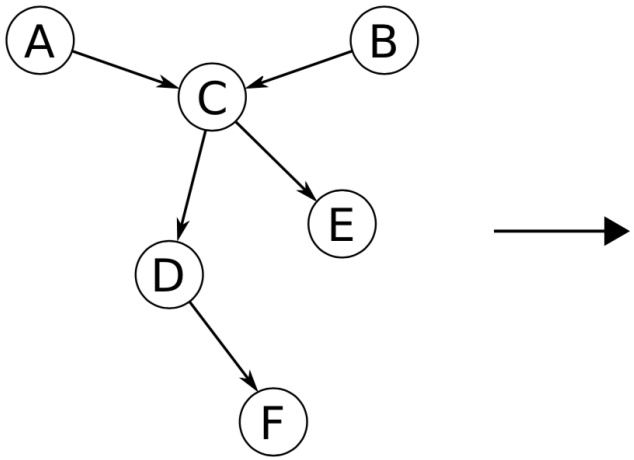
What Are V-Structures, and What Are Not

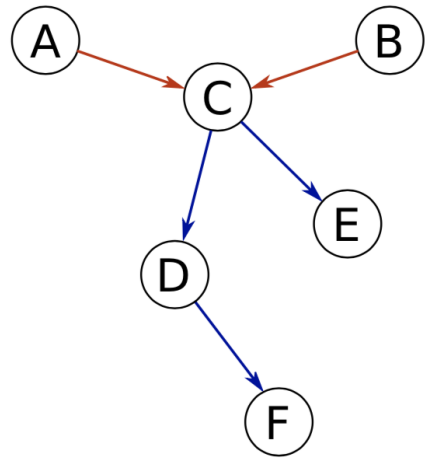
It is important to note that even though $A \to B \leftarrow C$ is a convergent connection, it is not a v-structure if A and C are connected by $A \to C$. As a result, we are no longer able to identify which nodes are the parents in the connection. For example:

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

Therefore, the fact that the two parents in a convergent connection are not connected by an edge (v-structure) is crucial in the identification of the correct CPDAG.

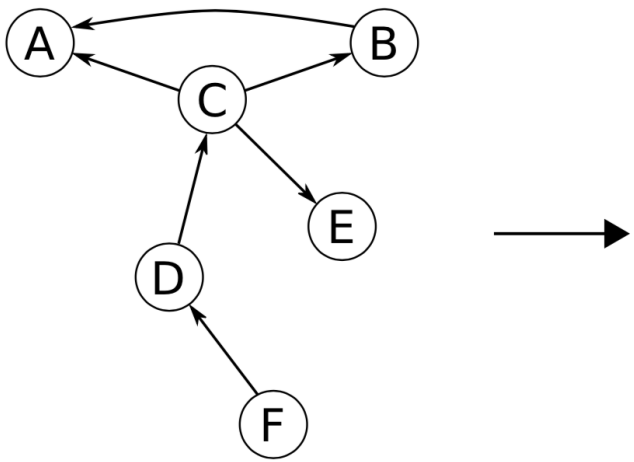
Completed Partially Directed Acyclic Graphs (CPDAGs)

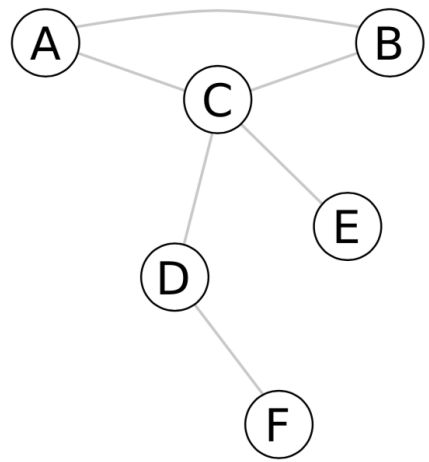




DAG

CPDAG





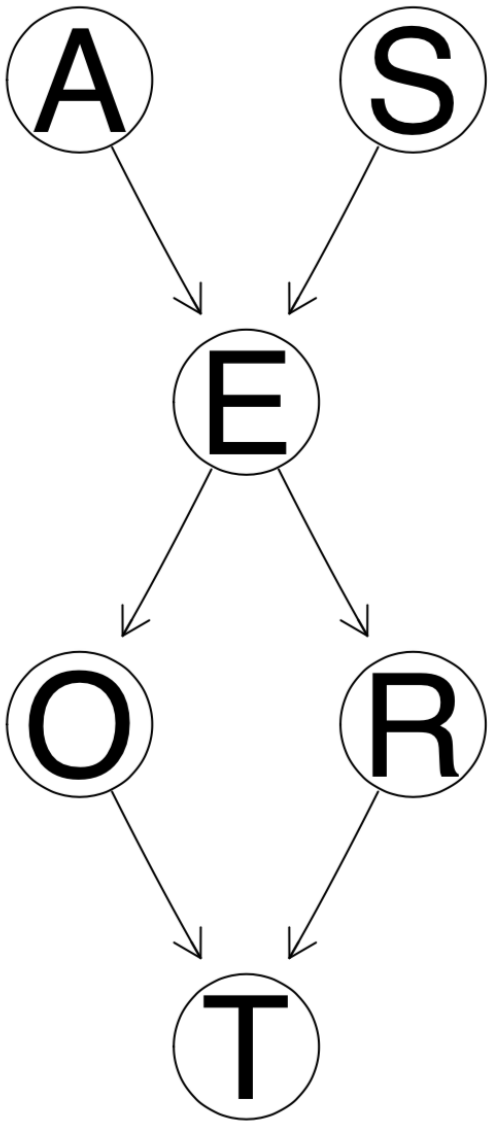
An Example: Train Use Survey

Consider a simple, hypothetical survey whose aim is to investigate the usage patterns of different means of transport, with a focus on cars and trains.

- Age (A): young for individuals below 30 years old, adult for individuals between 30 and 60 years old, and *old* for people older than 60.
- Sex (S): *male* or *female*.
- Education (E): *up to high school* or *university degree*.
- Occupation (O): *employee* or *self-employed.*
- Residence (R): the size of the city the individual lives in, recorded as either *small* or *big.*
- Travel (T): the means of transport favoured by the individual, recorded either as *car*, *train* or *other*.

The nature of the variables recorded in the survey suggests how they may be related with each other.

The Train Use Survey as a BN (I)

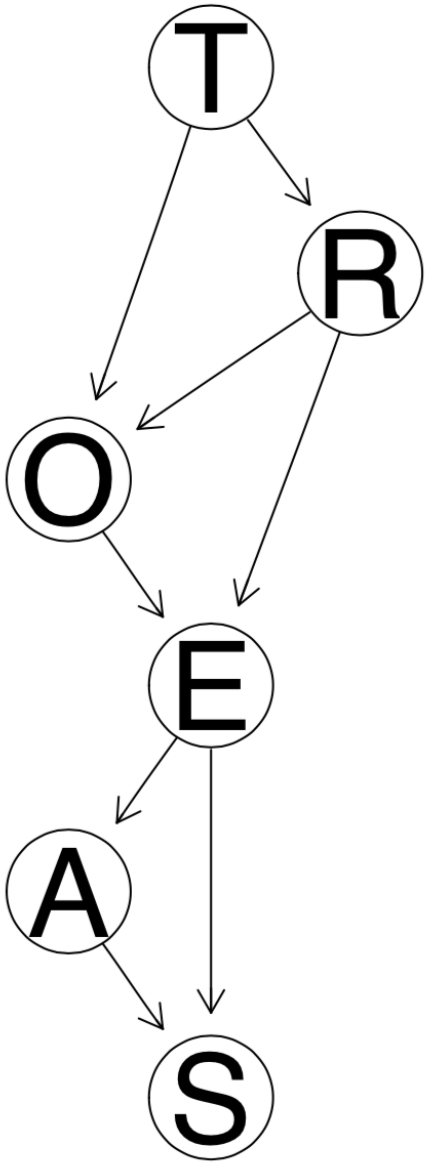


That is a prognostic view of the survey as a BN:

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

Variables that can be thought as "causes" are on above variables that can be considered their "effect", and con-founders are on above everything else.

The Train Use Survey as a BN (II)



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. For discrete BNs, one representation may have fewer parameters than the other.

bnlearn: Creating Graphs (I)

- Setting individual edges.

```
survey.dag = empty.graph(nodes = c("A", "S", "E", "O", "R", "T"))
survey.dag = set.edge(survey.dag, from = "A", to = "E")
survey.dag = set.edge(survey.dag, from = "S", to = "E")
survey.dag = set.edge(survey.dag, from = "E", to = "O")
survey.dag = set.edge(survey.dag, from = "E", to = "R")
survey.dag = set.edge(survey.dag, from = "O", to = "T")
survey.dag = set.edge(survey.dag, from = "R", to = "T")
```

- Setting the whole edge set at once.

```
edge.set = matrix(c("A", "E",
    "S", "E",
    "E", "O",
    "E", "R",
    "O", "T",
    "R", "T"),
    byrow = TRUE, ncol = 2,
    dimnames = list(NULL, c("from", "to")))
edges(survey.dag) = edge.set
```

bnlearn: Creating Graphs (II)

- Using the adjacency matrix representation of the edge set.

```
amat(survey.dag) =
matrix(c(0L, 0L, 1L, 0L, 0L, 0L,
    0L, 0L, 1L, 0L, 0L, 0L,
    0L, 0L, 0L, 1L, 1L, 0L,
    0L, 0L, 0L, 0L, 0L, 1L,
    0L, 0L, 0L, 0L, 0L, 1L,
    0L, 0L, 0L, 0L, 0L, 0L),
    byrow = TRUE, nrow = 6, ncol = 6,
    dimnames = list(nodes(survey.dag), nodes(survey.dag)))
```

- Using the formula representation for the Bayesian network.

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

Acyclicity is enforced by all there functions by default, e.g.,

```
set.edge(survey.dag, from = "T", to = "E")
## Error in edge.operations(x = x, from = from, to = to, op = "set",
check.cycles = check.cycles, :  the resulting graph contains cycles.
```

bnlearn: BN graph objects

```
survey.dag
##
##    Random/Generated Bayesian network
##
##    model:
##      [A][S][E|A:S][O|E][R|E][T|O:R]
##    nodes:                                 6
##    edges:                                  6
##       undirected edges:                    0
##       directed edges:                      6
##    average markov blanket size:           2.67
##    average neighbourhood size:            2.00
##    average branching factor:              1.00
##
##    generation algorithm:                  Empty
```

This is what the graph structure of BN looks like when printed: note the model formula, which is the same as that you would pass to model2network(). Additional information will be printed as well if the graph is learned from data.

bnlearn: Manipulating Graphs

- Adding, removing and reversing edges.

```
survey.dag = set.edge(survey.dag, from = "A", to = "O")
survey.dag = drop.edge(survey.dag, from = "E", to = "O")
survey.dag = reverse.edge(survey.dag, from = "R", to = "E")
```

- Finding the skeleton (the underlying undirected graph).

```
Finding the skeleton (the underlying undirected graph).
```

- Finding the moral graph.

```
moral(survey.dag)
```

- Extracting a subgraph.

```
subgraph(survey.dag)
```

Plus many others...

bnlearn: Investigating Graphs (I)

- Sets of nodes close to a target node (here E).

```
mb(survey.dag, "E")
## [1] "A" "O" "R" "S"
nbr(survey.dag, "E")
## [1] "A" "O" "R" "S"
parents(survey.dag, "E")
## [1] "A" "S"
children(survey.dag, "E")
## [1] "O" "R"
```

- Roots (no parents) and leaves (no children).

```
root.nodes(survey.dag)
## [1] "A" "S"
leaf.nodes(survey.dag)
## [1] "T"
```

bnlearn: Investigating Graphs (II)

- Directed and undirected edges.

```
directed.edges(survey.dag)
##      from to
## [1,] "A"  "E"
## [2,] "S"  "E"
## [3,] "E"  "O"
## [4,] "E"  "R"
## [5,] "O"  "T"
## [6,] "R"  "T"
undirected.edges(survey.dag)
##      from to
```

- Different graph representations.

```
edges(survey.dag)
amat(survey.dag)
```

- Looking for paths.

```
path(survey.dag, from = "A", to = "T")
## [1] TRUE
```

bnlearn: D-Separation and Markov Blankets

The dsep() and mb() functions can be used to show how d-separation and Markov blankets interact in practice. Firstly, node that a node is never part of its own Markov blanket.

```
mbE = mb(survey.dag, "E")
"E" %in% mbE
## [1] FALSE
```

Secondly, note that the Markov blanket is minimal and that it makes all other nodes independent of the target node.

```
for (node in mbE)
print(dsep(survey.dag, "E", node, setdiff(mbE, c("E", node))))
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
for (node in setdiff(nodes(survey.dag), c("E", mbE)))
print(dsep(survey.dag, "E", node, mbE))
## [1] TRUE
```

bnlearn: Moral Graphs and CPDAGs

There are functions to compute them:

```
moral(survey.dag)
cpdag(survey.dag)
```

And if we go back to the survey example, we find that all edges are compelled and that the CPDAG is identical to the original DAG.

```
all.equal(cpdag(survey.dag), survey.dag)
## [1] TRUE
compelled.edges(survey.dag)
##      from to
## [1,] "A"  "E"
## [2,] "E"  "O"
## [3,] "E"  "R"
## [4,] "O"  "T"
## [5,] "R"  "T"
## [6,] "S"  "E"
```

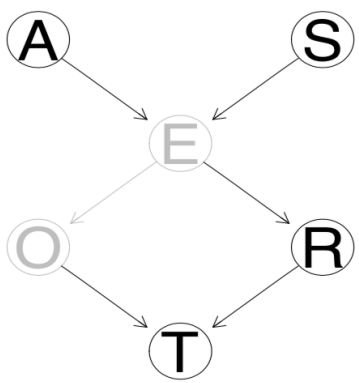
And we can observe that:

```
all.equal(compelled.edges(survey.dag), directed.edges(cpdag(survey.dag)))
## [1] TRUE
```

bnlearn: Plotting Graphs

bnlearn uses the functionality implemented in the **Rgraphviz** package to plot graphs, through the graphviz.plot function.

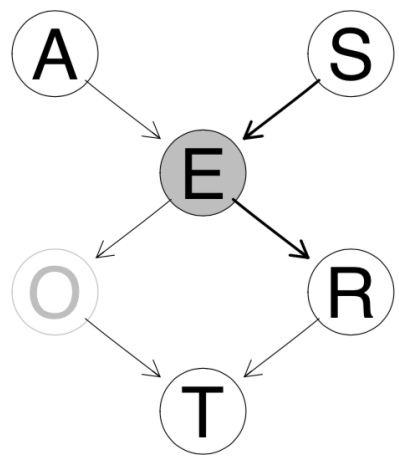
```
hlight = list(nodes = c("E", "O"),
edges = c("E", "O"),
col = "grey",
textCol = "grey")
pp = graphviz.plot(survey.dag,
highlight = hlight)
```



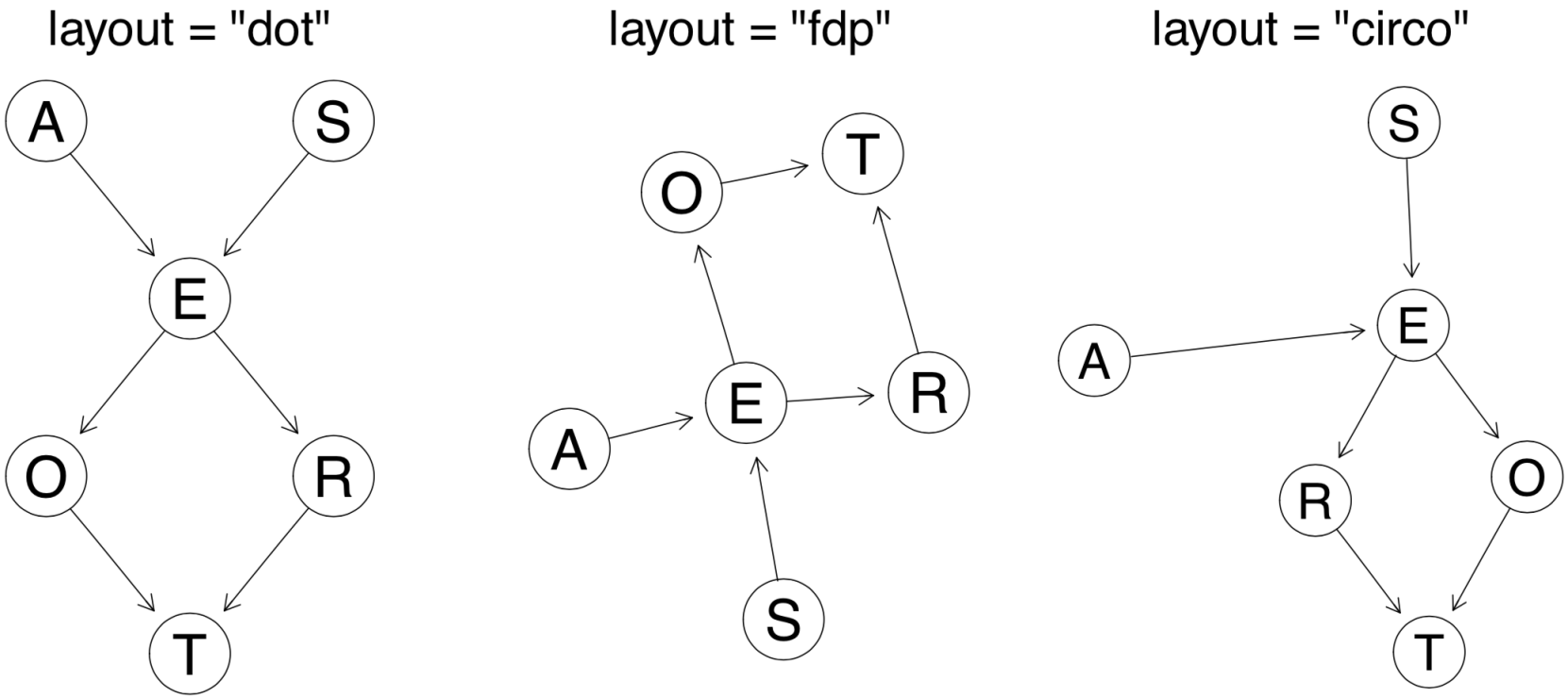

```
edgeRenderInfo(pp) =
list(col = c("S~E" = "black",
"E~R" = "black"),
lwd = c("S~E" = 3, "E~R" = 3))
```

```
nodeRenderInfo(pp) =
list(col =
c("S" = "black", "E" = "black",
"R" = "black"),
textCol =
c("S" = "black", "E" = "black",
"R" = "black"),
fill = c("E" = "grey"))

renderGraph(pp)
```

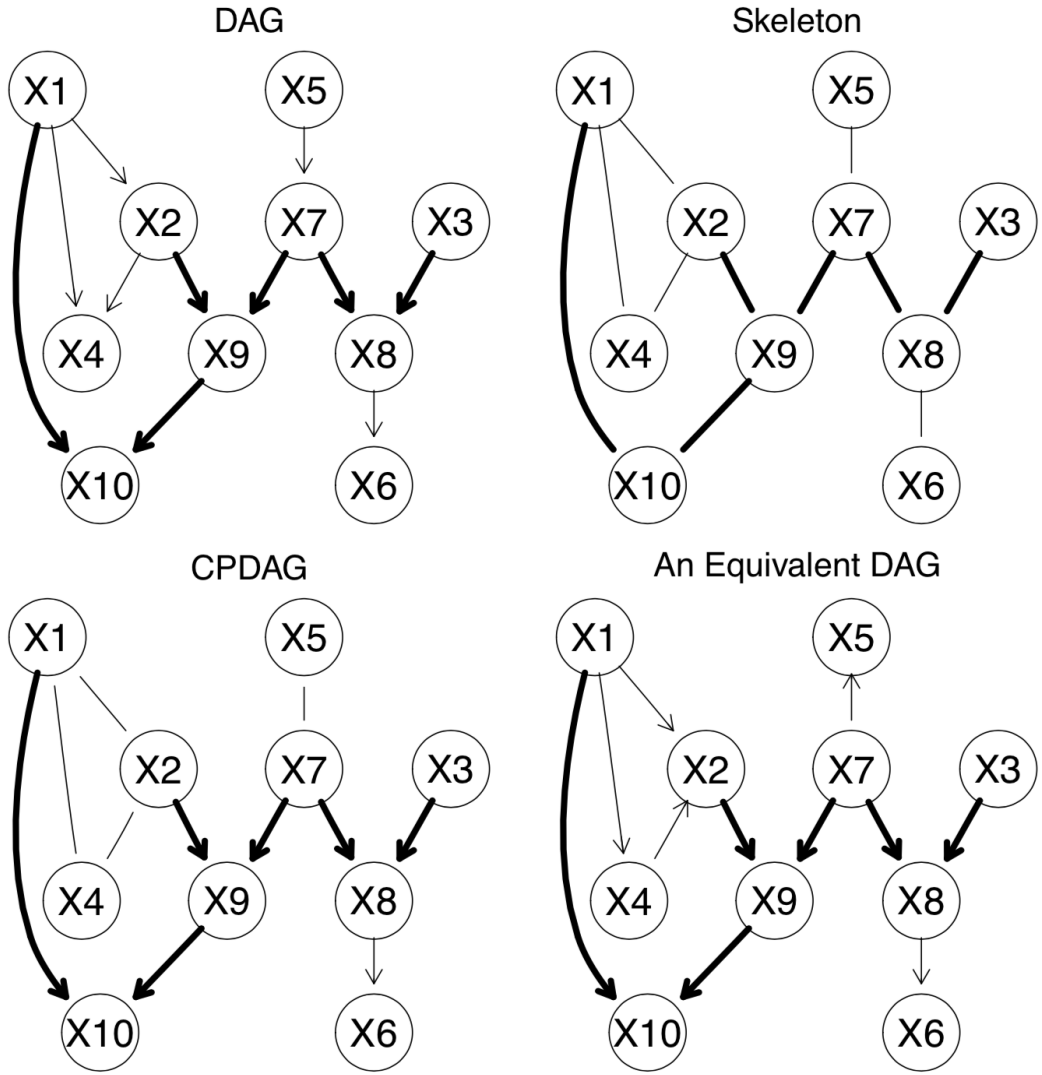


Different Layouts Available in Rgraphviz

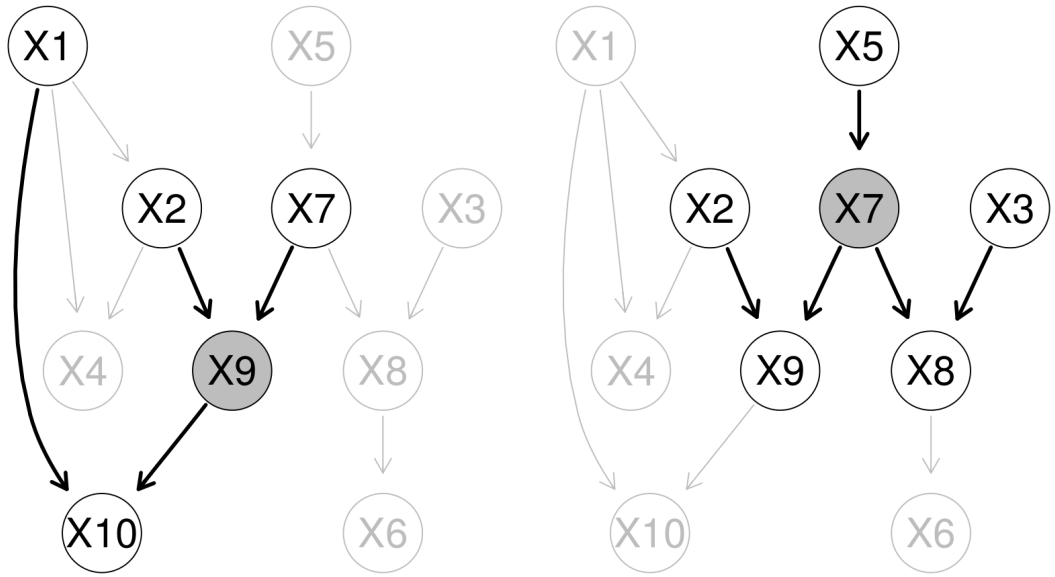


NOTE: unlike **igraph** we cannot rearrange the layout of the nodes, which makes plotting graphs with the same node positions but different edges very difficult.

Another Example, from the C&H Book (I)



Another Example, from the C&H Book (II)



Another Example, from the C&H Book (III)

We can verify again that the Markov blanket contains the children, the parents and the spouses of the node it is centred on; and that it does not contain that node.

```
M = paste("[X1][X3][X5][X6|X8][X2|X1][X7|X5][X4|X1:X2]",
"[X8|X3:X7][X9|X2:X7][X10|X1:X9]", sep = "")
dag = model2network(M)
mb(dag, node = "X9")
## [1] "X1"  "X10" "X2"  "X7"
par.X9 = parents(dag, node = "X9")
ch.X9 = children(dag, node = "X9")
sp.X9 = sapply(ch.X9, parents, x = dag)
sp.X9 = sp.X9[sp.X9 != "X9"]
unique(c(par.X9, ch.X9, sp.X9))
## [1] "X2"  "X7"  "X10" "X1"
```

Another Example, from the C&H Book (IV)

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.

```
sapply(nodes(dag), function(node) node %in% mb(dag, node = "X9"))
##    X1   X10    X2    X3    X4    X5    X6    X7    X8    X9
##  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
sapply(nodes(dag), function(node) "X9" %in% mb(dag, node = node))
##    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.

What About the Probability Distributions?

The second component of a BN is the probability distribution $P(X)$. The choice should such that the BN:

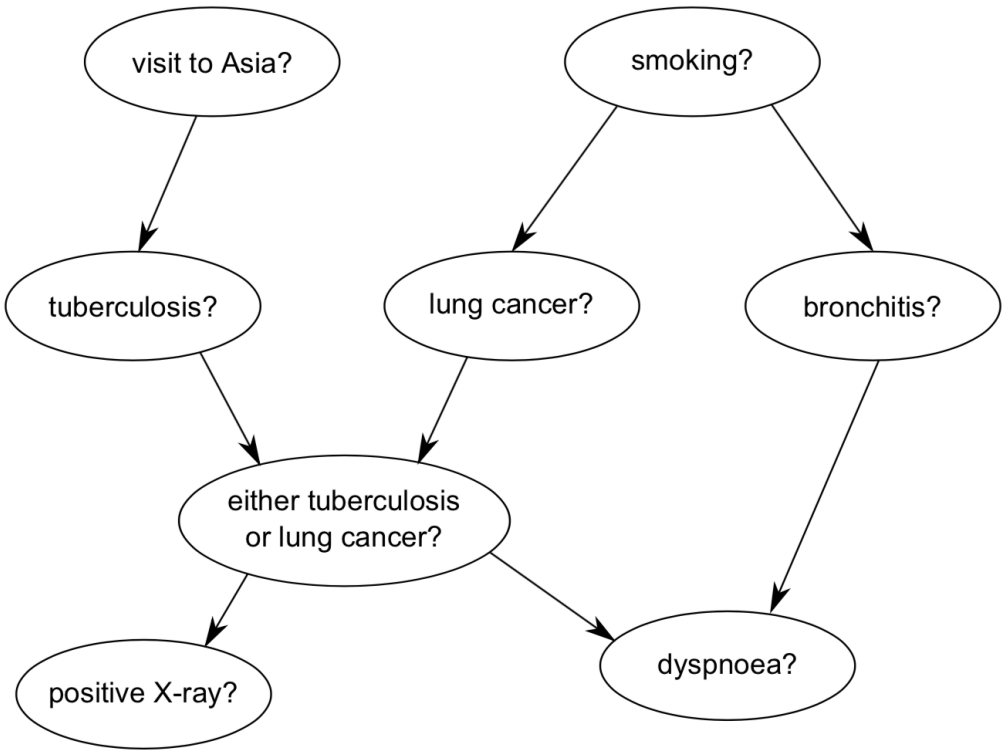
- can be learned efficiently from data;
- is flexible (distributional assumptions should not be too strict);
- is easy to query to perform inference.

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

- Discrete BNs (DBNs), in which X and the $X_j \mid X_{\mathrm{Pa}(j)}$ are multinomial;
- Gaussian BNs (GBNs), in which X is multivariate normal and the $X_j \mid X_{\mathrm{Pa}(j)}$ are univariate normal;
- Conditional linear Gaussian BNs (CLGBNs), in which X is a mixture of multivariate normals, and $X_j \mid X_{\mathrm{Pa}(j)}$ are either multinomial, univariate normal or mixtures of normals.
- Count BNs (CBNs), in which X and the $X_j \mid X_{\mathrm{Pa}(j)}$ are Poisson;

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

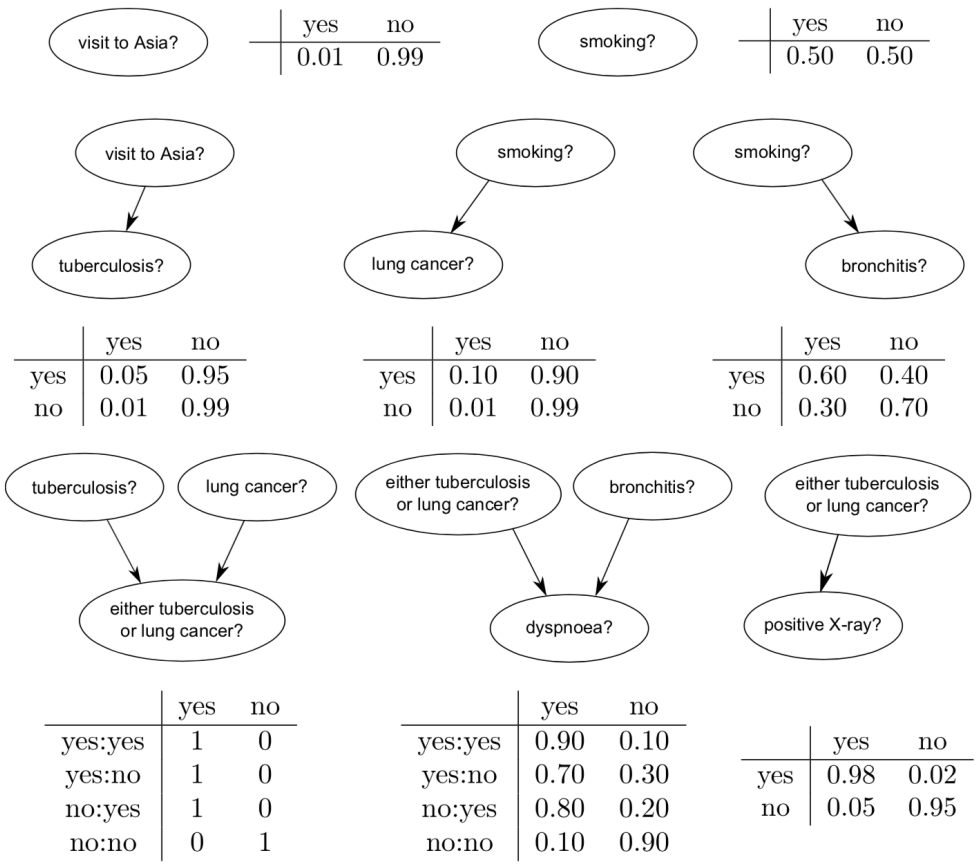
Discrete BNs



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 X :
$2^8 - 1 = 255$

Conditional Probability Tables (CPTs)



The local distributions $X_j \mid X_{\mathrm{Pa}(j)}$ take the form of conditional probability tables for each node given all the configurations of the values of its parents.

Overall parameters of the $X_j \mid X_{\mathrm{Pa}(j)}$: 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)
bn = custom.fit(asia.dag, cpt)
```

bnlearn: Conditional Probability Tables (I)

```
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
```

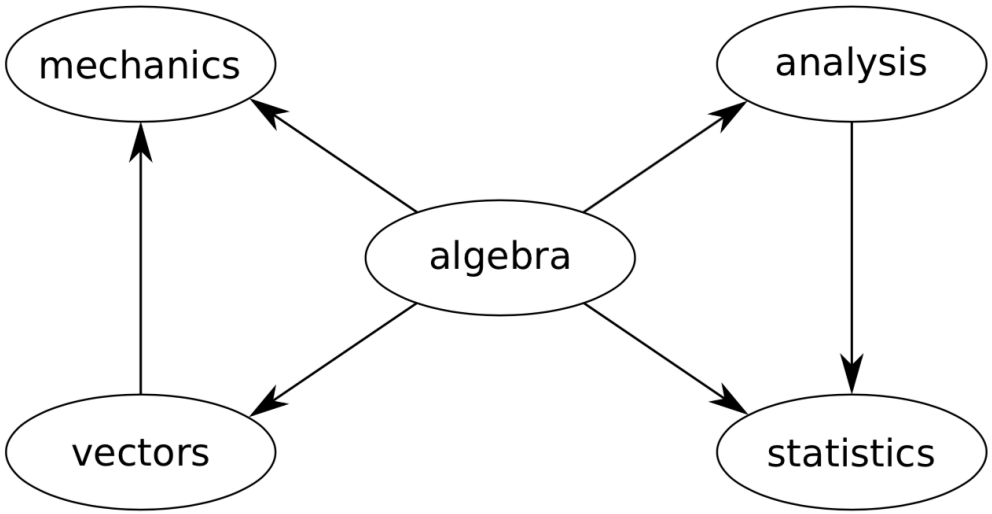
bnlearn: Creating a Discrete BN (Survey)

```
A.lv = c("young", "adult", "old")
S.lv = c("M", "F")
E.lv = c("high", "uni")
O.lv = c("emp", "self")
R.lv = c("small", "big")
T.lv = c("car", "train", "other")
A.prob = array(c(0.30, 0.50, 0.20), dim = 3, dimnames = list(A = A.lv))
S.prob = array(c(0.60, 0.40), dim = 2, dimnames = list(S = S.lv))
O.prob = array(c(0.96, 0.04, 0.92, 0.08), dim = c(2, 2),
dimnames = list(O = O.lv, E = E.lv))
R.prob = array(c(0.25, 0.75, 0.20, 0.80), dim = c(2, 2),
dimnames = list(R = R.lv, E = E.lv))
E.prob = array(c(0.75, 0.25, 0.72, 0.28, 0.88, 0.12, 0.64,
0.36, 0.70, 0.30, 0.90, 0.10), dim = c(2, 3, 2),
dimnames = list(E = E.lv, A = A.lv, S = S.lv))
T.prob = array(c(0.48, 0.42, 0.10, 0.56, 0.36, 0.08, 0.58,
0.24, 0.18, 0.70, 0.21, 0.09), dim = c(3, 2, 2),
dimnames = list(T = T.lv, O = O.lv, R = R.lv))
cpt = list(A = A.prob, S = S.prob, E = E.prob, O = O.prob,
R = R.prob, T = T.prob)
bn = custom.fit(survey.dag, cpt)
```

bnlearn: Conditional Probability Tables (II)

```
bn$T
##
##   Parameters of node T (multinomial distribution)
##
## Conditional probability table:
##
## , , R = small
##
##       O
##  T           emp   self
##     car      0.48  0.56
##     train    0.42  0.36
##     other    0.10  0.08
##
## , , R = big
##
##       O
##  T           emp   self
##     car      0.58  0.70
##     train    0.24  0.21
##     other    0.18  0.09
```

Gaussian BNs



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

Assuming $X \sim N(\mu, \Sigma)$, we can compute $\Omega = \Sigma^{-1}$. Then $\Omega_{ij} = 0$ implies $X_i \perp\!\!\!\perp_P X_j \mid X \backslash \{X, X_j\}$. The absence of an edge $X_i \to X_j$ in the DAG implies $X_i \perp\!\!\!\perp_G X_j \mid X \backslash \{X, X_j\}$, which in turn implies $X_i \perp\!\!\!\perp_P X_j \mid X \backslash \{X, X_j\}$.

Total parameters of X : $5 + 15 = 20$

Partial Correlations and Linear Regressions

The local distributions $X_j \mid X_{\mathrm{Pa}(j)}$ take the form of linear regression models with the $X_{\mathrm{Pa}(j)}$ acting as regressors and with independent error terms.

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

That is because $\Omega_{ij} \propto \beta_j$ for X_i, so $\beta_j > 0$ if and only if $\Omega_{ij} > 0$. Also $\Omega_{ij} \propto \rho_{ij}$, the partial correlation between X_i and X_j, so we are implicitly assuming all probabilistic dependencies are linear.

Overall parameters of the $X_j \mid X_{\mathrm{Pa}(j)}$: $11 + 5 = 16$

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)
bn = custom.fit(marks.dag, ldist)
```

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

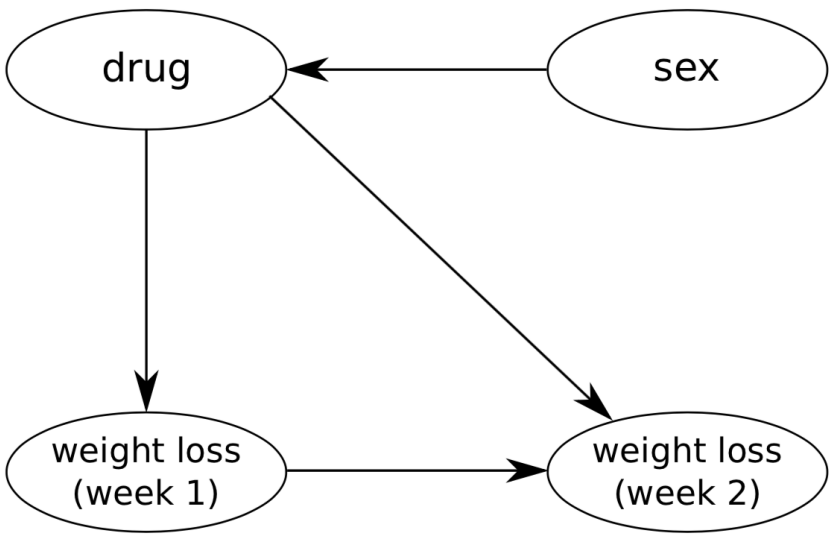
bnlearn: Local Linear Regressions

```
bn[c("MECH", "STAT")]
## $MECH
##
##   Parameters of node MECH (Gaussian distribution)
##
## Conditional density: MECH | ALG + VECT
## Coefficients:
## (Intercept)          ALG         VECT
##      -12.36         0.54         0.46
## Standard deviation of the residuals: 14
##
## $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
```

Conditional Linear Gaussian BNs

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

- 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.



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

Mixtures of Linear Regressions

The resulting local distribution for the first weight loss for drugs D_1, D_2 and D_3 is:

$$W_{1,D_1} = 7 + \varepsilon_{D_1} \sim N(0, 2.5)$$

$$W_{1,D_2} = 7.50 + \varepsilon_{D_2} \sim N(0, 2)$$

$$W_{1,D_3} = 14.75 + \varepsilon_{D_3} \sim N(0, 11)$$

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

$$W_{2,D_1} = 1.02 + 0.89\beta_{W_1} + \varepsilon_{D_1} \sim N(0, 3.2)$$

$$W_{2,D_2} = -1.68 + 1.35\beta_{W_1} + \varepsilon_{D_2} \sim N(0, 4)$$

$$W_{2,D_3} = -1.83 + 0.82\beta_{W_1} + \varepsilon_{D_3} \sim N(0, 1.9)$$

Overall, they look like random effect models with random intercepts and random slopes.

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)
bn = custom.fit(rats.dag, ldist)
```

The regression coefficients are stored in a matrix with one conditional regression in each column, so that 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
```

Limitations of These Probability Distributions

- No real-world, multivariate data set follows a multivariate Gaussian distribution; even if the marginal distributions are normal, not all dependence relationships are linear.
- Computing partial correlations is problematic in most large data sets (and in a lot of small ones, too) because of singularities.
- Parametric assumptions for mixed data have strong limitations, as they impose constraints on which edges may be present in the graph (e.g. a continuous node cannot be the parent of a discrete node).
- Discretization is a common solution to the above problems, but it may discard useful information and it is tricky to get right (i.e. 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.

Limitations of These Probability Distributions

- Other exponential family distributions cannot be applied such as Binomial, Exponential, Poisson, Hyper Poisson and many others.
- QVF DAG Models cover many exponential family distributions in DAG settings.
- GHD DAG Models cover many count distributions in a DAG language.

Equivalence and Singularity

Assuming the DAG is an I-map means that serial and divergent connections result in equivalent factorization of the variables involved. It is easy to show that

$$\underbrace{P(X_i)P(X_j \mid X_i)P(X_k \mid X_j)}_{\text{serial connection}} = P(X_j, X_i)P(X_k \mid X_j) =$$

$$= \underbrace{P(X_i \mid X_j)P(X_j)P(X_k \mid X_j)}_{\text{divergent connection}}.$$

Then $X_i \rightarrow X_j \rightarrow X_k$ and $X_i \leftarrow X_j \rightarrow X_k$ are equivalent. This is true, however, only if the global distribution is positive everywhere because it may not be possible to reverse the direction of the conditioning:

$$P(X_i \mid X_j) \neq \frac{P(X_i, X_j)}{P(X_j)} \qquad \text{if } P(X_j) = 0.$$

Summary

- Bayesian networks are a combination of a DAG and a global distribution, both defined on the same variables.
- Bayesian networks provide a systematic decomposition of the global distribution into lower-dimensional local distributions, in a divide-and-conquer way.
- Bayesian network provide a principled solution to the problem of feature selection using Markov blankets.
- Three distributional assumptions are common: discrete, Gaussian, and conditional linear Gaussian.

Advanced Inference

Bayesian Networks are not Necessarily Causal

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

The existence of equivalence classes of networks that are indistinguishable from a probabilistic point of view provides a simple proof that edge 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 edges in causal terms, please do not do it lightly, especially with observational data.

Probabilistic and Causal Bayesian Networks

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 which is faithful to the probability distribution **P** of X, so that the only dependencies in **P** are those arising from d-separation in the DAG.
- There must be no latent variables (unobserved variables in uencing 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 edge 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, as 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 edges 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

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).

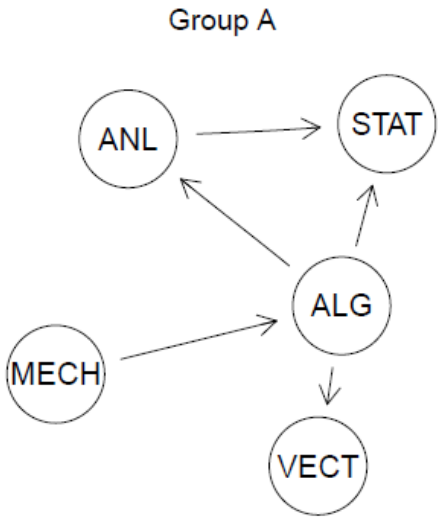
```
library(bnlearn)
head(marks)
## MECH VECT ALG ANL STAT
## 1 77 82 67 67 81
## 2 63 78 80 70 81
## 3 75 73 71 66 81
## 4 55 72 63 70 68
## 5 63 63 65 70 63
## 6 53 61 72 64 73
```

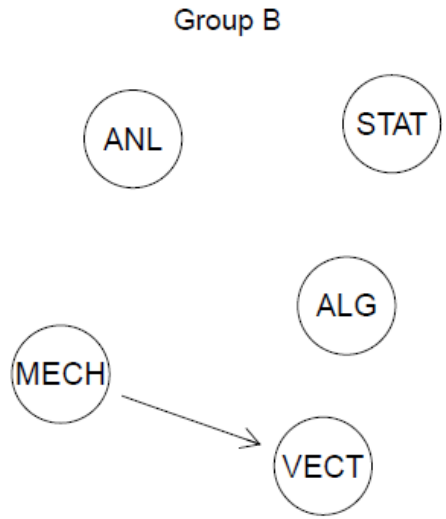
Add Latent Grouping...

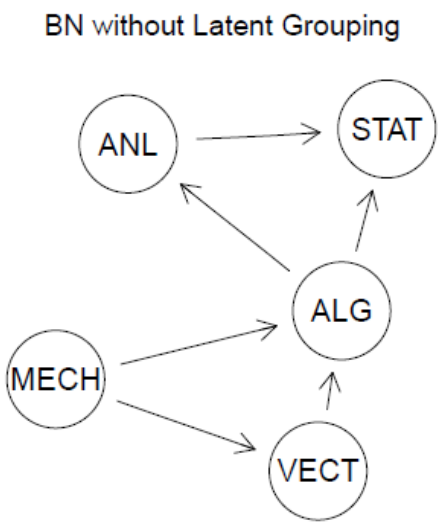
Edwards noted that the students apparently belonged to two groups (which we will call A and B) with substantially different academic profiles. He then assigned each student to one of those two groups using the EM algorithm to impute group membership as a latent variable (say, LAT). The EM algorithm assigned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

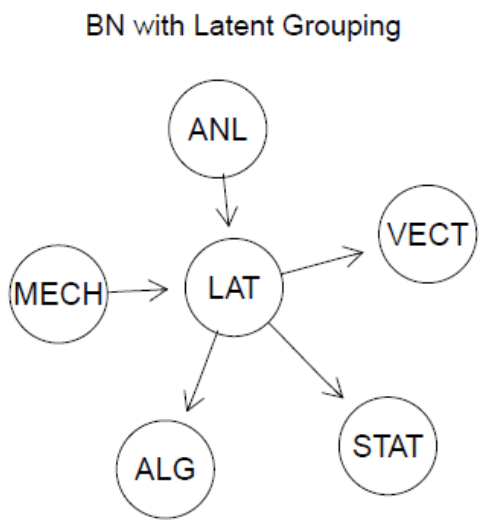
```
latent = factor(c(rep("A", 44), "B", rep("A", 7), rep("B", 36)))
modelstring(hc(marks[latent == "A", ]))
## [1] "[MECH][ALG|MECH][VECT|ALG][ANL|ALG][STAT|ALG:ANL]"
modelstring(hc(marks[latent == "B", ]))
## [1] "[MECH][ALG][ANL][STAT][VECT|MECH]"
modelstring(hc(marks))
## [1] "[MECH][VECT|MECH][ALG|MECH:VECT][ANL|ALG][STAT|ALG:ANL]"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

... And the Models Look Nothing Alike









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 discretize 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]"
```

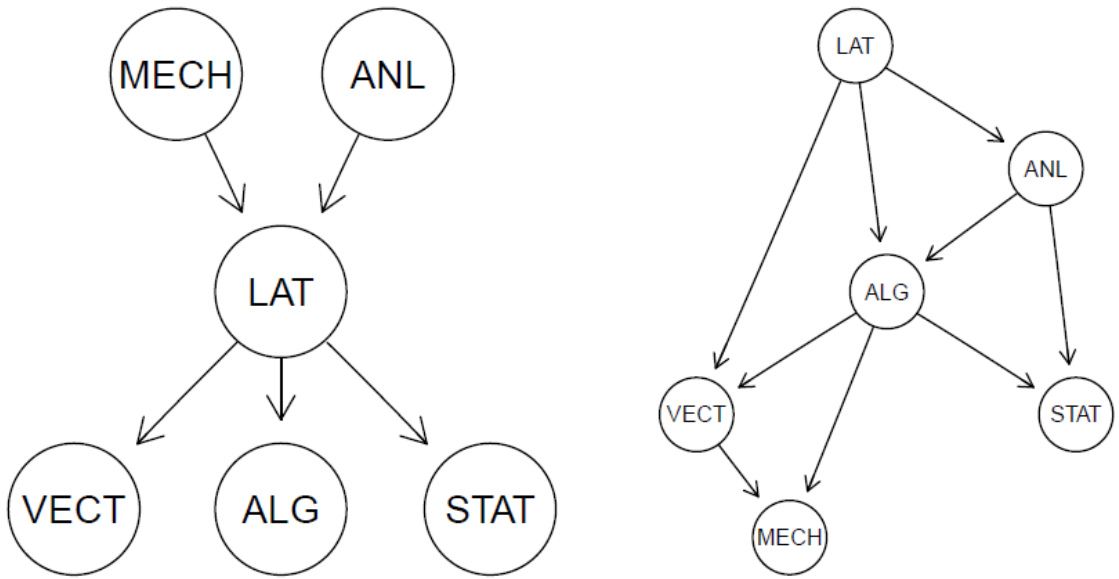
This BN seems to provide a simple interpretation of the relationships between the topics: the grades in mechanics and analysis can be used to infer which group a student belongs to, and that in turn influences the grades in the remaining topics.

However, if we choose not to discretise:

```
modelstring(hc(data.frame(marks, LAT = latent)))
## [1] "[LAT][ANL|LAT][ALG|ANL:LAT][VECT|ALG:LAT][STAT|ALG:ANL][MECH|VECT:ALG]"
```

With Discretization, Without Discretization

```
par(mfrow = c(1, 2))
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. And 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 $B \leftarrow A \rightarrow C$:

```
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 $B \not\perp\!\!\!\perp_G C$ 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 $B \perp\!\!\!\perp_P C \mid A$.

```
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 A $\rightarrow$ B $\rightarrow$ 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 A $\not\perp\!\!\!\perp_G$ 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 edge from the other parent correctly;
- if the common child is the latent variable, the parents are not linked by a (spurious) edge.

In Conclusion

- The robustness of causal networks rests on the assumptions 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 edge.

Causal Inference

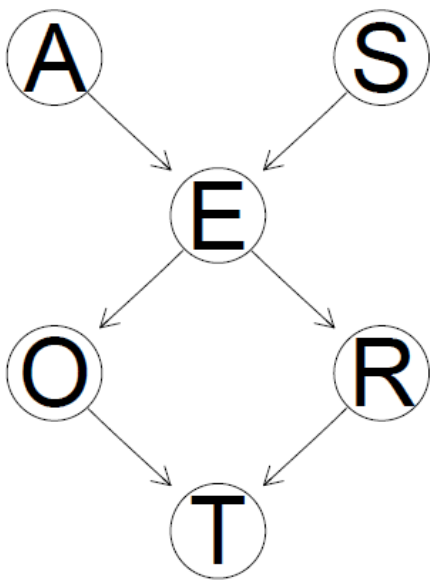
Once we have a causal BN we are happy with, we can again focus on using it to answer relevant questions. In the context of causal networks, we call this causal inference. Compared to the posterior inference we have seen in the previous lecture:

- 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

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:

$$P(O \mid R = big \mid G, \Theta).$$

The Train Use Survey (Posterior)

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

```
prop.table(table(cpdist(survey.bn, "O", evidence = (R == "big"))))
##
## emp self
## 0.954 0.046
```

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)))
##
## emp self
## 0.9476 0.0524
```

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 and work in a big city (say, by starting a public housing program) how will that affect the occupation status? Note that if we do this we alter the characteristics of the population so the BN will be a valid tool to investigate this. The effects of the intervention (the public housing program) will change

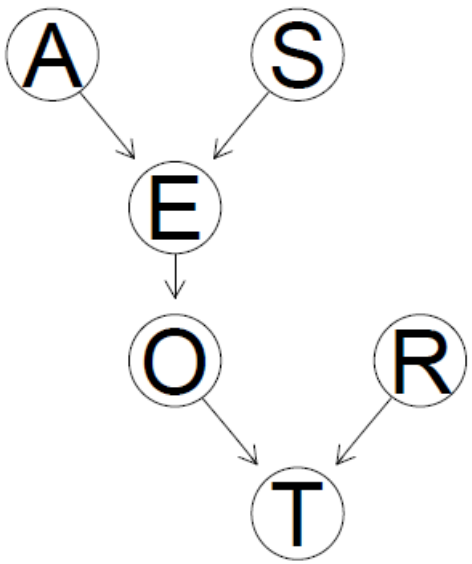
```
coef(survey.bn$R)
## E
## R high uni
## small 0.25 0.20
## big 0.75 0.80
```

to

```
mut.bn = mutilated(survey.bn, evidence = list(R = "big"))
coef(mut.bn$R)
## small big
## 0 1
```

because we give everybody a house in a big city, regardless of their education E.

The Train Use Survey Revisited (Causal, II)



We can then compute the effect of this policy on the occupation by calling **cpquery** again but on the mutilated network that incorporates the intervention.

```
prop.table(table(cpdist(mut.bn, "O",
evidence = TRUE)))
##
## emp self
## 0.9492 0.0508
```

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

This approach is called 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)))
##
## 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
```

Causal Inference and Experimental Design

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 is expensive and/or impossible in many cases.
- 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.
- Randomization is equivalent to a soft causal intervention.
- Since randomized variables have no parents, causality necessarily flows from them to the target variables

Fundamentals of Structure Learning

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 work ow is implicitly Bayesian; given a data set $\mathcal{D}$ and if we denote the parameters of the global distribution as X with Θ, we have

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

and structure learning is done in practise as

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

Local Distributions: Divide and Conquer

Most tasks related to both learning and inference are NP-hard (they cannot be solved in polynomial time in the number of variables). They are still feasible thanks to the decomposition of X into local distributions; under some assumptions we can use local computations and we never need to manipulate more than one at a time. In Bayesian networks, for example, structure learning boils down to

$$P(\mathcal{D} \mid \mathcal{G}) = \int \prod_{j=1}^{p} \left[P(X_j \mid X_{\mathrm{Pa}(j)}, \Theta_{X_j}) P(\Theta_{X_j} \mid \Pi_{X_j}) \right] d\Theta$$

$$= \prod_{j=1}^{p} \left[\int P(X_j \mid X_{\mathrm{Pa}(j)}, \Theta_{X_j}) P(\Theta_{X_j} \mid X_{\mathrm{Pa}(j)}) d\Theta_{X_j} \right]$$

and parameter learning boils down to

$$P(\Theta \mid \mathcal{G}, \mathcal{D}) = \prod_{j=1}^{p} P(\Theta_{X_j} \mid X_{\mathrm{Pa}(j)}, \mathcal{D}).$$

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 X;
- using available data and extract the information the contain.

In structure learning, elicitation involves favoring or penalizing the inclusion of specific (patterns of) edges in the DAG; in parameter learning, it means partially or completely specify the parameters of local distribution, or to constrain them in various ways.

There are pros and cons to either approach:

- it maybe diffcult to find experts, or it may be diffcult to find data, depending on the phenomenon;
- the data may be noisy or not fit distributional assumptions;
- it is usually diffcult 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

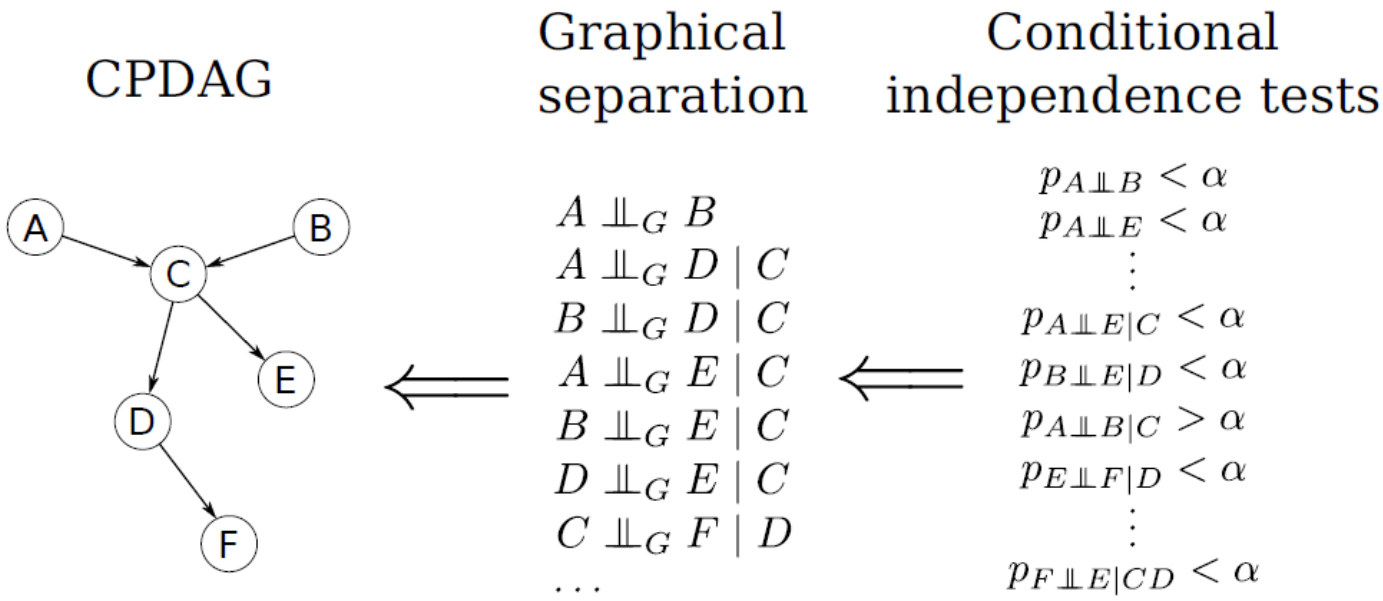
- There must be a one-to-one correspondence between the nodes in the DAG and the random variables in X; there must not be multiple nodes which are deterministic functions of a single variable.
- All the relationships between the variables in X must be conditional independencies, because they are by definition the only kind of relationships that can be expressed by a BN.
- Every combination of the possible values of the variables in X must represent a valid, observable (even if really unlikely) event. This assumption implies a strictly positive global distribution, which is needed to have uniquely determined Markov blankets and, therefore, 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 Bayesian networks.

Classes of Structure Learning Algorithms from Data

Despite the (sometimes confusing) variety of theoretical backgrounds and terminology they 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: each candidate DAG is assigned a score reflecting its goodness of fit, which is then taken as an objective function to maximize.
- Hybrid algorithms: conditional independence tests are used to learn at least part of the conditional independence relationships from the data, thus restricting the seedgeh space for a subsequent score-based seedgeh. The latter determines which edges are actually present in the graph and their direction.

Constraint-Based Structure Learning Algorithms



The mapping between edges and conditional independence relationships lies at the core of BNs; therefore, one way to learn the structure of a BN is to check which such relationships hold using a suitable conditional independence test. Such an approach results in a set of conditional independence constraints that identify a single equivalence class.

Assuming a Perfect Map

BNs are defined as I-maps so

$$A \perp\!\!\!\perp_G B \mid C \quad \Rightarrow \quad A \perp\!\!\!\perp_P B \mid C.$$

However, constraint-based algorithms treat them as perfect maps since they do

$$A \perp\!\!\!\perp_P B \mid C \quad \Leftrightarrow \quad A \perp\!\!\!\perp_G B \mid C.$$

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

- the assumption that the DAG is a perfect map for X is impossible to verify;
- but it is a suffcient assumption to uniquely identify Markov blankets, and thus we no longer need to assume P(X) is strictly positive everywhere;
- not all P(X) have a faithful DAG.

The Inductive Causation Algorithm

1. For each pair of variables A and B in X seedgeh for set $S_{AB} \in X$ such that A and B are independent given S_{AB} and $A, B \notin S_{AB}$. If there is no such a set, place an undirected edge between A and B.
2. For each pair of non-adjacent variables A and B with a common neighbour C, check whether $C \in S_{AB}$. If this is not true, set the direction of the edges A - C and C - B to A $\rightarrow$ C and C $\leftarrow$ B.
3. Set the direction of edges 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.

Other Constraint-based algorithms

- Peter & Clark (PC): a true-to-form implementation of the Inductive Causation algorithm, specifying only the order of the conditional independence tests. Starts from a saturated network and performs tests gradually increasing the number of conditioning nodes.
- Grow-Shrink (GS) and Incremental Association (IAMB) variants: these algorithms learn the Markov blanket of each node to reduce the number of tests required by the Inductive Causation algorithm. Markov blankets are learned using different forward and step-wise approaches; the initial network is assumed to be empty (i.e. not to have any edge).
- Max-Min Parents & Children (MMPC): uses a minimax approach to avoid conditional independence tests known a priori to accept the null hypothesis of independence.
- Hiton-PC (HITON-PC): currently the most scalable choice, it uses a first pass based on marginal tests followed by a backward selection.

Conditional Independence Tests: Discrete Variables

Conditional independence tests used to learn DBN are functions of the observed frequencies $\{n_{ijk}, i = 1, \ldots, R, j = 1, \ldots, C, k = 1, \ldots, L\}$ for the random variables X and Y and all the con
gurations of the conditioning variables Z. Classic choices are:

- mutual information/log-likelihood ratio

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

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

$$X^2(X, Y \mid Z) = \sum_{i=1}^{R} \sum_{j=1}^{C} \sum_{k=1}^{L} \frac{(n_{ijk} - m_{ijk})^2}{m_{ijk}}, \text{ where } m_{ijk} = \frac{n_{i+k} n_{+jk}}{n_{++k}}.$$

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

Conditional Independence Tests: Gaussian Variables

Conditional independence tests used to learn GBNs are functions of the partial correlations $\rho_{XY|Z}$ that are used as proxies for the cells of $\Omega = \Sigma^{-1}$. Classic choices are:

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

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

  and distributed as a Student's t with $n - |Z| - 2$ degrees of freedom;

- Fisher's Z test, a transformation of $\rho_{XY|Z}$ with an asymptotic normal distribution and defined as

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

  where n is the number of observations and |Z| is the number of nodes belonging to Z.

Conditional Independence Tests: Conditional Gaussian (I)

It is more complicated to specify tests for CLGBNs, because not all triplets (X, Y, Z) can be directly represented as a single local distribution. Going case by case:

- if X, Y and Z are all categorical, we can use any test for DBNs;
- if X, Y and 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

$$\propto \log \frac{P(Y \mid X, Z)}{P(Y \mid Z)}$$

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

Conditional Independence Tests: Conditional Gaussian (II)

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

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

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

Conditional Independence Tests: Permutations

Asymptotic tests require a sample size large enough for the null distribution to converge to its asymptotic behaviour. We can use permutation tests instead:

1. Compute the test statistic $\hat{t}$ on the original (X, Y, Z).
2. For $b = 1, \ldots, B$:
   - 2.1 permute Y while keeping X and Z fixed, to obtain a new sample (X, Y_b^*, Z) from the null distribution in which $X \perp\!\!\!\perp_P Y_b^* \mid Z$.
   - 2.2 Compute the test statistic $\hat{t}_b$ on (X, Y_b^*, Z).
3. The p-value of the test as

$$\frac{1}{B}\sum_{b=1}^{B} \mathbf{1}\{\hat{t} > t_b\}$$

for one-tailed tests and

$$\frac{1}{B}\sum_{b=1}^{B} \mathbf{1}\{|\hat{t}| > |t_b|\}$$

for two-tailed tests.

Conditional Independence Tests: Shrinkage

An alternative is to regularise the test statistic by shrinking it towards a regular target distribution. For instance, in the case of a covariance matrix we estimate $\bar{\Sigma}$ as a linear combination of the maximum likelihood estimator $\hat{\Sigma}$ and a target distribution with a diagonal covariance matrix T:

$$\bar{\Sigma} = \lambda T + (1 - \lambda)\hat{\Sigma}, \qquad \lambda \in [0, 1].$$

λ can be estimated in closed form as

$$\lambda^* = \frac{\sum_{i=1}^{k} \sum_{j=1}^{k} VAR(\hat{\sigma}_{ij}) - COV(\hat{\sigma}_{ij}, t_{ij})}{\sum_{i=1}^{k} \sum_{j=1}^{k} (t_{ij} - \hat{\sigma}_{ij})^2}.$$

The modified $\bar{\Sigma}$ can then be used to compute the (partial) correlations used in the conditional independence tests.

A similar approach can be used for categorical data and mutual information.

The ASIA Example, Revisited

An alternative is to regularise the test statistic by 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.

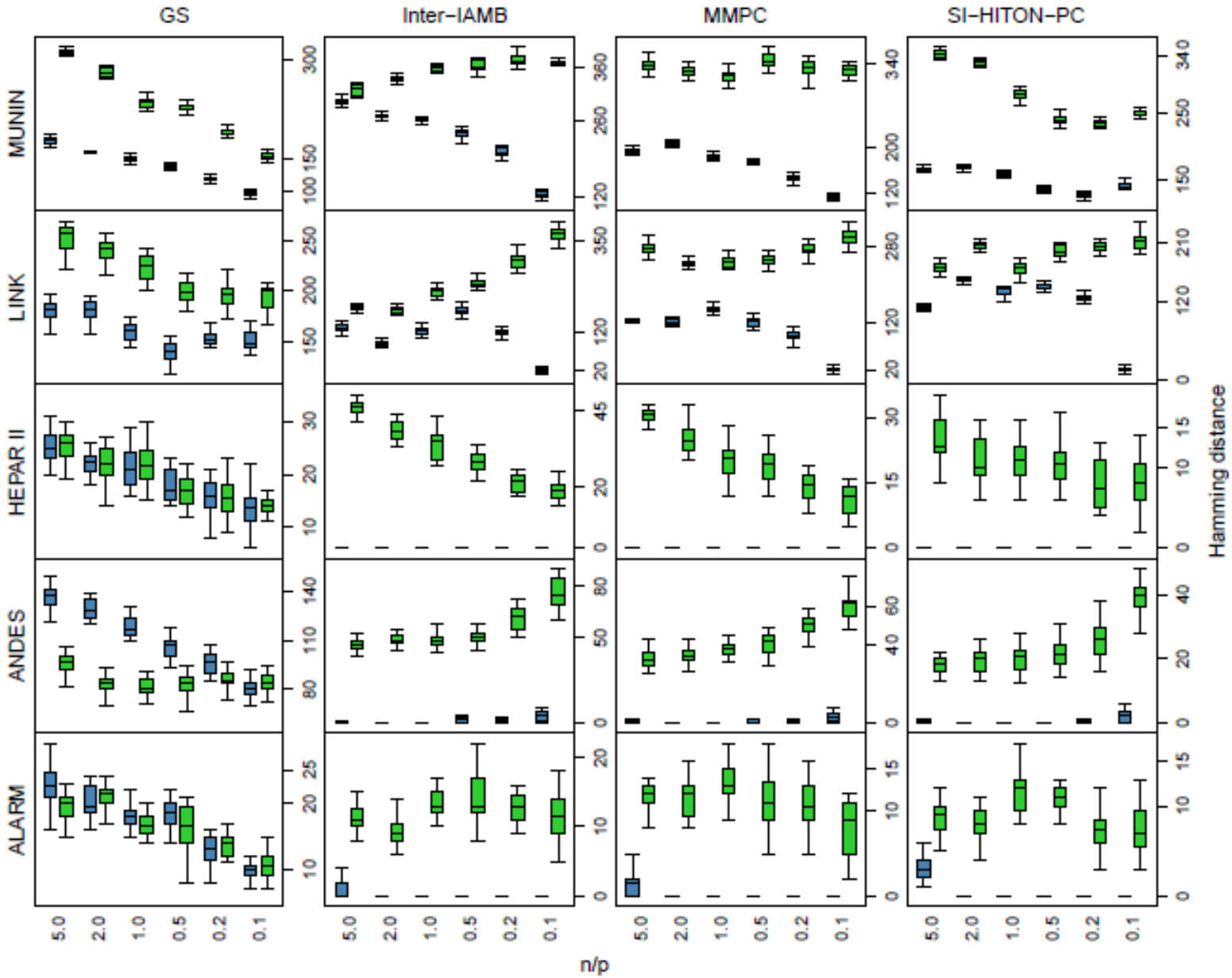
```
cpdag = si.hiton.pc(asia, undirected = FALSE)
cpdag
##
## Bayesian network learned via Constraint-based methods
##
## model:
## [partially directed graph]
## nodes: 8
## edges: 5
## undirected edges: 1
## directed edges: 4
## average markov blanket size: 1.75
## average neighbourhood size: 1.25
## average branching factor: 0.50
##
## learning algorithm: Semi-Interleaved HITON-PC
## conditional independence test: Mutual Information (disc.)
## alpha threshold: 0.05
## tests used in the learning procedure: 55
## optimized: TRUE
```

bnlearn: Parameters and Tuning Arguments

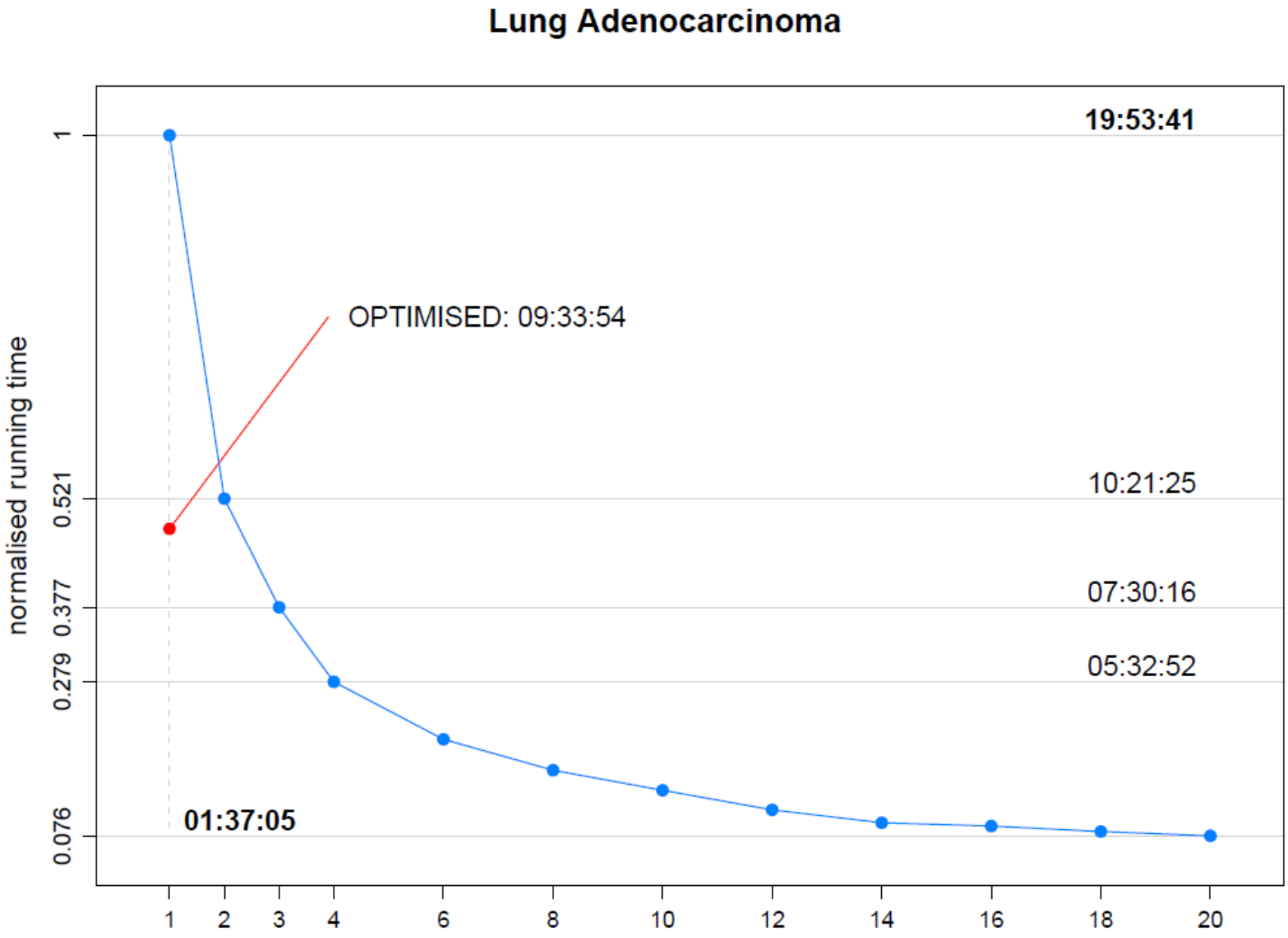
The arguments for the tuning parameters of constraint-based learning algorithms have the same names in the respective functions:

- 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).
- B: number of permutations to use in permutation tests.
- optimized: use (or not) backtracking to roughly halve the number of tests by using the symmetry of Markov blankets and neighbours.
- skeleton: whether to learn just the skeleton instead of the CPDAG.
- debug: whether to print out the steps performed by the algorithm.

Using Backtracking Is Not Such A Good Idea...

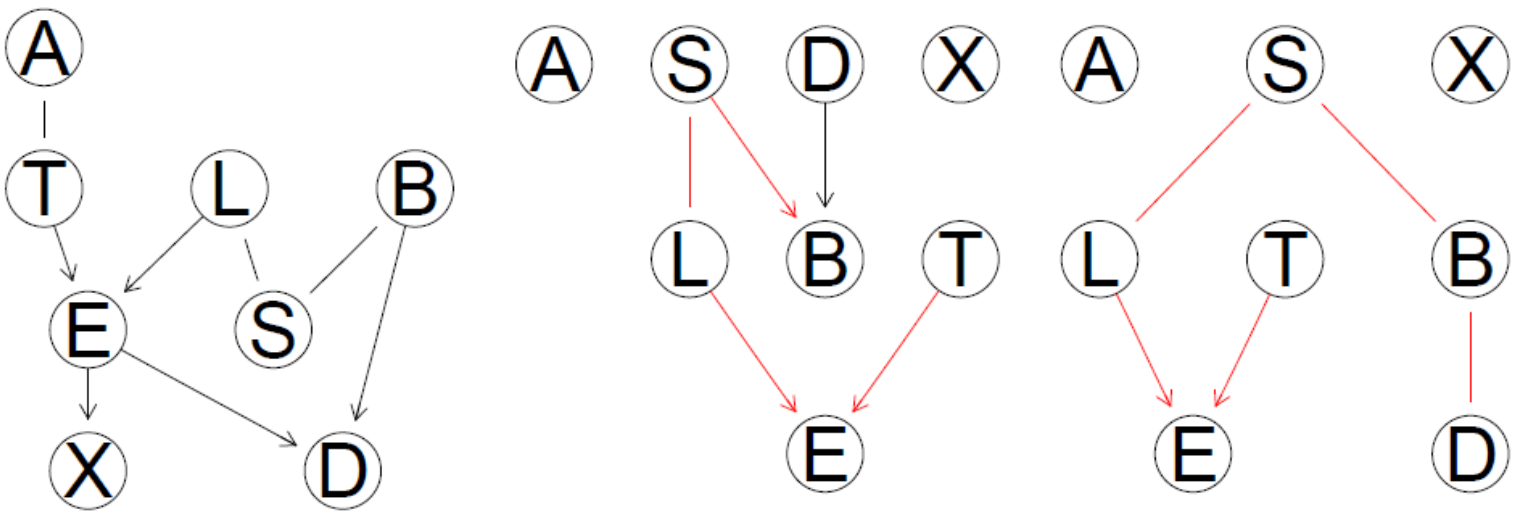


... Because Parallel Computing is Safer and Faster



bnlearn: With and Without Backtracking

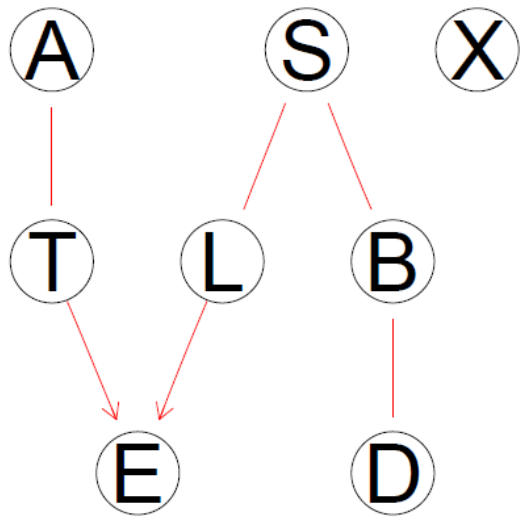
```
par(mfrow = c(1, 3))
true.dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
graphviz.plot(cpdag(true.dag))
graphviz.plot(cpdag, highlight = list(edges = edges(cpdag(true.dag))), )
cpdag2 = si.hiton.pc(asia, undirected = FALSE, optimized = FALSE)
graphviz.plot(cpdag2, highlight = list(edges = edges(cpdag(true.dag))))
```



The reason why si.hiton.pc() cannot learn the CPDAG is that there are many nodes with 0s and 1s in the CPTs, which breaks the convergence of the mutual information to the χ^2 distribution.

bnlearn: Permutation Tests Do A Little Better

```
cpdag2 = si.hiton.pc(asia, test = "mc-mi", undirected = FALSE, optimized = FALSE)
graphviz.plot(cpdag2, highlight = list(edges = edges(cpdag(true.dag))))
```



There is only one edge missing; all the reference DBNs are impossible to learn perfectly at any reasonable sample size, so this is a pretty good result.

bnlearn: The Debugging Output (I)

```
debugging.output = capture.output(
si.hiton.pc(asia, test = "mc-mi", undirected = FALSE, optimized = 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 0.0046 ."
## [7] " * nodes that will be disregarded from now on."
## [8] " > S has p-value 0.131 ."
## [9] " > L has p-value 0.368 ."
## [10] " > B has p-value 0.0616 ."
## [11] " > E has p-value 0.0758 ."
## [12] " > X has p-value 0.182 ."
## [13] " > D has p-value 0.0858 ."
## [14] " @ T accepted as a parent/children candidate ( p-value: 0.0046 )."
## [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("^\\*", 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] "* backward phase for candidate node A ."
## [11] "* forward phase for node L ."
## [12] "* backward phase for candidate node B ."
## [13] "* backward phase for candidate node E ."
## [14] "* backward phase for candidate node X ."
## [15] "* backward phase for candidate node D ."
```

bnlearn: The Debugging Output (III)

```
head(grep("^\\*|\\s*@", debugging.output, value = TRUE), n = 20)
## [1] "* forward phase for node A ."
## [2] " @ T accepted as a parent/children candidate ( p-value: 0.0046 )."
## [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] "* backward phase for candidate node A ."
## [15] " @ A accepted as a parent/children candidate ( p-value: 0.0056 )."
## [16] "* forward phase for node L ."
## [17] " @ S accepted as a parent/children candidate ( p-value: 0 )."
## [18] "* backward phase for candidate node B ."
## [19] "* backward phase for candidate node E ."
## [20] " @ E accepted as a parent/children candidate ( p-value: 0 )."
```

bnlearn: Learning Markov Blankets and Neighbourhoods

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.

```
ci.test(x = "S", y = "E", z = "L", data = asia, test = "mc-mi")
##
## Mutual Information (disc., MC)
##
## data: S ~ E | L
## mc-mi = 4e-06, Monte Carlo samples = 5000, p-value = 0.9
## 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 $x \perp\!\!\!\perp_P y \mid z$ where z can be either absent (for marginal tests) or a vector of labels (to condition on one or more variables).

Pros & 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 underperform.
  - Permutation tests on the other hand 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 is slower than score-based and hybrid algorithms.
- At any single time they evaluate a small subset of variables, which makes them very memory effcient.
- 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.

Score-based Structure Learning Algorithms

The dimensionality of the space of graph structures makes an exhaustive seedgeh 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, i.e.

$$\text{Score}(\mathcal{G}) = \sum_{j=1}^{p} \text{Score}(X_j \mid X_{\text{Pa}(j)})$$

such as

$$\text{BIC}(\mathcal{G}) = \sum_{j=1}^{p} \log P(X_j \mid X_{\text{Pa}(j)}) - \frac{|\Theta_{X_j}|}{2} \log n$$

$$\text{BDe}(\mathcal{G}), \text{BGe}(\mathcal{G}) = \sum_{j=1}^{p} \log \left[\int P(X_j \mid X_{\text{Pa}(j)}, \Theta_{X_j}) P(\Theta_{X_j} \mid X_{\text{Pa}(j)}) d\Theta_{X_j} \right]$$

if each comparison involves structures differing 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 $\text{Score}_{\mathcal{G}} = \text{Score}(\mathcal{G})$.
3. Set *maxscore* $= \mathit{Score}_{\mathcal{G}}$.
4. Repeat the following steps as long as *maxscore* increases:
   - 4.1 for every possible edge addition, deletion or reversal not resulting in a cyclic network:
     - 4.1.1 compute the score of the modified network $\mathcal{G}^*$, $\text{Score}_{\mathcal{G}^*} = \text{Score}(\mathcal{G}^*)$:
     - 4.1.2 if $\text{Score}_{\mathcal{G}^*} > \text{Score}_{\mathcal{G}}$, set $\mathcal{G} = \mathcal{G}^*$ and $\text{Score}_{\mathcal{G}} = \text{Score}_{\mathcal{G}^*}$.
   - 4.2 update *maxscore* with the new value of $\text{Score}_{\mathcal{G}}$.
5. Return the directed acyclic graph $\mathcal{G}$.

DBNs: The Bayesian Dirichlet Marginal Likelihood

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

- Dirichlet conjugate prior
  $X_j \mid X_{\mathrm{Pa}(j)} \sim \mathit{Multinomial}(\Theta_{X_j})$ and
  $\Theta_{X_j} \mid X_{\mathrm{Pa}(j)} \sim \mathit{Dirichlet}(\alpha_{ijk}), \sum_{jk} \alpha_{ijk} = \alpha_i$ the imaginary sample size;
- positivity (all conditional probabilities $\pi_{ijk} > 0$);
- parameter independence (π_{ijk} for different parent con gurations are independent) and modularity (π_{ijk} in different nodes are independent);

Heckerman et al. 1995 derived a closed form expression for $P(\mathcal{D} \mid \mathcal{G})$:

$$\mathrm{BD}(\mathcal{G}, \mathcal{D}; \alpha) = \prod_{i=1}^{p} \mathrm{BD}(X_i, X_{\mathrm{Pa}(i)}; \alpha_i) = \prod_{i=1}^{N} \prod_{j=1}^{q_i} \left[\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + n_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk} + n_{ijk})}{\Gamma(\alpha_{ijk})} \right]$$

where r_i is the number of states of X_i; q_i is the number of configurations of $X_{\mathrm{Pa}(i)}$; $n_{ij} = \sum_k n_{ijk}$; and $\alpha_{ij} = \sum_k \alpha_{ijk}$.

DBNs: Bayesian Dirichlet Equivalent Uniform (BDeu)

The most common implementation of BD assumes $\alpha_{ijk} = \alpha/(r_i q_i)$, $\alpha_i = \alpha$ and is known as the Bayesian Dirichlet equivalent uniform (BDeu) marginal likelihood. 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 edges 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 $\mathcal{D}$.
- There are formal proofs of all this.

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 $X_{\mathsf{Pa}(i)}$ that are not observed in $\mathcal{D}$.

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

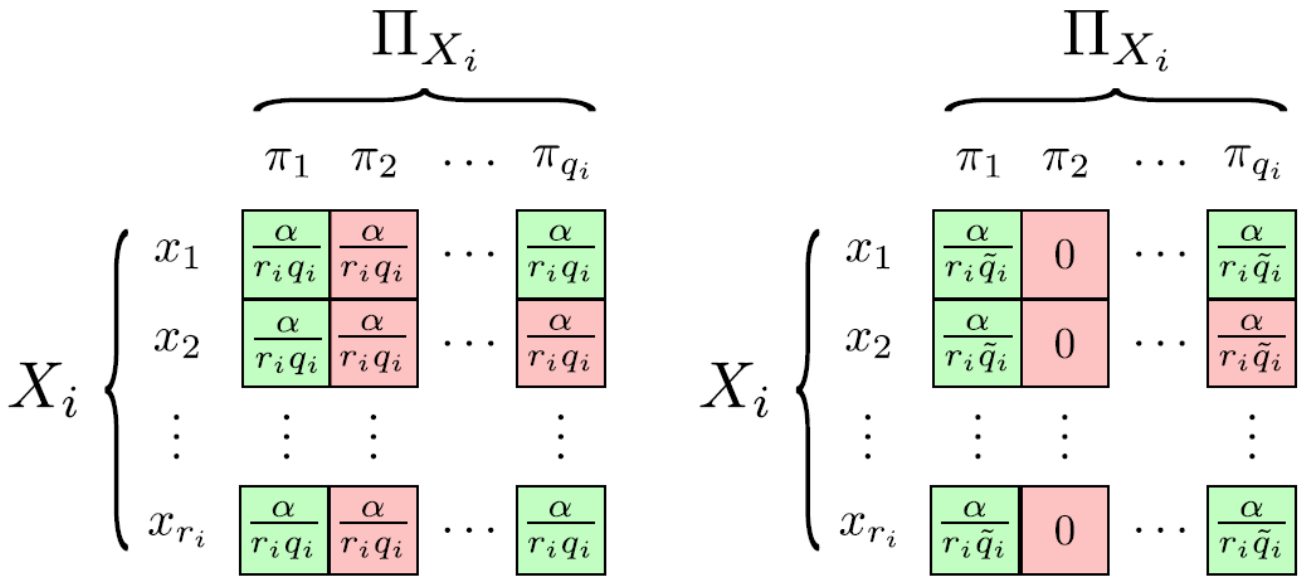
So the effective imaginary sample size decreases as the number of unobserved parents configurations increases, and the MAP estimates of π_{ijk} gradually converge to the ML and favour overfitting.

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

$$\widetilde{\alpha} = \begin{cases} \alpha/(r_i\widetilde{q}_i) & \text{if } n_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}, \widetilde{q}_i = \{\text{number of } X_{\mathsf{Pa}(i)} \text{ such that } n_{ij} > 0\}$$

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 $(X_i, X_{\mathsf{Pa}(i)})$ 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 $P(\mathcal{D} \mid \mathcal{G})$ associated with a normal-Wishart prior (μ, W) with $\mu \sim N(\nu, \alpha_\mu W)$ and $W \sim \mathit{Wishart}(T, \alpha_w)$:

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

where

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

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

($|.|$ is defined to be $|X_i \cup X_{\mathrm{Pa}(i)}| = |X_{\mathrm{Pa}(i)}| + 1$.)

Penalized Likelihoods: AIC and BIC

Penalised likelihoods also make very popular scores for DBNs, GBNs and CLGBNs. AIC tends to overfit a lot, while BIC tends to underfit a bit but it often used an approximation to $P(\mathcal{D} \mid \mathcal{G})$. For DBNs, the log-likelihood and the number of parameters associated with a local distribution are:

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

for GBNs:

$$LL(X_i, X_{\mathrm{Pa}(i)}) = \prod_{m=1}^{n} N(x_m; \mu_{X_i} + \pi_m \beta_{X_i}, \sigma^2_{X_i}), \ |\Theta_{X_i}| = |X_{\mathrm{Pa}(i)}| + 1;$$

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

$$LL(X_i, X_{\mathrm{Pa}(i)}) = \prod_{m=1}^{n} N(x_m; \mu_{X_i,\delta_m} + \gamma_m \beta_{X_i,\delta_m}, \sigma^2_{X_i,\delta_m}),$$

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

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.

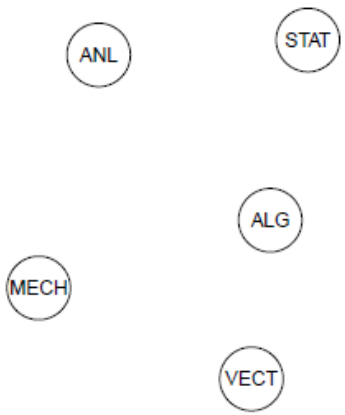
```
dag.marks = hc(marks, score = "bic-g")
```

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.

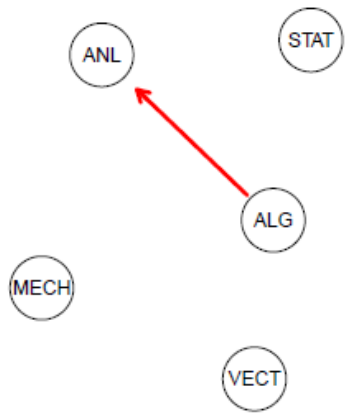
```
true.dag =
model2network("[ALG][ANL|ALG][MECH|ALG:VECT][STAT|ALG:ANL][VECT|ALG]")
unlist(compare(dag.marks, true.dag))
## tp fp fn
## 3 3 3
unlist(compare(cpdag(dag.marks), cpdag(true.dag)))
## tp fp fn
## 6 0 0
```

The Hill-Climbing Algorithm (MARKS)

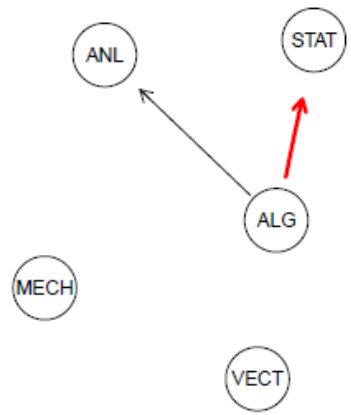
Initial BIC score: −1807.528



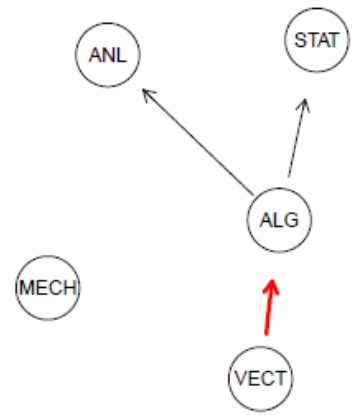
Current BIC score: −1778.804



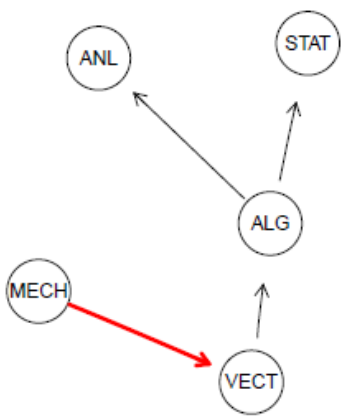
Current BIC score: −1755.383



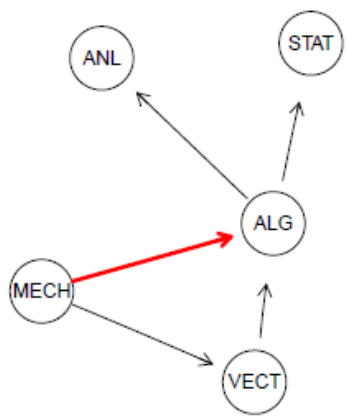
Current BIC score: −1737.176



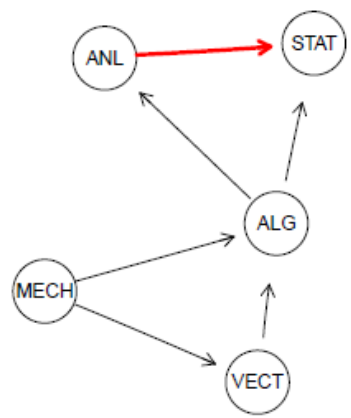
Current BIC score: −1723.325



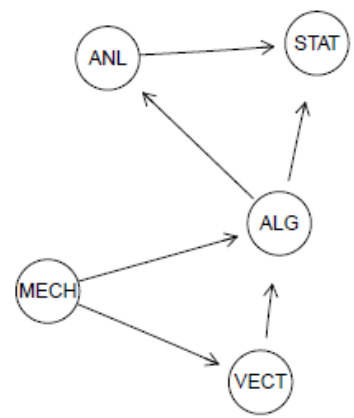
Current BIC score: −1720.901



Current BIC score: −1720.150



Final BIC score: −1720.150



bnlearn: Comparing Networks

- 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 edges 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 edge 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 edges 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("^nn* (best|doing)", 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] "* doing a random restart, 9 of 10 left."
## [9] "* best operation was: adding E -> X ."
## [10] "* best operation was: adding E -> D ."
```

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, changing a number of its edges to created a new DAG.

```
head(grep("^nn* (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: -11237 "
## [11] "* current score: -11106 "
## [12] "* current score: -11101 "
## [13] "* current score: -11096 "
## [14] "* doing a random restart, 8 of 10 left."
```

bnlearn: Hill-Climbing With Preseeded Networks

Another way of avoid getting stuck in local maxima is to start the seedgeh 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|A][E|A][D|S][L|T][B|S:L][X|S:B] "
```

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

Other Score-based Algorithms

- Greedy Equivalent Seedgeh: hill-climbing over equivalence classes rather than graph structures; the seedgeh space is much smaller.
- Tabu Seedgeh: a modified hill-climbing that keeps a list of the last k structures visited (the tabu list), and returns only if they are all worse than the current one.
- Genetic Algorithms: they perturb (mutation) and combine (crossover) features through several generations of structures, and keep the ones leading to better scores. Inspired by Darwinian evolution.
- Simulated Annealing: again similar to hill-climbing, but not looking at the maximum score improvement at each step. Very difficult to use in practice because of its tuning parameters.

bnlearn: TAbU Seedgeh

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("^nn* (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 decrease
## [9] "* best operation was: reversing S -> L ."
## [10] "* network score did not increase (for 2 times), looking for a minimal decrease
```

Pros & Cons of Score-based Algorithms

- Convergence to the global maximum (i.e. the best structure) is not guaranteed for finite samples, the seedgeh 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 nobody can 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 maximizes some score function and that satis es 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 **maximize**: find the network structure $\mathcal{G}^*$ that maximizes $\mathrm{Score}(\mathcal{G}^*)$ 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 maximize phase we obtain the Max-Min Hill-Climbing (MMHC) algorithm as a particular case.

bnlearn: rsmax2()

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

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

Its main arguments are:

- test: the conditional independence test to use in the restrict phase;
- score: score function to use in the maximize phase;
- restrict: constraint-based algorithm to use in the restrict phase;
- restrict.args: its optional arguments;
- maximize: score-based algorithm to use in the maximize phase;
- maximize.args: its optional arguments.

bnlearn: mmhc()

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:     Max-Min Parent Children "
## [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:     Max-Min Parent Children "
## [4] " score-based method:          Hill-Climbing "
```

Pros & 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. We 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 Comprasion

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
##  5  3  1
shd(cpdag2, true.dag)
## [1] 3
```

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 difined 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.

Advanced Structure Learning, Parameter Learning

The DAGs and Distributions

BN literature focuses mostly on (the parameters of) local probability distributions. However:

- Comparing models learned with different algorithms is difficult, because they maximize different scores, use different estimators for the parameters, work under different sets of hypotheses, etc.
- Unless the true global probability distribution is known it is difficult to assess the uncertainty of the estimated models.
- The few available measures of structural difference are completely descriptive in nature (e.g. the Structural Hamming distance), and are difficult to interpret.
- When learning causal graphical models often we are looking for particular patterns of edges in the DAG.

Looking for a Solution

Focusing on the DAGs G sidesteps some of these problems and is useful in structure learning as well, since

$$\mathrm{P}(\mathcal{G} \mid \mathcal{D}) \propto \mathrm{P}(\mathcal{G})\mathrm{P}(\mathcal{D} \mid \mathcal{G}).$$

So:

1. We need to know more about the properties of priors $\mathrm{P}(\mathcal{G})$ and posteriors $\mathrm{P}(\mathcal{G} \mid \mathcal{D})$ over the space of DAGs, preferably as a function of their edge sets, say $\mathrm{P}(\mathcal{G}(\mathcal{E}))$ and $\mathrm{P}(\mathcal{G}(\mathcal{E}) \mid \mathcal{D})$ with $\mathcal{E} = \{(u_i, v_j), i \neq j\}$

And then:

1. It would be good to have measures of spread for $\mathcal{G}$, to assess the noisiness of $\mathrm{P}(\mathcal{G}(\mathcal{E}) \mid \mathcal{D})$ and the informativeness of $\mathrm{P}(\mathcal{G}(\mathcal{E}))$.
2. It would be interesting to study the convergence speed of structure learning algorithms given their tuning parameters using those measures.

A Simpler Case: Undirected Graphs

Each edge e_{ij} in an undirected graph $\mathcal{G} = (\mathbf{V}, \mathcal{E})$ has only two possible states and therefore can be modelled as a bernoulli random variable:

$$e_{ij} \sim E_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in E \\ 0 & \text{otherwise} \end{cases}.$$

The natural extension of this approach is to model any set of edges as a multivariate bernoulli random variable $\mathbf{B} \sim Ber_k(\mathbf{p})$. B is uniquely identified by the parameter set

$$\mathbf{p} = \{p_I : I \subseteq \{1, \ldots, k\}, i \neq \emptyset\}, \qquad k = \frac{|\mathbf{V}|(|\mathbf{V}| - 1)}{2}$$

which represents the dependence structure among the marginal distributions $B_i \sim Ber(p_i), i = 1, \ldots, k$ of the edges. $\mathbf{p}$ can be estimated using a large number of bootstrap samples or MCMC samples from $\mathrm{P}(\mathcal{G}(\mathcal{E}) \mid \mathcal{D})$.

DAGs as Multivariate Trinomials

Each are a_{ij} in $\mathcal{G} = (\mathbf{V}, A)$ has three possible states, and therefore it can be modelled as a Trinomial random variable A_{ij}:

$$a_{ij} \sim A_{ij} \begin{cases} -1 & \text{if } a_{ij} = \overleftarrow{a_{ij}} = \{v_i \leftarrow v_j\} \\ 0 & \text{if } a_{ij} \notin A, \text{ denoted with } \dot{a_{ij}} \\ 1 & \text{if } a_{ij} = \overrightarrow{a_{ij}} = \{v_i \rightarrow v_j\} \end{cases}.$$

As before, the natural extension to model any set of edges is to use a multivariate Trinomial random variable $\mathbf{T} \sim Tri_k(\mathbf{p})$. However:

- the acyclicity constraint of bayesian networks makes deriving exact results very difficult because it cannot be written in closed form;
- the score equivalence of most structure learning strategies makes inference on $Tri_k(\mathbf{p})$ tricky.

Second Order Properties of $ber_k(\mathbf{p})$ and $Tri_k(\mathbf{p})$

All the elements of the covariance matrix Σ fo an edge set $\mathcal{E}$ are bounded,

$$p_i \in [0,1] \Rightarrow \sigma_{ii} = p_i - p_i^2 \in \left[0, \frac{1}{4}\right] \Rightarrow \sigma_{ij} \in \left[0, \frac{1}{4}\right]$$

and similar bounds exist for the eigenvalues $\lambda_1, \ldots, \lambda_k$,

$$0 \leq \lambda_i \leq \frac{k}{4} \qquad \text{and} \qquad 0 \leq \sum_{i=1}^{k} \leq \frac{k}{4}.$$

These bounds difine a closed convex set in $\mathbb{R}^k$,

$$\mathcal{L} = \left\{ \Delta^{k-1}(c) : c \in \left[0, \frac{k}{4}\right] \right\}$$

where $\Delta^{k-1}(c)$ is the non-standard $k-1$ simplex

$$\Delta^{k-1}(c) = \left\{ (\lambda_1, \ldots, \lambda_k) \in \mathbb{R}^k : \sum_{k=1}^{k} \lambda_i = c_i, \lambda_i \geq 0 \right\}.$$

Similar results hold for edge sets, with $\sigma_{ii} \in [0,1]$ and $\lambda_i \in [0,k]$.

Minimum and Maximum Entropy

These results provide the foundation for characterising three cases corresponding to different configurations of the probability mass in $\mathrm{P}(\mathcal{G}(\mathcal{E}))$ and $\mathrm{P}(\mathcal{G}(\mathcal{E}) \mid \mathcal{D})$:

- minimum entropy: the probability mass is concentrated on a single DAG. This is the best possible configuration for $\mathrm{P}(\mathcal{G}(\mathcal{E}) \mid \mathcal{D})$, because only one edge set A has a non-zero posterior probability.
- intermediate entropy: several DAGs have non-zero probability. This is the case for informative priors $\mathrm{P}(\mathcal{G}(\mathcal{E}))$ and for the posteriors $\mathrm{P}(\mathcal{G}(\mathcal{E}) \mid \mathcal{D})$ resulting from real-world data sets.
- maximum entropy: all DAGs have the same probability. This is the worst possible configuration for $\mathrm{P}(\mathcal{G}(\mathcal{E}) \mid \mathcal{D})$: it corresponds to a non-informative prior. In other words, the data $\mathcal{D}$ do not provide any information useful in identifying a high-posterior $\mathcal{G}$.

Properties of the Multivariate bernoulli

In the minimum entropy case, only one configuration of edges E has non-zero probability, which means that

$$p_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in E \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad \Sigma = \mathbf{O}$$

where $\mathbf{O}$ is the zero matrix.

The uniform distribution over $\mathbf{G}$ arising from the maximum entropy case has been studied extensively in random graph theory; its two most relevant properties are that all edges e_{ij} are independent and have $p_{ij} = \frac{1}{2}$. As a result, $\Sigma = \frac{1}{4} I_k$; all edges display their maximum possible variability, which along with the fact that they are independent makes this distribution non-informative for $\mathcal{E}$ as well as $\mathcal{G}(\mathcal{E})$.

Properties of the Multivariate Trinomial

The minimum entropy is the same; in the maximum entropy case:

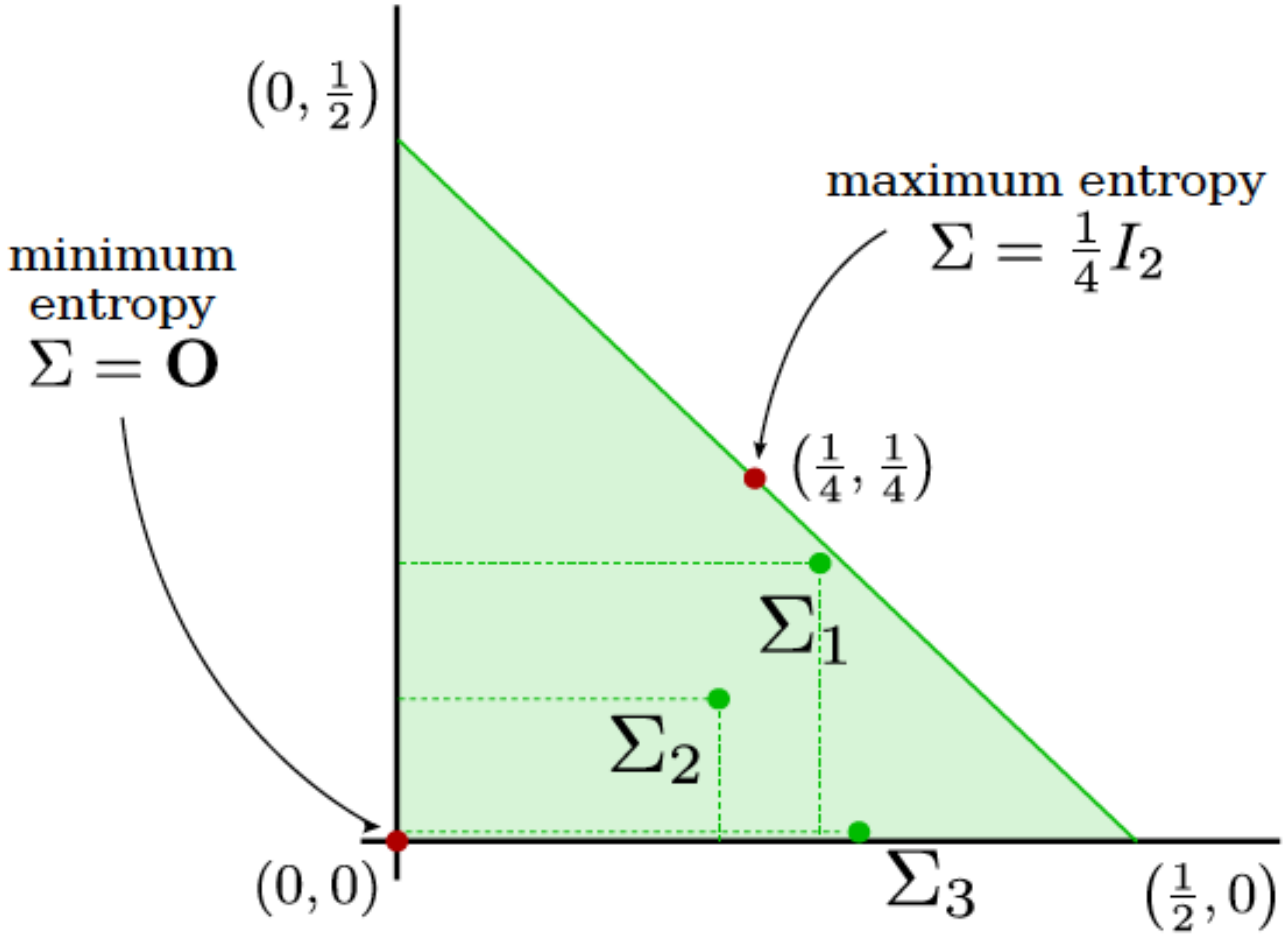
$$\mathrm{P}(\overrightarrow{a_{ij}}) = \mathrm{P}(\overleftarrow{a_{ij}}) \approx \frac{1}{4} + \frac{1}{4(N-1)} \to \frac{1}{4},$$
$$\mathrm{P}(\dot{a_{ij}}) \approx \frac{1}{2} - \frac{1}{2(N-1)} \to \frac{1}{2} \text{ as } N \to \infty$$

and

$$\mathrm{E}(\mathrm{A}_{ij}) = \mathrm{P}(\overrightarrow{a_{ij}}) - \mathrm{P}(\overleftarrow{a_{ij}}) = 0,$$
$$\mathrm{VAR}(\mathrm{A}_{ij}) = 2\mathrm{P}(\overrightarrow{a_{ij}}) \approx \frac{1}{2} + \frac{1}{2(N-1)} \to \frac{1}{2},$$
$$|\mathrm{COV}(\mathrm{A}_{ij}, \mathrm{A}_{kl})| = 2[\mathrm{P}(\overrightarrow{a_{ij}}, \overrightarrow{a_{kl}}) - \mathrm{P}(\overleftarrow{a_{ij}}, \overleftarrow{a_{kl}})]$$
$$\leqq 4\left[\frac{3}{4} - \frac{1}{4(N-1)}\right]^2 \left[\frac{1}{4} + \frac{1}{4(N-1)}\right]^2 \to \frac{9}{64}.$$

with $\mathrm{COV}(\mathrm{A}_{ij}, \mathrm{A}_{jl}) \to \frac{9}{64}$ and $\mathrm{COV}(\mathrm{A}_{ij}, \mathrm{A}_{kl}) = 0$.

A Geometric Representation of Entropy in $\mathcal{L}$



The space of the eigenvalues $\mathcal{L}$ for two edges in an undirected graph.

Univariate Measures of Variability

- The generalised variance, $\mathrm{VAR}_G(\Sigma) = \det(\Sigma) = \prod_{i=1}^{k} \lambda_i \in \left[0, \frac{1}{4^k}\right]$.
- The total variance (or total variability),

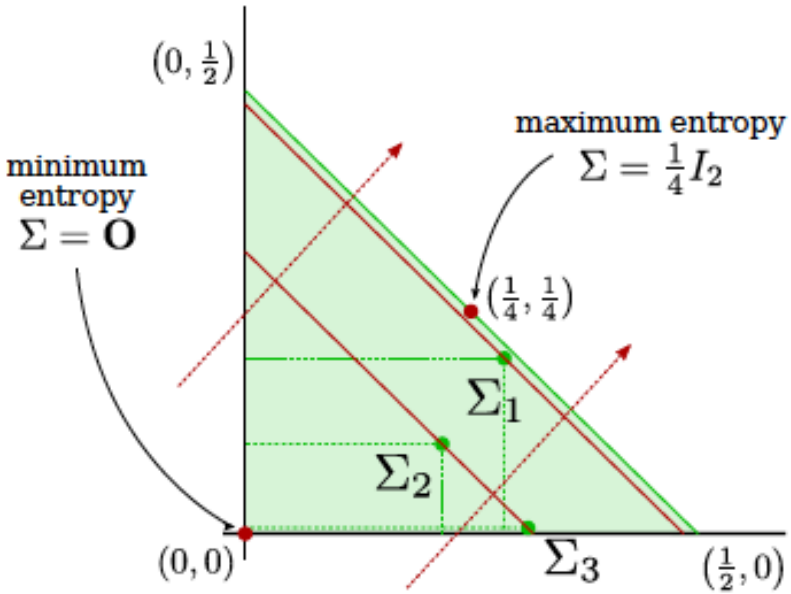
$$\mathrm{VAR}_T(\Sigma) = \mathrm{tr}(\Sigma) = \sum_{i=1}^{k} \lambda_i \in \left[0, \frac{k}{4}\right].$$

- The squared Frobenius matrix norm,

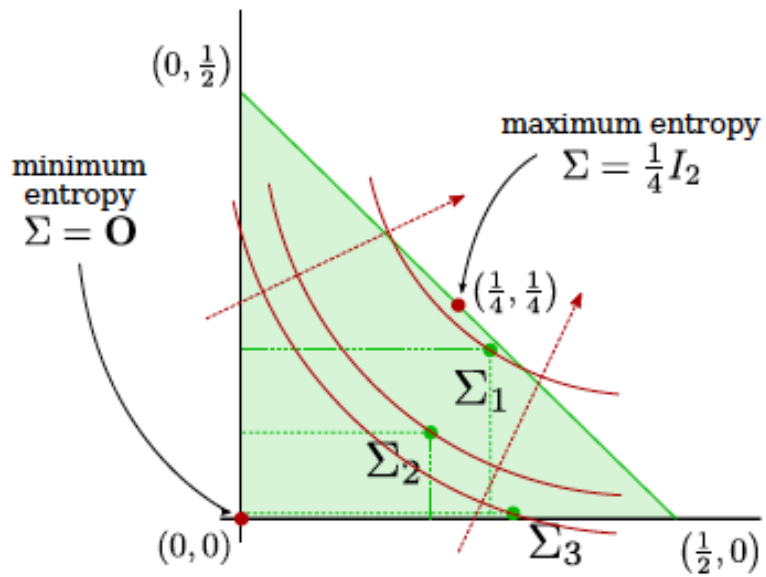
$$\mathrm{VAR}_F(\Sigma) = |||\Sigma - \frac{k}{4} I_k|||_F^2 = \sum_{i=1}^{k} \left(\lambda_i - \frac{k}{4}\right)^2 \in \left[\frac{k(k-1)^2}{16}, \frac{k^3}{16}\right].$$

All of these measures can be rescaled to vary in [0, 1] and to associate high values to networks whose structure displays a high entropy. The equivalent measures of variability for DAGs work in the same way.

Structure Variability: Level Curves



Level curves in $\mathcal{L}$ for $\mathrm{VAR}_T(\Sigma)$.

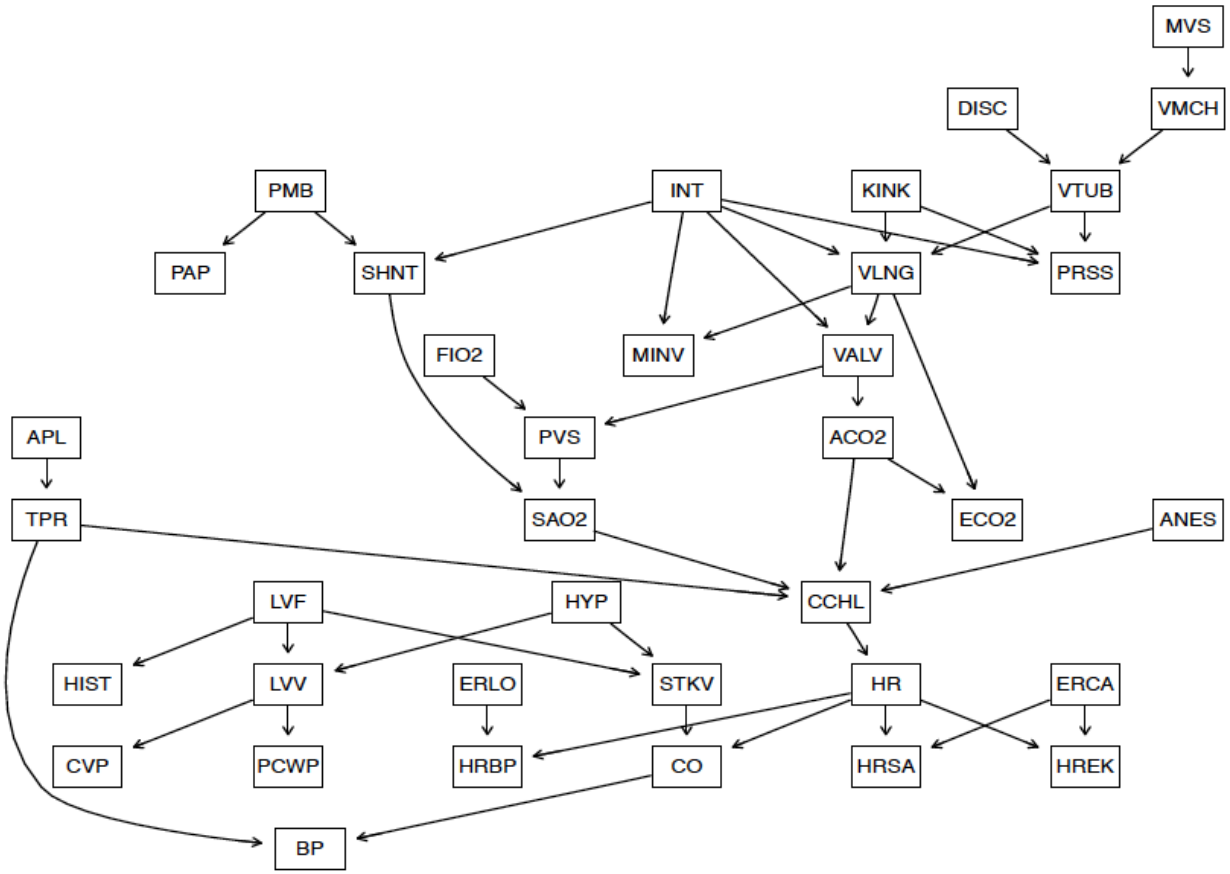


Level curves in $\mathcal{L}$ for $\mathrm{VAR}_F(\Sigma)$.

Pros & Cons About This Approach

- First and second order properties of $\mathrm{P}(\mathcal{G}(\mathcal{E}))$ and $\mathrm{P}(\mathcal{G}(\mathcal{E}) \mid \mathcal{D})$ can be often derived in closed form, and have a geometric interpretation.
- We now have descriptive measures of variability over the space of DAGs; we know that structure learning algorithms are consistent, so we can check how quickly the variability decreases as $n \to \infty$.
- Is there a way of identifying paths using covariance matrix decompositions?
- The covariance matrix $\mathrm{COV}(A_{ij}, A_{kl})$ is very big; so may want to regularise it by shrinking. This affects $\mathrm{p}(a_{ij})$ as well, and it is possible to use it for regularisation purposes. Applications to bayesian model averaging and to identify significant edges?

The ALARM Network



ALARM is a network designed to provide an alarm message system for intensive care unit patient monitoring. It has 37 nodes and 46 edges (of 666 possible edges), and its distribution has 509 parameters.

bnlearn: An Aside, Generating Observations from a BN

ALARM is one of several golden standard networks, which we can download from bnlearn.com to use in **bnlearn**. The fitted BN provides the true DAG of the network, which we can save as an R objects with bn.net().

```
load("alarm.rda")
true.dag = bn.net(bn)
```

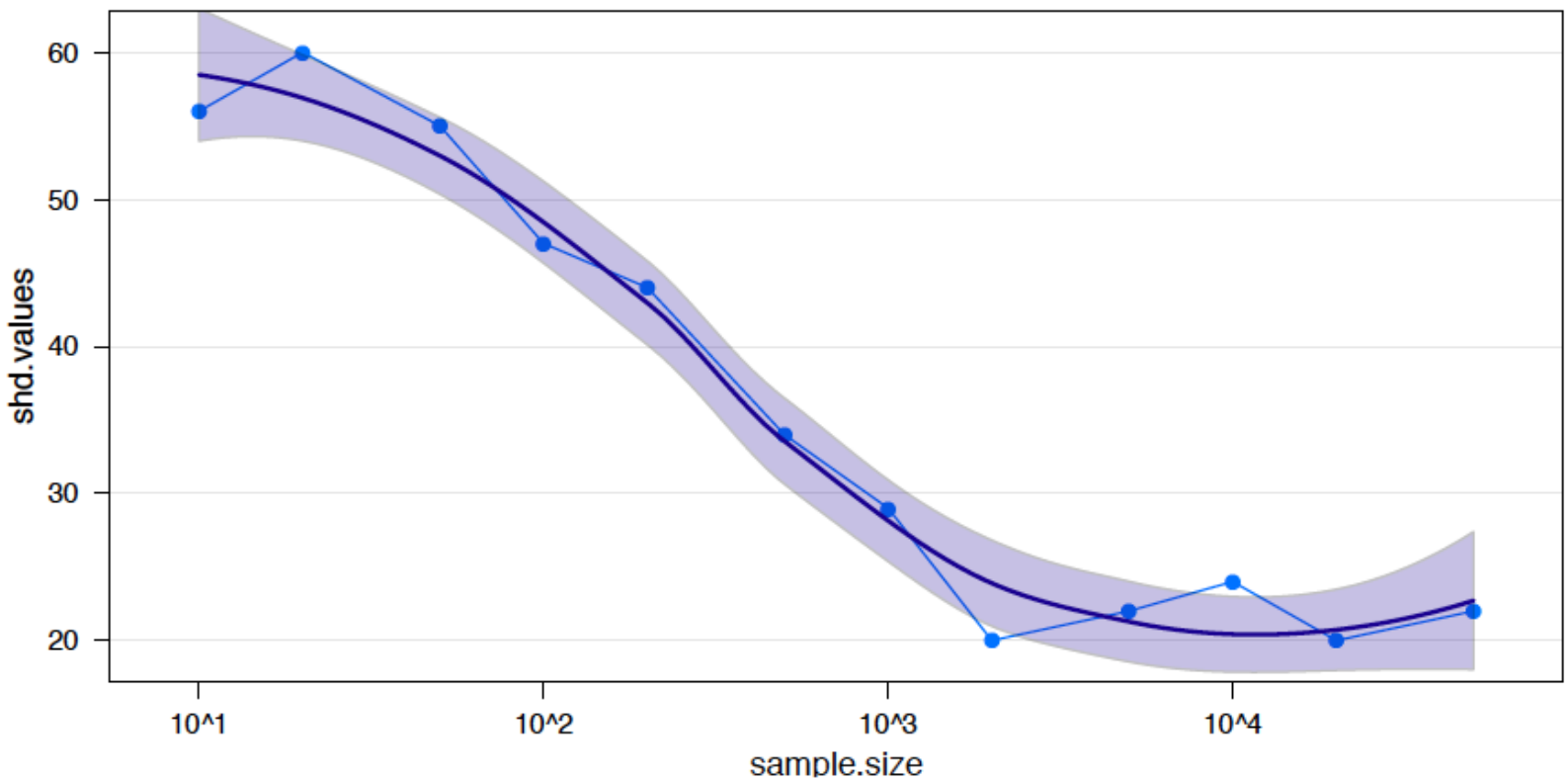
And we can use it to generate random samples from the bN for use in simulations and inference.

```
sim = rbn(bn, 100)
shd(hc(sim), true.dag)
## [1] 51
```

So, with these two functions we can now investigate whether structure learning algorithms are consistent.

So, Are Structure Learning Algorithms Consistent?

```
sample.size = outer(c(1, 2, 5), c(10, 10^2, 10^3, 10^4))
shd.values = numeric(length(sample.size))
for (i in seq_along(sample.size)) {
sim = rbn(bn, sample.size[i])
shd.values[i] = shd(hc(sim), true.dag)
}#FOR
```



bnlearn: Graph Priors in Structure Learning

The posterior scores bDe and bGe accept prior as an additional, optional argument specifying the prior $\mathrm{P}(\mathcal{G}(\mathcal{E}))$. The default is the uniform prior. So

```
unif = hc(alarm, score = "bde", iss = 1)
```

is equivalent to

```
unif = hc(alarm, score = "bde", iss = 1, prior = "uniform")
```

and the uniform graph prior has no tuning arguments.

```
shd(unif, dag)
## [1] 38
```

That is the reason why it was originally chosen as a "default" prior: it does not require prior information on the data and it is computationally very simple.

The Uniform Graph Prior, Revisited

Assuming a uniform prior is problematic because:

- Score-based structure learning algorithms typically generate new candidate DAGs by a single edge addition, deletion or reversal, e.g.

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

  U always simplifies, and that implies $\overrightarrow{p_{ij}} = \overleftarrow{p_{ij}} = \dot{p}_{ij} = \frac{1}{3}$ favouring the inclusion of new edges as $\overrightarrow{p_{ij}} + \overleftarrow{p_{ij}} = \frac{2}{3}$ for each possible edge a_{ij}.

- Two edges are correlated if they are incident on a common node ($\mathrm{COV}(A_{ij}, A_{jl}) \to \frac{9}{64}$), 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.

The Marginal Uniform (MU) Graph Prior

We showed that

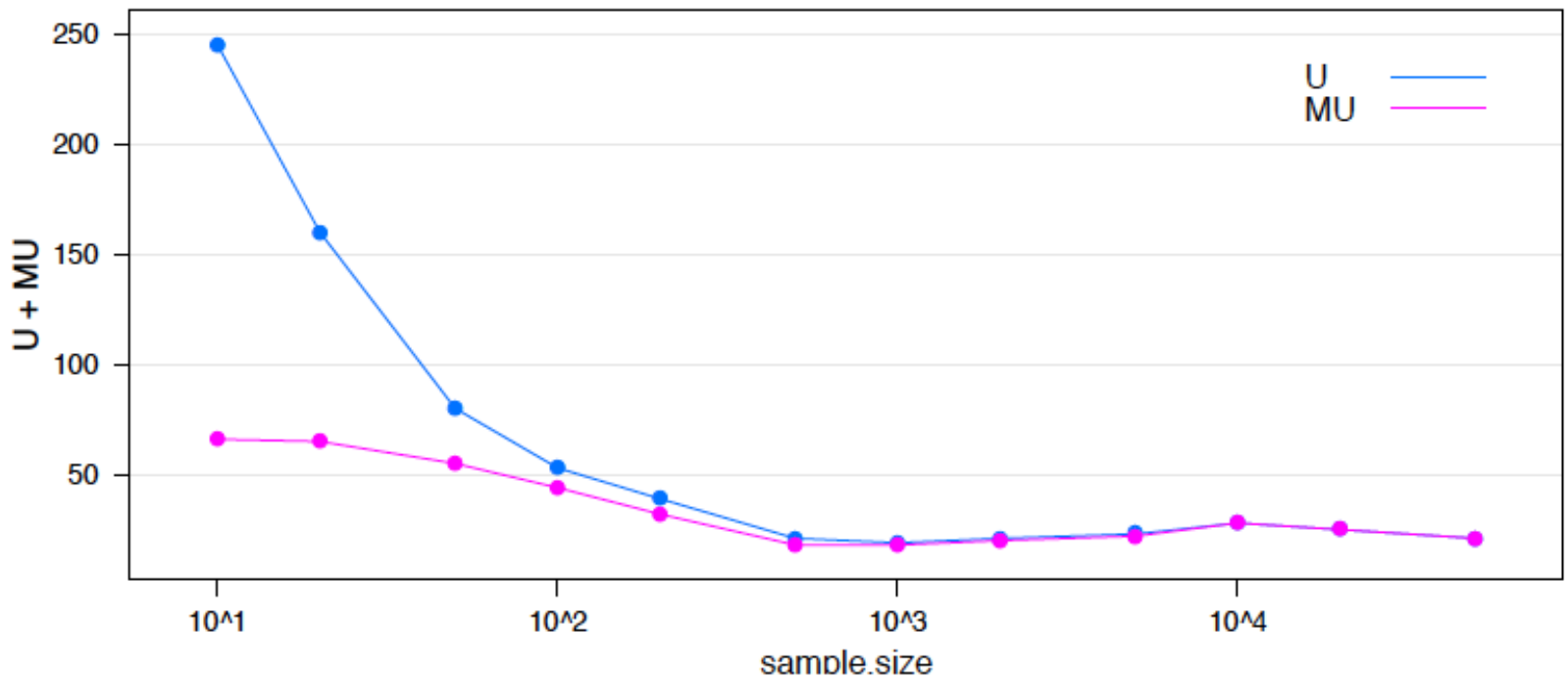
$$\overrightarrow{p_{ij}} = \overleftarrow{p_{ij}} \approx \frac{1}{4} + \frac{1}{4(N-1)} \to \frac{1}{4} \quad \text{and} \quad (\dot{p_{ij}}) \approx \frac{1}{2} - \frac{1}{2(N-1)} \to \frac{1}{2},$$

so each possible edge is present in $\mathcal{G}$. with marginal probability $\approx \frac{1}{2}$ and, when present, it appears in each direction with probability $\frac{1}{2}$. We can use that as a starting point, and assume an independent prior for each edge with the same marginal probabilities (hence the name MU).

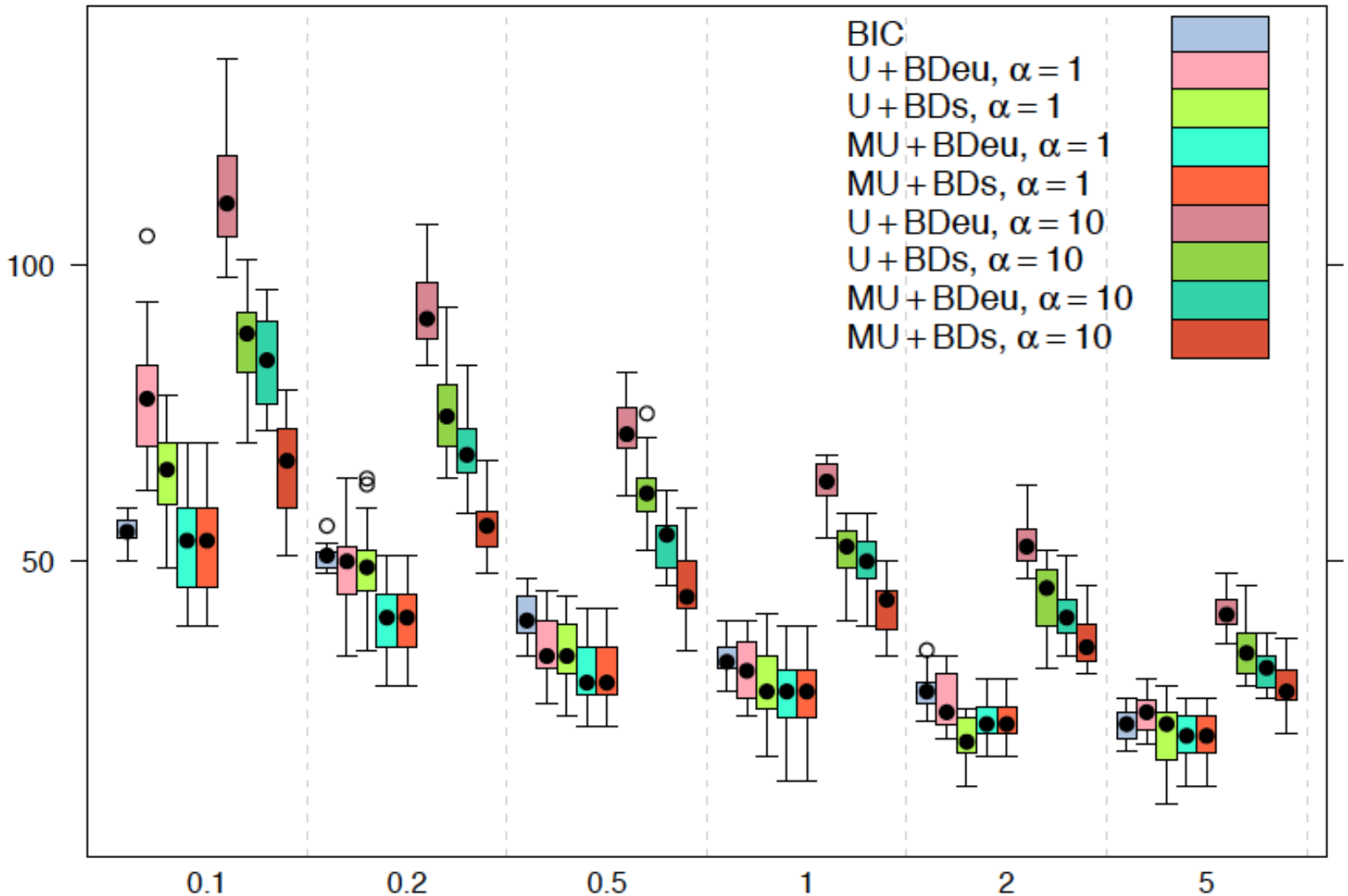
- MU does not favour edge inclusion as $\overrightarrow{p_{ij}} + \overleftarrow{p_{ij}} = \frac{1}{2}$.
- MU does not favour the propagation of errors in structure learning because edges are independent from each other.
- MU computationally trivial to use: the ratio of the prior probabilities is $\frac{1}{2}$ for edge addition, 2 for edge deletion and 1 for edge reversal, for all edges.

bnlearn: A Comparison of Uniform Priors

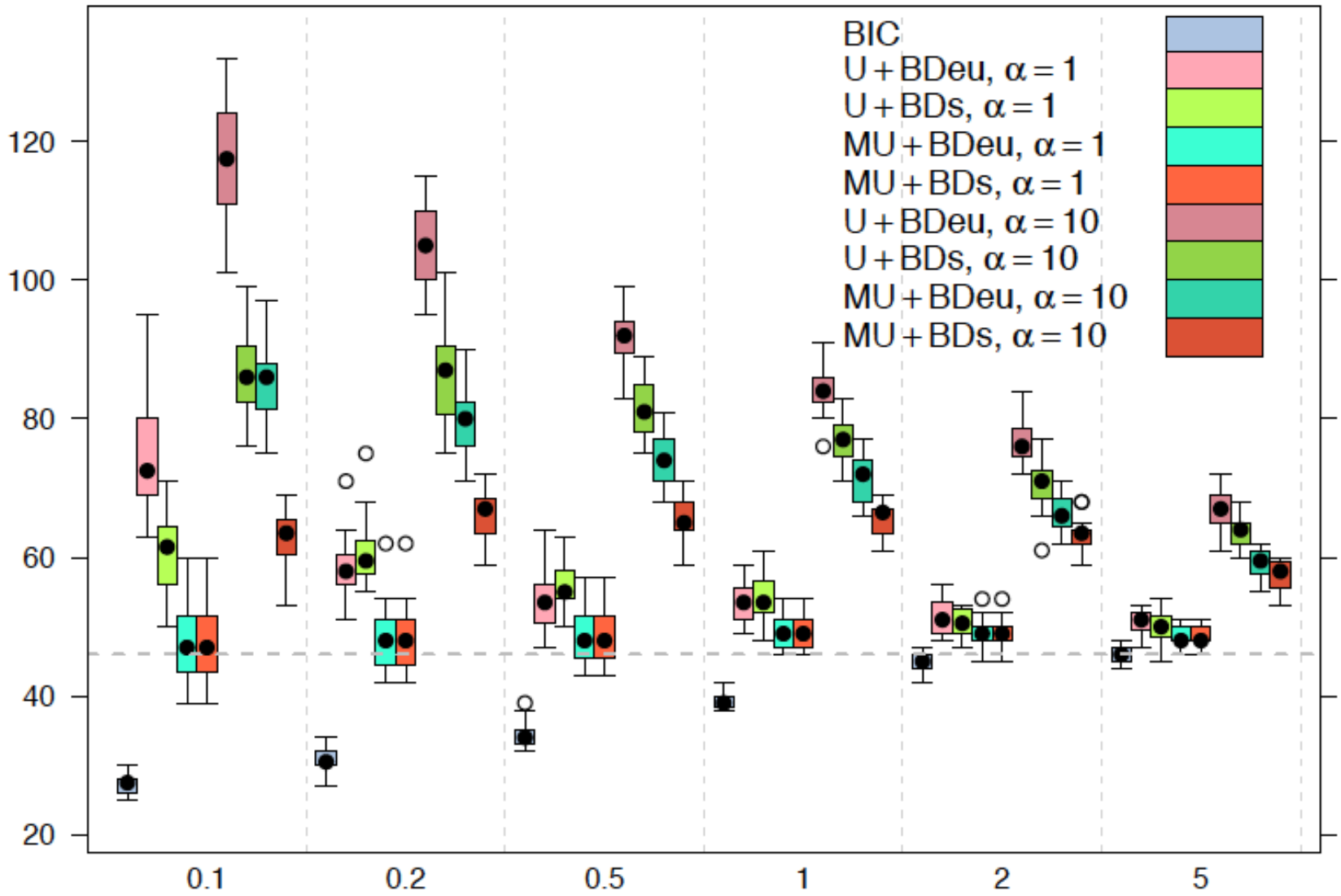
```
shd =
data.frame(sample.size = outer(c(1, 2, 5), c(10, 10^2, 10^3, 10^4)),
U = numeric(length(sample.size)), MU = numeric(length(sample.size)))
for (i in seq_along(sample.size)) {
sim = rbn(bn, sample.size[i])
dagU = hc(sim, score = "bde", iss = 1, prior = "uniform")
dagMU = hc(sim, score = "bde", iss = 1, prior = "marginal")
shd[i, c("U", "MU")] = c(shd(dagU, true.dag), shd(dagMU, true.dag))
}#FOR
```



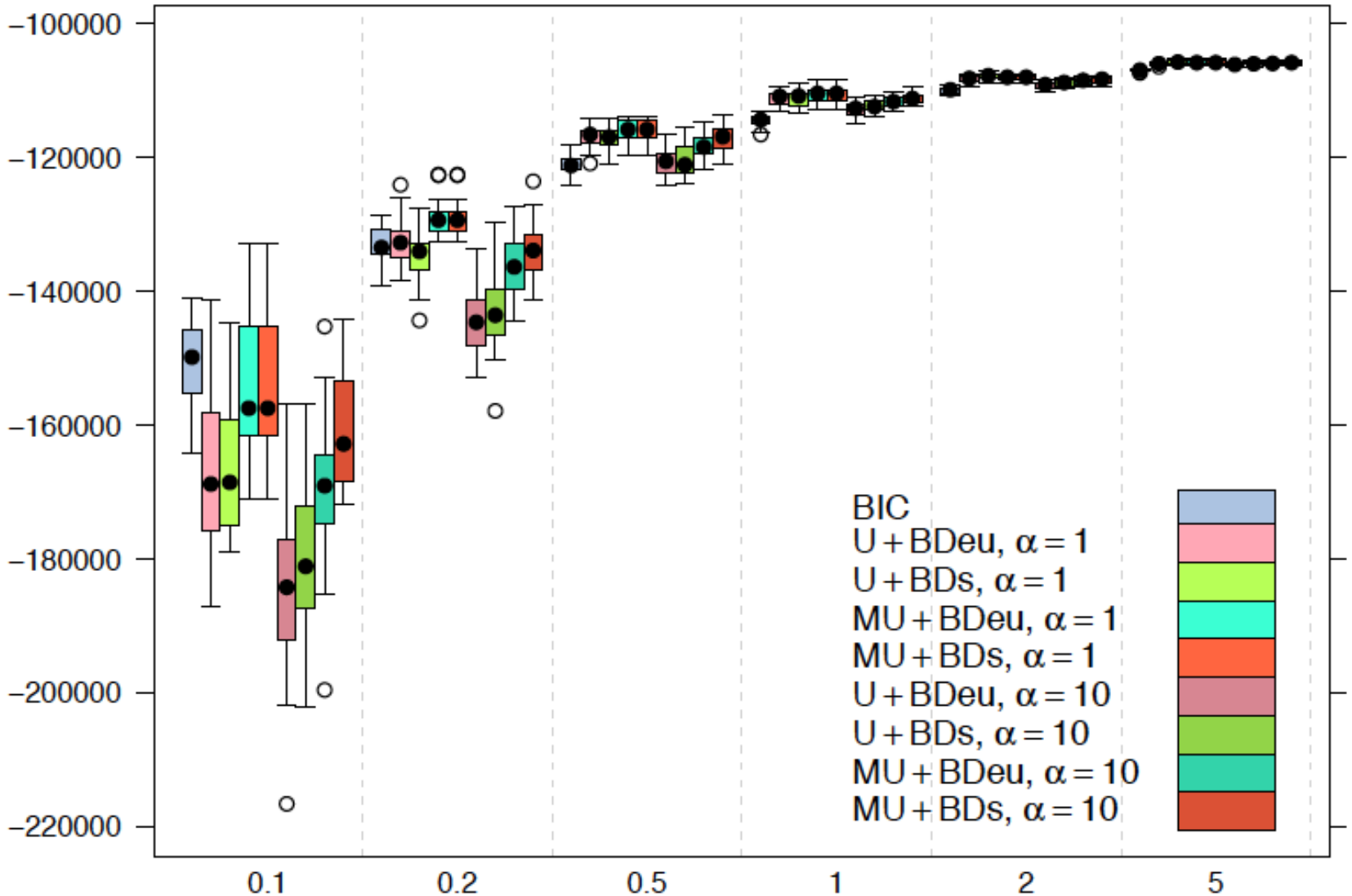
bnlearn: More Simulations (SHD)



bnlearn: More Simulations (edges)



bnlearn: More Simulations (Prediction)



The Castelo & Siebes Marginal Prior

In the marginal uniform prior the probabilities are
xed ; in the general case the Castelo & Siebes marginal prior makes it possible to specify different $\overrightarrow{p_{ij}}, \overleftarrow{p_{ij}}, \dot{p_{ij}}$ for each edge. We can do this in a number of functions in bnlearn by setting prior = "cs" and beta as follows:

```
beta = data.frame(from = c("LVF", "CCHL"), to = c("LVV", "MVS"),
prob = c(0.9, 0.1), stringsAsFactors = FALSE)
beta
## from to prob
## 1 LVF LVV 0.9
## 2 CCHL MVS 0.1
dag.cs = hc(alarm, score = "bde", iss = 1, prior = "cs", beta = beta)
dag.cs$learning$args$beta
## from to aid fwd bkwd
## 1 MVS CCHL 445 0.45 0.10
## 2 LVF LVV 482 0.90 0.05
```

Setting values for any number of edges requires a substantial amount of prior knowledge, and it is easy to get them wrong!

The Variable Selection Prior

We can also borrow the classic variable selection prior from linear regression models, that is,

$$\mathrm{P}(k\text{parents},\ N-k\text{non-parents}) = \frac{\beta^k}{(1-\beta)^{N-k}} \qquad \beta \in (0,1);$$

whether or not a new parent is added to a node is controlled by the corresponding odds

$$\frac{\mathrm{P}(k+1\text{parents},\ N-k-1\text{non-parents})}{\mathrm{P}(k\text{parents},\ N-k\text{non-parents})} = \frac{\beta}{1-\beta}.$$

We can use it by setting prior = "vsp" and beta to β.

```
hc(alarm, score = "bde", iss = 1, prior = "vsp", beta = 0.1)
```

Limiting the Number of Parents

A more drastic measure along the same lines is to put a hard limit on the number of parents of each node, which is implies the prior:

$$\mathrm{P}(\text{adding } (k+1)\text{th parent}) = \begin{cases} \frac{1}{2} & \text{if } k+1 \leq \max p \\ 0 & \text{otherwise} \end{cases}$$

that sets $\mathrm{P}(\mathcal{G}) = 0$ for any $\mathcal{G}$ that has at least one node with more than $\max p$ parents, while all other graphs have the same $\mathrm{P}(\mathcal{G})$.

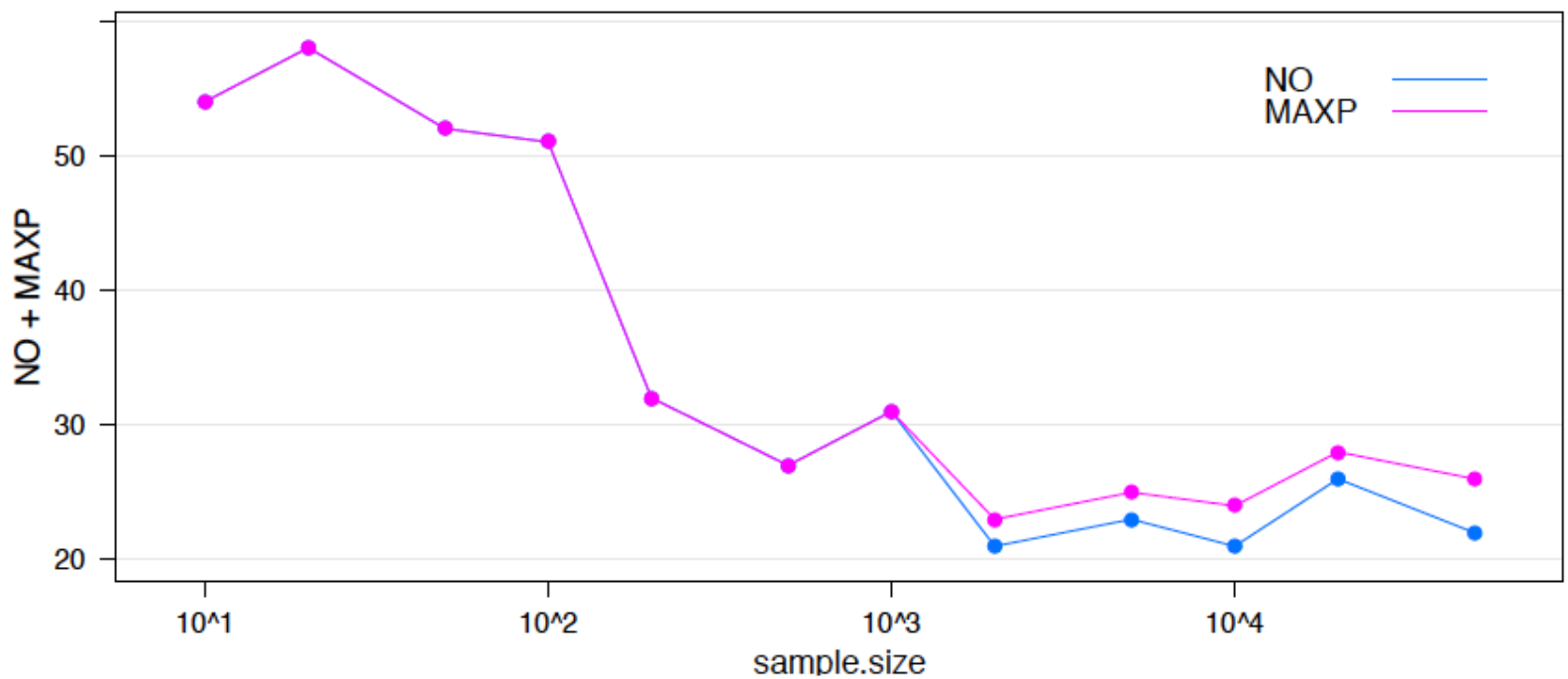
by convention we call sparse a DAG that has $\mathcal{O}(\mathbf{V}) = \mathcal{O}(A)$, so we usually want to set $\max p \in [1, 4]$ (1 forces DAGs to be trees):

```
hc(alarm, score = "bde", iss = 1, maxp = 3)
hc(alarm, score = "bic", maxp = 3)
```

Customarily, this has been used in the literature with all kinds of scores, so the maxp argument is available for use with any score in **bnlearn**.

bnlearn: It Can Make Things Worse If You Set It Too Low

```
shd =
data.frame(sample.size = outer(c(1, 2, 5), c(10, 10^2, 10^3, 10^4)),
NO = numeric(length(sample.size)), MAXP = numeric(length(sample.size)))
for (i in seq_along(sample.size)) {
sim = rbn(bn, sample.size[i])
dagNO = hc(sim, score = "bic")
dagMAXP = hc(sim, score = "bic", maxp = 2)
shd[i, c("NO", "MAXP")] = c(shd(dagNO, true.dag), shd(dagMAXP, true.dag))
}#FOR
```



Whitelisting and blacklisting

A more granular application of this kind of hard prior constraints leads to the use of whitelists and blacklists:

- edges blacklisted in one direction only (i.e. $A \to B$ is blacklisted but $B \to A$ is not) are never present in that particular direction, but may be present in the other direction.
- edges blacklisted in both directions (i.e. both $A \to B$ and $B \to A$ are blacklisted) are never present in the graph, even as an undirected edge in a CPDAG.
- edges whitelisted in one direction only (i.e. $A \to B$ is whitelisted but $B \to A$ is not) have the respective reverse edges blacklisted, and are always present in the graph.
- edges whitelisted in both directions (i.e. both $A \to B$ and $B \to A$ are whitelisted) are present in the graph, but their direction is set by the learning algorithm.

Any edge whitelisted and blacklisted at the same time is assumed to be whitelisted, and is thus removed from the blacklist.

bnlearn: Whitelists and Blacklists (I)

All structure learning algorithms in **bnlearn** have a whitelist and a blacklist arguments, that are interpreted as appropriate in terms of directed and undirected edges at various stages of the algorithms.

In score-based algorithms, individual edges are whitelisted and blacklisted.

```
head(edges(hc(alarm)), n = 4)
## from to
## [1,] "PCWP" "LVV"
## [2,] "HRBP" "HR"
## [3,] "MINV" "VALV"
## [4,] "HR" "HREK"
bl = data.frame(from = c("HRBP", "MINV"), to = c("HR", "VALV"))
head(edges(hc(alarm, blacklist = bl)), n = 4)
## from to
## [1,] "PCWP" "LVV"
## [2,] "HREK" "HRSA"
## [3,] "HR" "HRBP"
## [4,] "HREK" "HR"
```

bnlearn: Whitelists and Blacklists (II)

In constraint-based algorithms, edges must be blacklisted in both directions to prevent them from being included in Markov blankets and neighbour sets; whitelists work normally.

```
head(edges(si.hiton.pc(alarm)), n = 3)
## from to
## [1,] "CVP" "LVV"
## [2,] "PCWP" "LVV"
## [3,] "HIST" "LVF"
bl = data.frame(from = c("PCWP"), to = c("LVV"))
head(edges(si.hiton.pc(alarm, blacklist = bl)), n = 3)
## from to
## [1,] "CVP" "LVV"
## [2,] "PCWP" "LVV"
## [3,] "HIST" "LVF"
bl = data.frame(from = c("PCWP", "LVV"), to = c("LVV", "PCWP"))
head(edges(si.hiton.pc(alarm, blacklist = bl)), n = 3)
## from to
## [1,] "CVP" "LVV"
## [2,] "PCWP" "LVF"
## [3,] "HIST" "LVF"
```

Parameter Learning: Likelihood, Bayesian and Shrinkage

Once 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 the local distributions, 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: 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 coeffcients are a bad choice for almost all real-world problems. They are still around because:

- they are used in benchmark simulations;
- computer scientists do not care much about parameter estimation.

However:

- 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, i.e.

$$Dir(\alpha_{k|x_{\mathrm{Pa}_{(j)}}=\pi}) \quad \xrightarrow{data} \quad Dir(\alpha_{k|x_{\mathrm{Pa}_{(j)}}=\pi} + n_{k|x_{\mathrm{Pa}_{(j)}}=\pi})$$

  meaning that $\hat{p}_{k|x_{\mathrm{Pa}_{(j)}}=\pi} = \alpha_{k|x_{\mathrm{Pa}_{(j)}}=\pi} / \sum_{\pi} \alpha_{k|x_{\mathrm{Pa}_{(j)}}=\pi}$.

- the Inverse Wishart for Gaussian models, i.e.

$$IW(\psi, m) \quad \xrightarrow{data} \quad IW(\psi + n\Sigma, m + n).$$

In both cases (when a non-informative prior is used) the only free parameter is the equivalent or imaginary sample size, which gives the relative weight of the prior compared to the observed sample. Medgeo Scutari University of

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 coeffcients,

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

Ridge Regression corresponds to a Gaussian prior,

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

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. Also, excessive regularisation might lead to zero coeffcients that would make a node independent of its parents.

Shrinkage, James-Stein Estimation

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.

Shrinkage, James-Stein Estimation

For discrete data, conditional probabilities $p_{k|\pi} = p_{k|X_{\mathrm{Pa}(j)}=\pi}$

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

where t is the uniform (discrete) distribution.

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

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

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

bnlearn: Parameter Learning, DBNs

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(hc(asia), asia, method = "mle")
coef(fitted$X)
## E
## X no yes
## no 0.95659 0.00541
## yes 0.04341 0.99459
fitted = bn.fit(hc(asia), asia, method = "bayes", iss = 20)
coef(fitted$X)
## E
## X no yes
## no 0.9556 0.0184
## yes 0.0444 0.9816
```

bnlearn: Parameter Learning, GBNs

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

```
library(penalized)
fitted = bn.fit(hc(marks), marks)
coef(fitted$ALG)
## (Intercept) MECH VECT
##      25.362 0.183 0.358
fitted$ALG = penalized(response = marks[, "ALG"],
penalized = marks[, parents(fitted, "ALG")],
lambda2 = 100, model = "linear", trace = FALSE)
coef(fitted$ALG)
## (Intercept) MECH VECT
##      25.481 0.184 0.355
```

We can also fit the parameters directly using penalized() and a DAG, and collect them in a BN with custom.fit().

Model Averaging: Frequentist, Bayesian and Hybrid

The results of both structure learning and parameter learning should be validated before using a BN for inference. Since parameters are learned conditional on the results of structure learning, validating the (CP)DAG learned from the data would be the first step.

- frequentist: generating network structures using bootstrap and model averaging (aka bagging).
- Bayesian: generating network structures from the posterior $\mathrm{P}(\mathcal{G} \mid \mathcal{D})$ using exhaustive enumeration or Markov Chain Mote Carlo approximations.
- hybrid: generating network structures again using bootstrap, but weighting them with their posterior probabilities when performing model averaging.

A Frequentist Approach: Friedman's Confidence

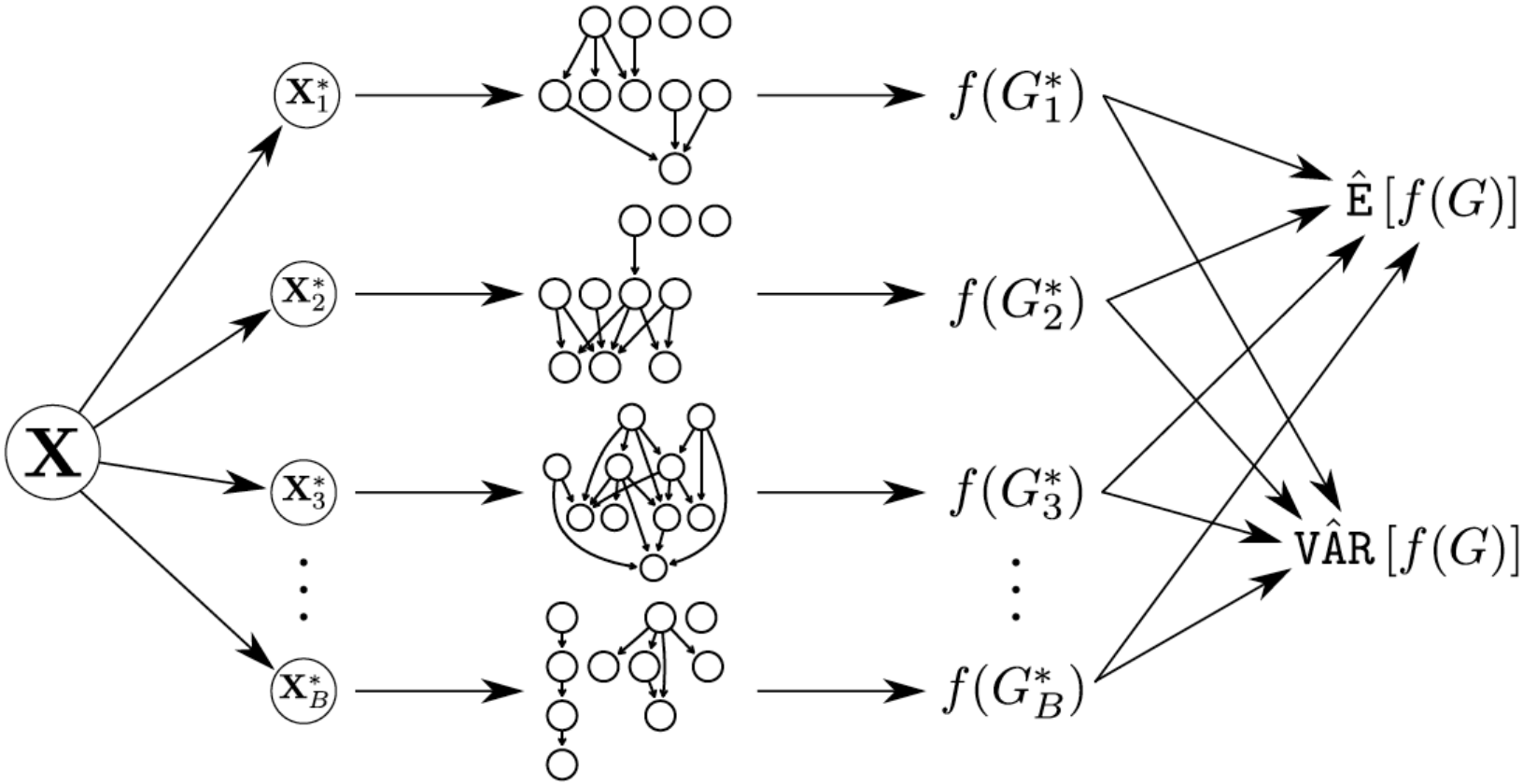
Friedman et al. proposed an approach to model validation based on bootstrap resampling and model averaging:

1. For $b = 1, 2, \ldots, B$:
   - 1.1 sample a new data set $\mathcal{D}_b^*$ from the original data $\mathcal{D}$ using either parametric or nonparametric bootstrap;
   - 1.2 learn the structure of the BN $\mathcal{G}_b = (\mathbf{V}, A_b)$ from $\mathcal{D}_b^*$.
2. Estimate the strength or confidence that each possible edge a_i is present in the true DAG $\mathcal{G}_0 = (\mathbf{V}, A_0)$ as

$$\hat{p}_i = \hat{\mathrm{P}}(a_i) = \frac{1}{B} \sum_{b=1}^{B} \mathbb{1}_{\{e_i \in A_b\}},$$

where $\mathbb{1}_{\{e_i \in A_b\}}$ is equal to 1 if $e_i \in E_b$ and 0 otherwise.

A Frequentist Approach: Friedman's Confidence



bnlearn: edge Strength

This approach is implemented in boot.strength(), which takes a data set $\mathcal{D}$, a structure learning algorithm and its algorithm.args, and performs bootstrap resampling R times.

```
str = boot.strength(alarm, algorithm = "hc",
algorithm.args = list(score = "bde", iss = 1), R = 100)
head(str[str$strength > 0.50, ])
##     from   to strength direction
## 24   CVP  LVV        1     0.160
## 53  PCWP  LVF        1     0.165
## 60  PCWP  LVV        1     0.510
## 89  HIST  LVF        1     0.755
## 112  TPR   BP        1     1.000
## 118  TPR SAO2        1     0.000
```

The return value has two strength measures, strength and direction, representing

$$\mathrm{P}(\overrightarrow{p}_{ij} + \overleftarrow{p}_{ij}) \qquad \text{and} \qquad \mathrm{P}(\overrightarrow{p}_{ij} \mid \overrightarrow{p}_{ij} + \overleftarrow{p}_{ij}).$$

A (Full) Bayesian Approach

Performing a full posterior Bayesian analysis on DAGs, that is, working with

$$\hat{p}_i = \mathrm{E}(e_i \mid \mathcal{D}) = \sum_{\mathcal{G}} \mathbb{1}_{\{e_i \in E_{\mathcal{G}}\}} \mathrm{P}(\mathcal{G} \mid \mathcal{D}),$$

is considered unfeasible for DAGs with more than ≈ 10 nodes because:

- an exhaustive enumeration takes too long, and it's even worse for BNs because of the acyclicity constraint;
- generating DAGs from the posterior distribution is feasible but convergence of the MCMC to the stationary distribution is far from certain (mixing is often too slow).

A Hybrid Approach: the "Bayesian confidence"

Friedman's confidence and Bayesian posterior analysis may be combined as follows:

1. For $b = 1, 2, \ldots, B$:
   - 1.1 sample a new data set $\mathcal{D}_b^*$ from the original data $\mathcal{D}$ using either parametric or nonparametric bootstrap;
   - 1.2 learn the structure of the graphical model $\mathcal{G}_b = (\mathbf{V}, E_b)$ from $\mathcal{D}_b^*$.
2. Estimate the strength confidence for each possible edge e_i as

$$\hat{p}_i = \mathrm{E}(e_i \mid \mathcal{D}) \approx \frac{1}{B} \sum_{\mathcal{G}} \mathbb{1}_{\{e_i \in E_b\}} \mathrm{P}(\mathcal{G}_b \mid \mathcal{D}).$$

The result is a form of approximate Bayesian estimation, whose behaviour depends on how much of the posterior probability mass is concentrated in the subset of DAGs $\mathcal{G}_b$.

bnlearn: edge Strength and Weights (I)

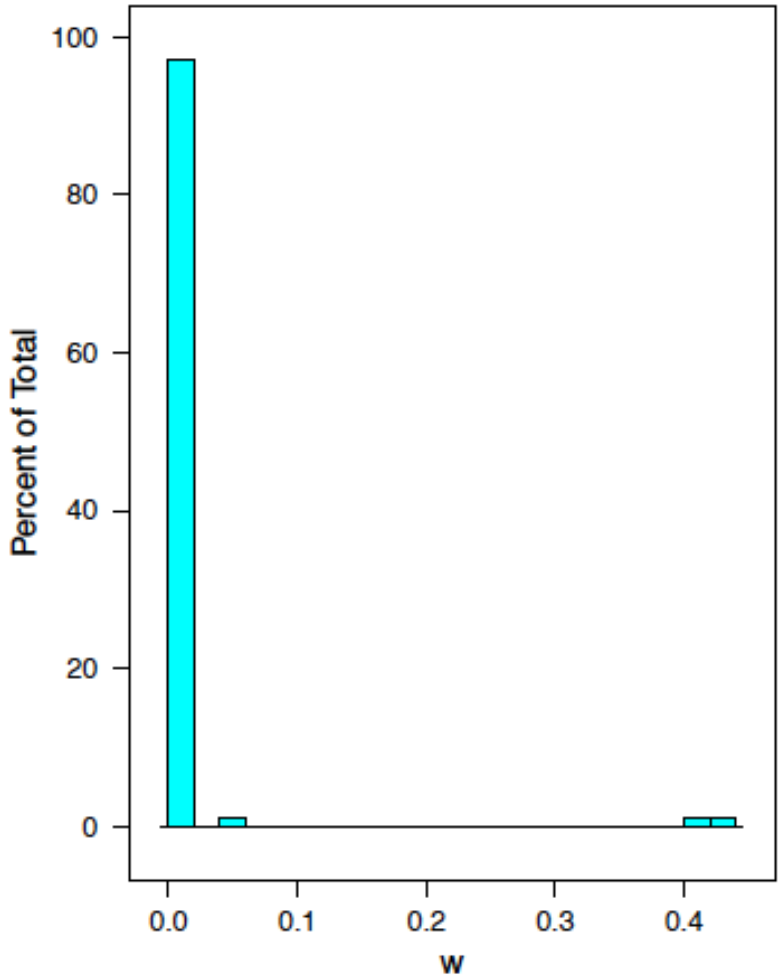
This approach requires two separate steps:

1. we can estimate the $\mathcal{G}_b$ with bn.boot(), without computing any statistic on them (the I() function does literally nothing);
2. and then we can iterate with sapply() over the DAGs to compute the $\mathrm{P}(\mathcal{G}_b \mid \mathcal{D})$.

```
Gb = bn.boot(alarm, algorithm = "hc", statistic = I,
algorithm.args = list(score = "bde", iss = 1), R = 100)
w = sapply(Gb, score, data = alarm, type = "bde", iss = 1)
library(Rmpfr)
w = mpfr(w, precBits = 160)
w = asNumeric(exp(w) / sum(exp(w)))
wstr = custom.strength(Gb, weights = w, nodes = names(alarm))
```

Note that score() returns $\log \mathrm{BDe}(\mathcal{G}_b)$ but we need $\exp(\log \mathrm{BDe}(\mathcal{G}_b))$; the $\log \mathrm{BDe}(\mathcal{G}_b)$ are so small that it impossible to exponentiate them without using an arbitrary precision library.

bnlearn: edge Strength and Weights (II)



Unfortunately, for any middle-sized and large BN (say, 10 or more nodes) the $\mathrm{P}(\mathcal{G}_b \mid \mathcal{D})$ will be so small that once normalised only 1-3 weights will be significantly different from zero.

The reason is that the space of the possible DAGs is extremely large and $\mathrm{P}(\mathcal{G}(\mathcal{E}) \mid \mathcal{D})$ will be extremely at, so $\mathrm{P}(\mathcal{G}_b \mid \mathcal{D}) \to 0$, with a few networks having values e.g. 10^{-200} compared to e.g. 10^{-205} for the rest.

Identifying Significant edges

- The con
  dence values $\hat{\mathbf{p}} = \{\hat{p}_i\}$ do not sum to one and are dependent on one another in a nontrivial way; the value of the confidence threshold (i.e. the minimum confidence for an edge to be accepted as an edge of $\mathcal{G}_0$ regardless of direction) is an unknown function of both the data and the structure learning algorithm.
- The ideal/asymptotic configuration $\tilde{\mathbf{p}}$ of confidence values would be

$$\tilde{p}_i = \begin{cases} 1 & \text{if } e_i \in E_0 \\ 0 & \text{otherwise,} \end{cases}$$

  i.e. all the networks $\mathcal{G}_b$ have exactly the same structure.
- Therefore, identifying the configuration $\tilde{\mathbf{p}}$ "closest" to $\hat{\mathbf{p}}$ provides a principled way of identifying significant edges and the confidence threshold.

The Confidence Threshold

Consider the order statistics $\tilde{\mathbf{p}}_{(.)}$ and $\hat{\mathbf{p}}_{(.)}$ and the cumulative distribution functions (CDFs) of their elements:

$$F_{\hat{\mathbf{p}}_{(.)}}(x) = \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}_{\{\hat{p}_{(i)} < x\}}$$

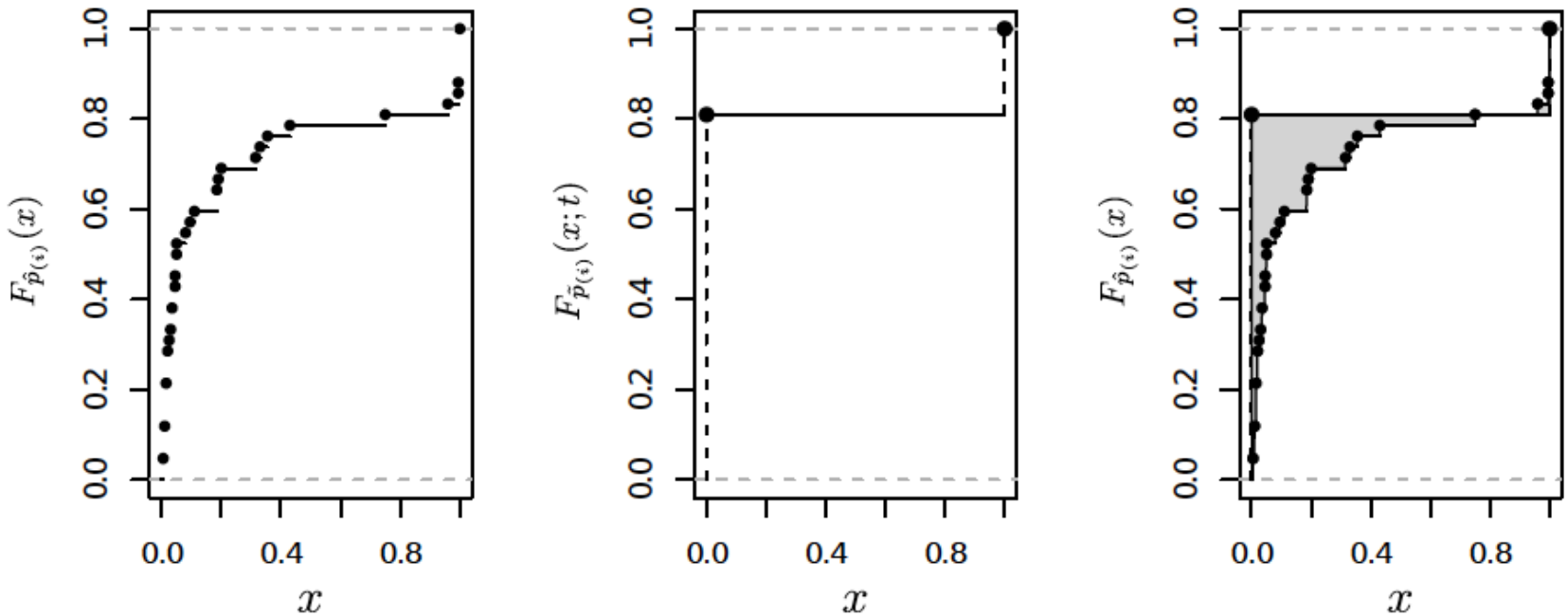
and

$$F_{\tilde{\mathbf{p}}_{(.)}}(x; t) = \begin{cases} 0 & \text{if } x \in (-\infty, 0) \\ t & \text{if } x \in [0, 1) \\ 1 & \text{if } x \in [1, +\infty) \end{cases} .$$

t corresponds to the fraction of elements of $\tilde{\mathbf{p}}_{(.)}$ equal to zero and is a measure of the fraction of non-significant edges, and provides a threshold for separating the elements of $\tilde{\mathbf{p}}_{(.)}$:

$$e_{(i)} \in E_0 \iff \hat{p}_{(i)} > F^{-1}_{\tilde{\mathbf{p}}_{(.)}}(t).$$

The CDFs $F_{\hat{\mathbf{p}}_{(\cdot)}}(x)$ and $F_{\tilde{\mathbf{p}}_{(\cdot)}}(x;t)$



One possible estimate of t is the value $\hat{t}$ that minimises some distance between $F_{\hat{\mathbf{p}}_{(\cdot)}}(x)$ and $F_{\tilde{\mathbf{p}}_{(\cdot)}}(x;t)$; an intuitive choice is using the L_1 norm of their difference (i.e. the shaded area in the picture on the right).

An L_1 Estimator for the Confidence Threshold

Since $F_{\hat{\mathbf{p}}_{(\cdot)}}(x)$ is piece-wise constant and $F_{\tilde{\mathbf{p}}_{(\cdot)}}(x;t)$ is constant in $[0, 1]$, the L_1 norm of their difference simplifies to

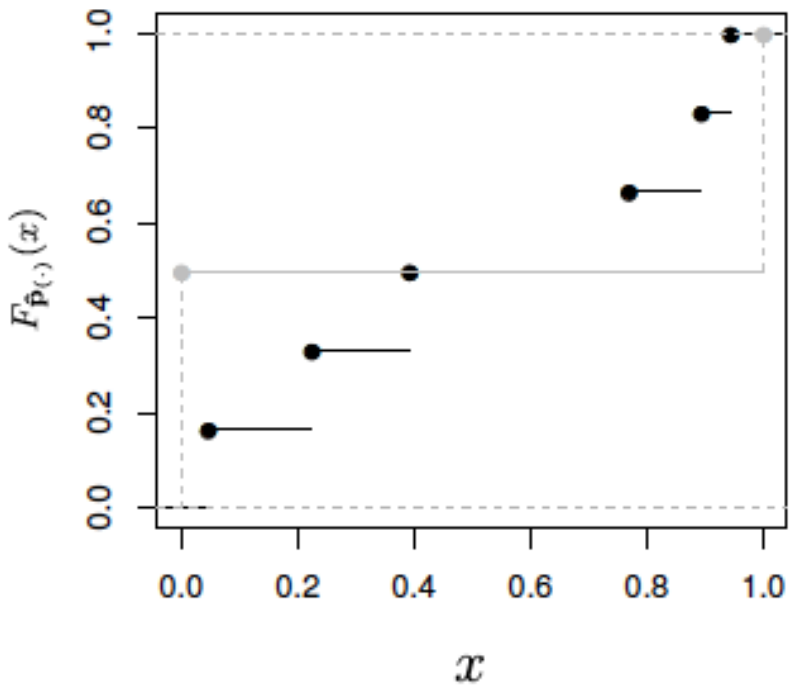
$$\begin{aligned} L_1(t;\hat{\mathbf{p}}_{(\cdot)}) &= \int |F_{\hat{\mathbf{p}}_{(\cdot)}}(x) - F_{\tilde{\mathbf{p}}_{(\cdot)}}(x;t)|dx \\ &= \sum_{x_i \in \{\{0\} \cup \hat{\mathbf{p}}_{(\cdot)} \cup \{1\}\}} |F_{\hat{\mathbf{p}}_{(\cdot)}}(x_i) - t|(x_{i+1} - x_i). \end{aligned}$$

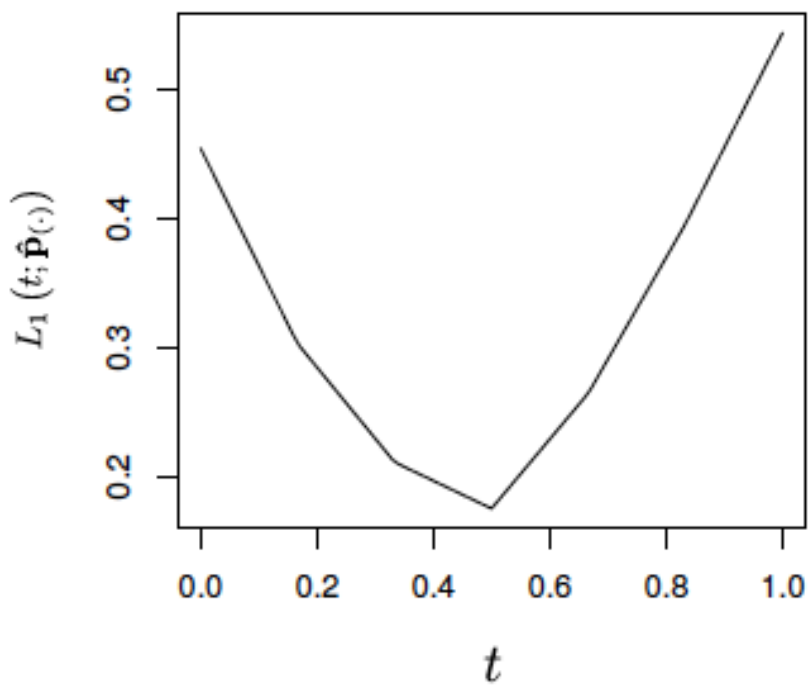
This form has two important properties:

- can be computed in linear time from $\hat{\mathbf{p}}_{(\cdot)}$;
- its minimisation is straightforward using linear programming.

Furthermore, the L_1 norm does not place as much weight on large deviations as other norms (L_2, L_∞), making it robust against a wide variety of configurations of $\hat{\mathbf{p}}_{(\cdot)}$.

A Simple Example





Consider a graph with 4 nodes and confidence values

$$\hat{\mathbf{p}}_{(\cdot)} = \{0.0460, 0.2242, 0.3921, 0.7689, 0.8935, 0.9439\}.$$

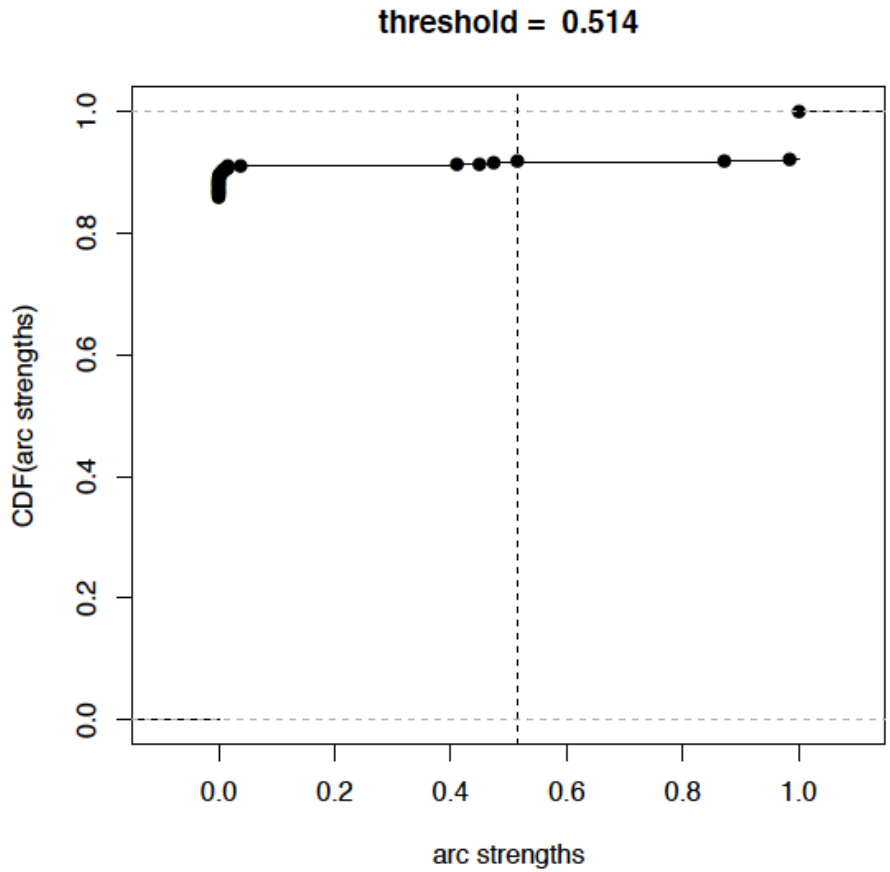
Then $\hat{t} = \min_t L_1(t; \hat{\mathbf{p}}_{(\cdot)}) = 0.4999816$ and $F^{-1}_{\tilde{\mathbf{p}}_{(\cdot)}}(0.4999816) = 0.3921$; only three edges are considered significant.

bnlearn: Model Averaging with averaged.network()

```
averaged.network(wstr)
##
## Random/Generated Bayesian network
##
## model:
##   [partially directed graph]
## nodes:                          37
## edges:                           55
##   undirected edges:              3
##   directed edges:                52
## average markov blanket size: 3.57
## average neighbourhood size:  2.97
## average branching factor:    1.35
##
## generation algorithm:        Model Averaging
## significance threshold:      0.514
head(wstr[wstr$strength > 0.514 & wstr$direction >= 0.50, ], n = 3)
##    from  to strength direction
## 60 PCWP LVV        1       0.5
## 112 TPR  BP        1       1.0
## 126 TPR APL        1       1.0
```

bnlearn: Plotting the ECDF

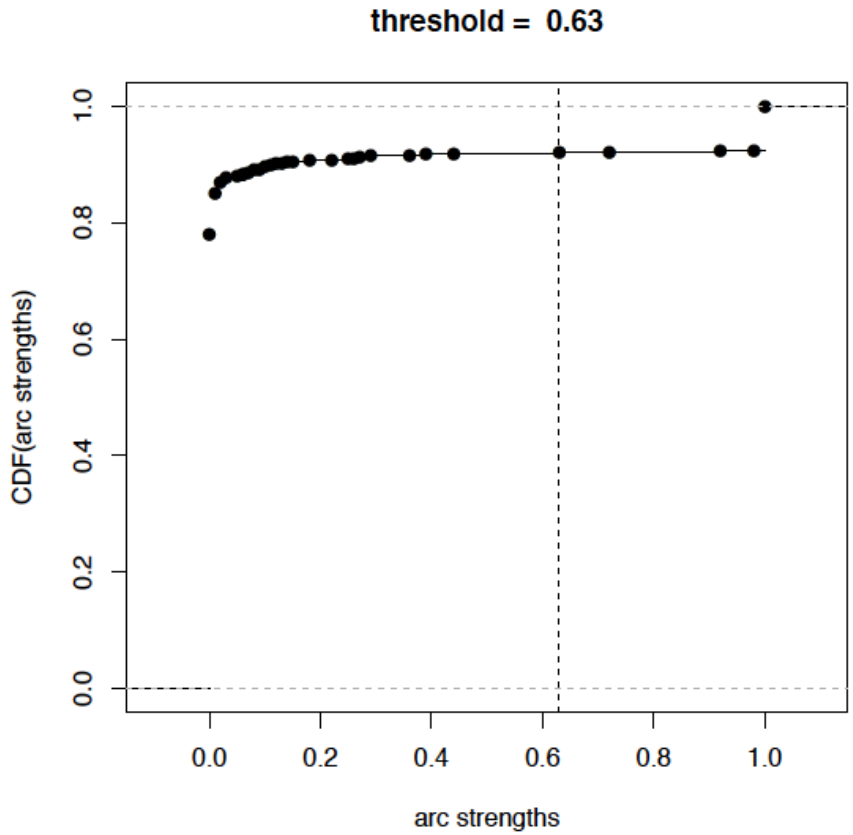
```
plot(wstr)
```



The effect of the uneven posterior probability is apparent from the fact that the edge weights are essentially either zero or one.

bnlearn: Plotting the ECDF

```
plot(str)
```



With the frequentist approach the weights are more spread out, and the threshold is different as a result.

bnlearn: Custom Thresholds

averaged.network() accepts custom values for the threshold, so we can investigate its on the resulting (CP)DAG.

```
unlist(compare(averaged.network(wstr), true.dag))
## tp fp fn
## 23 23 32
unlist(compare(averaged.network(str), true.dag))
## tp fp fn
## 22 24 31
unlist(compare(averaged.network(str, threshold = 0.4), true.dag))
## tp fp fn
## 22 24 33
unlist(compare(averaged.network(str, threshold = 0.8), true.dag))
## tp fp fn
## 22 24 30
```

There is not guarantee that the L_1 norm with produce the best DAG, say, that with the lowest SHD, but simulations and real-world data analyses suggest it performs well enough for practical purposes.

Summary

- Scoring the DAGs we evaluate in structure learning algorithms is crucial, but so are our assumptions on their prior probability.
- We can incorporate prior knowledge in structure learning in many ways with hard constraints (edges being present or absent, maximum number of edges) and/or informative priors (probability of parents and edges). If the prior knowledge we have is not wrong, this augments the information present in the data and improves the quality of the BN.
- Even if we have no prior knowledge, we can do better than assuming a uniform prior.
- Estimating the parameters of a BN given the DAG is comparatively easy; smooth estimates are preferable over maximum likelihood estimates as usual.
- We can use resampling to remove noisy edges with model averaging, typically along the lines of bagging. Averaged models tend to be more robust and better at prediction.

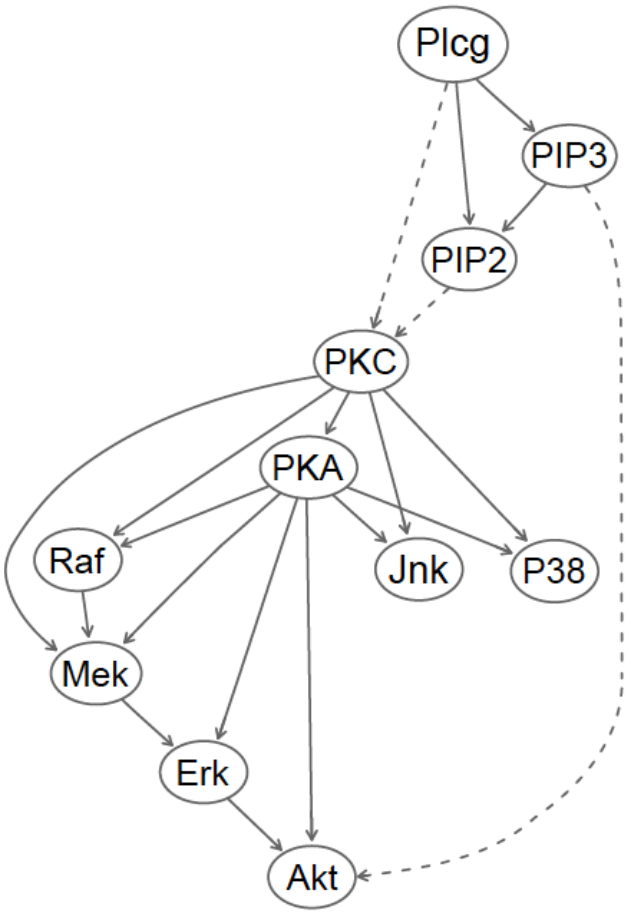
Hands-On Examples

Case Study: Human Physiology

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

That is a landmark application of BNs because it highlights the use of interventional data; and because results are validated. The data consist in the 5400 simultaneous measurements of 11 phosphorylated proteins and phospholypids; 1800 are subjected to spiking and knock-outs to control expression.

The goal of the analysis is to learn what relationships link these 11 proteins, that is, the signalling pathways they are part of.



Exploring the Data

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

The variables represent concentrations of the proteins and the phospholypids, and take positive values. For some variables, and observations, the cells were stimulated to produce artificially high or low levels of particular proteins:

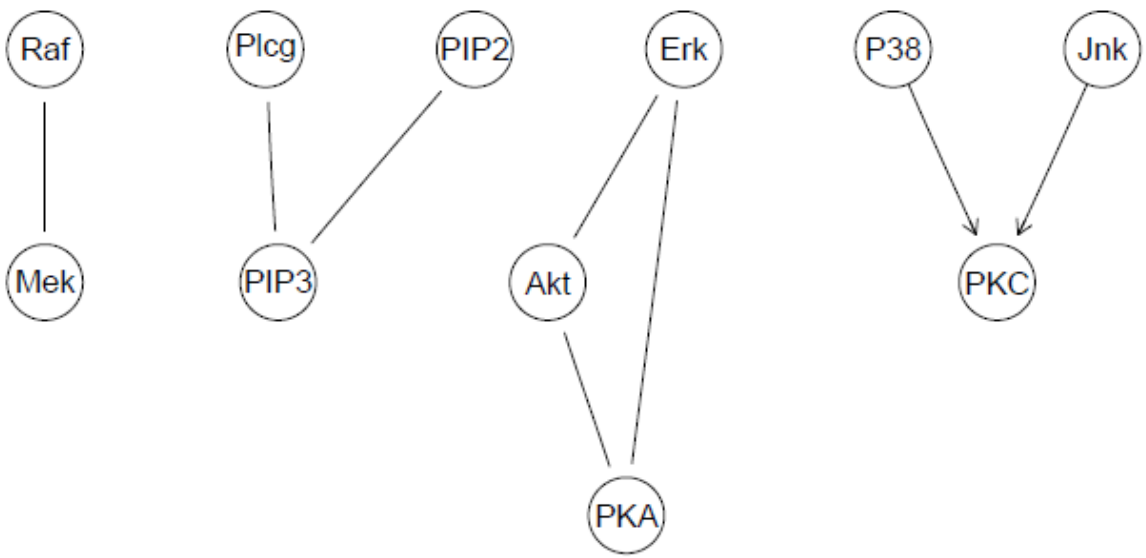
- 1800 data subject only to general stimolatory cues, so that the protein signalling paths are active;
- 600 data with with specific stimolatory/inhibitory cues for each of the following 4 proteins: Mek, PIP2, Akt, PKA;
- 1200 data with specific cues for PKA.

A First Try

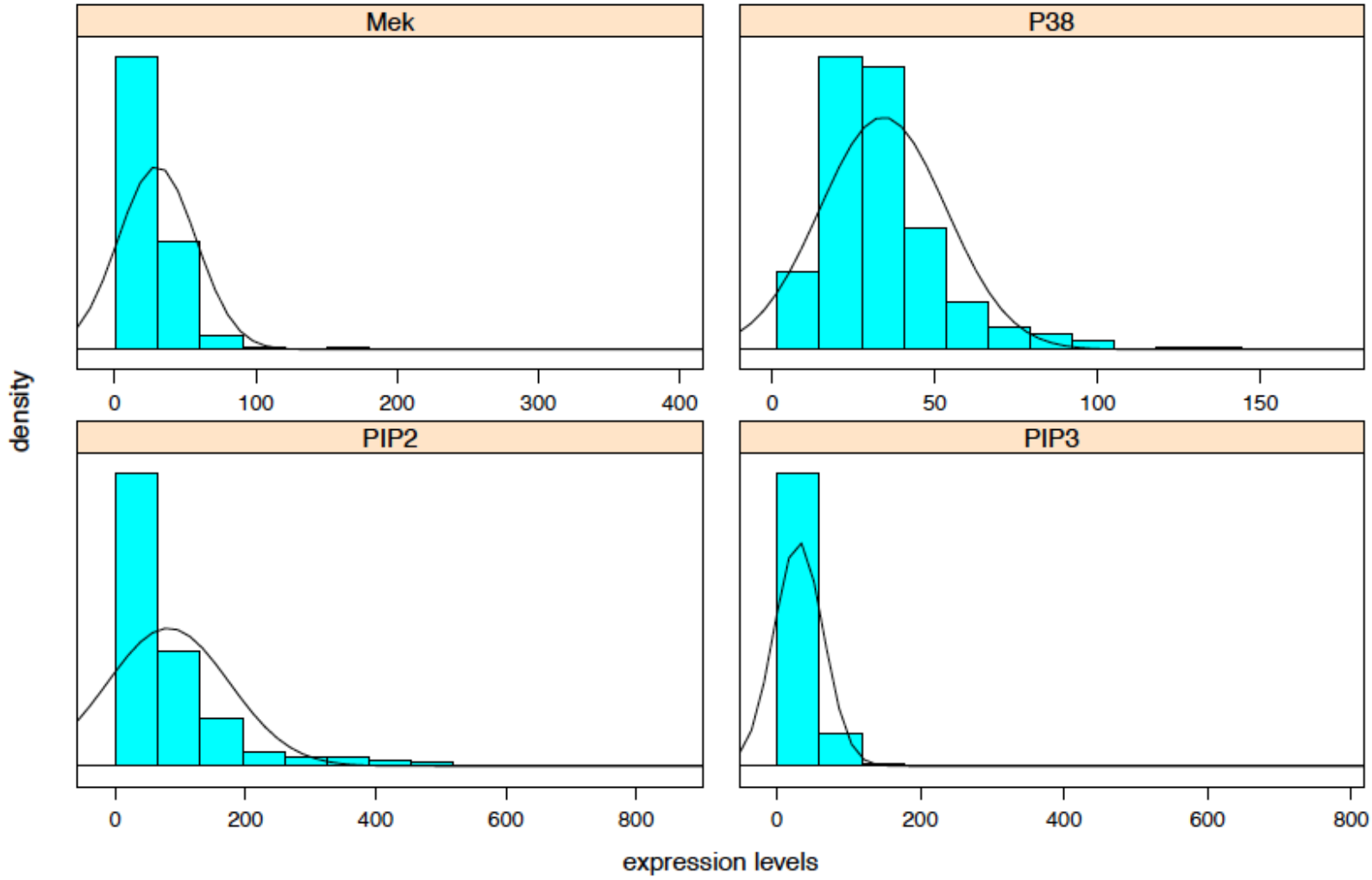
```
dag.hiton = si.hiton.pc(sachs, test = "cor", undirected = FALSE)
directed.edges(dag.hiton)
##      from  to
## [1,] "P38" "PKC"
## [2,] "Jnk" "PKC"
undirected.edges(dag.hiton)
##       from   to
##  [1,] "Raf"  "Mek"
##  [2,] "Mek"  "Raf"
##  [3,] "Plcg" "PIP3"
##  [4,] "PIP2" "PIP3"
##  [5,] "PIP3" "Plcg"
##  [6,] "PIP3" "PIP2"
##  [7,] "Erk"  "Akt"
##  [8,] "Erk"  "PKA"
##  [9,] "Akt"  "Erk"
## [10,] "Akt"  "PKA"
## [11,] "PKA"  "Erk"
## [12,] "PKA"  "Akt"
```

Compare with the Validated Model

```
sachs.modelstring =
paste("[PKC][PKA|PKC][Raf|PKC:PKA][Mek|PKC:PKA:Raf][Erk|Mek:PKA]",
"[Akt|Erk:PKA][P38|PKC:PKA][Jnk|PKC:PKA][Plcg][PIP3|Plcg]",
"[PIP2|Plcg:PIP3]")
dag.sachs = model2network(sachs.modelstring)
unlist(compare(dag.sachs, dag.hiton))
## tp fp fn
##  0  8 17
graphviz.plot(dag.hiton)
```

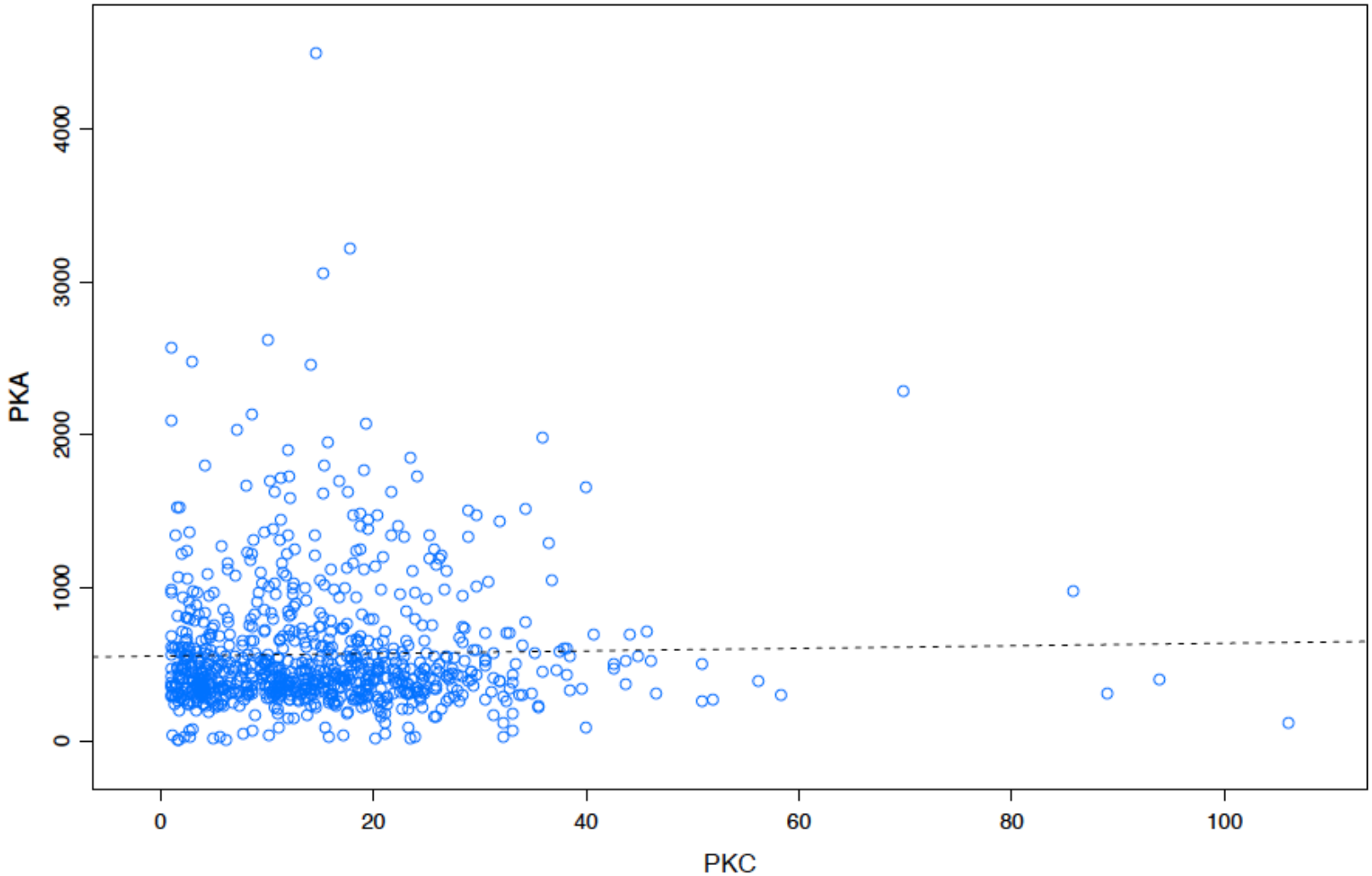


Are Variables Normally Distributed?



Variables are skewed and bounded below by zero, which makes them very different from a normal distribution. So, using a GBN may not be a good idea...

Are Dependencies Linear?



There is a PKC $\rightarrow$ PKA edge in the validated network, and PKC is the only parent of PKA. However, we cannot see any linear relationship...

What to Do Now?

Since GBNs are not appropriate, we must now consider alternatives:

- We explore monotone transformations like the $\log_1 0$ (tried, no improvements).
- We specify an appropriate conditional distribution for each variable using prior knowledge on the signalling pathways (which may or may not be available). However, the aim of the analysis was to use BNs as an automated probabilistic method to verify such information, not to build a BN with prior information and use it as an expert system.
- Discretise the data and to model them with a DBN, which can accommodate skewness and nonlinear relationships at the cost of potentially losing the ordering information. Since the variables in the BN represent concentration levels, Sachs et al. used three levels corresponding to low, average and high concentrations.

Hartemink's Information-Preserving Discretization

Input: a data set $\mathbf{X} = X_i, i = 1, \ldots, N$ where all X_i are continuous variables.

Output: a data set with N discrete variables, each with k_2 levels.

1. Discretise each variable independently using quantile Discretization and a large number k_1 of intervals, e.g., $k_1 = 50$ or even $k_1 = 100$.
2. Repeat the following steps until each variable has $k_2 \ll k_1$ intervals, iterating over each variable $X_i, i = 1, \ldots, N$ in turn:
   2.1 compute
   $$\mathrm{M}_{X_i} = \sum_{j \neq i} \mathrm{MI}(X_i, X_j);$$
   2.2 for each pair I of adjacent intervals of X_i, collapse them in a single interval, and with the resulting variable$X_i^*(I)$ compute
   $$\mathrm{M}_{X_i^*(I)} = \sum_{j \neq i} \mathrm{MI}(X_i^*(I), X_j);$$
   2.3 set $X_i = \arg\max_{X_i(I)} \mathrm{M}_{X_i^*(I)}$.

bnlearn: Discretising Data

An implementation of Hartemink's algorithm is provided in discretize(), which takes k_2 (breaks), k_1 (ibreaks) and the initial Discretization algorithm (idisc).

```
dsachs = discretize(sachs, method = "hartemink",
breaks = 3, ibreaks = 60, idisc = "quantile")
head(dsachs)
##            Raf        Mek       Plcg        PIP2        PIP3         Erk
## 1 (1.61,39.5]   (1,21.1]     (1,12] (1.11,34.9]  (50.9,764]    (1,15.3]
## 2 (1.61,39.5]   (1,21.1]  (12,23.1] (1.11,34.9]    (1,18.9] (15.3,29.4]
## 3 (39.5,62.6] (27.4,389]  (12,23.1] (1.11,34.9]    (1,18.9]    (1,15.3]
## 4  (62.6,552] (27.4,389] (23.1,167] (1.11,34.9]    (1,18.9]    (1,15.3]
## 5 (1.61,39.5]   (1,21.1]     (1,12] (1.11,34.9] (18.9,50.9] (15.3,29.4]
## 6 (1.61,39.5]   (1,21.1]  (12,23.1] (1.11,34.9]    (1,18.9]    (1,15.3]
##            Akt        PKA         PKC         P38         Jnk
## 1  (1.7,23.5] (1.95,547] (9.73,20.2]  (33.4,170]  (35.9,343]
## 2 (23.5,46.1] (1.95,547]    (1,9.73] (1.53,19.9]  (35.9,343]
## 3 (23.5,46.1] (1.95,547] (9.73,20.2] (19.9,33.4] (18.4,35.9]
## 4  (1.7,23.5] (1.95,547] (9.73,20.2] (19.9,33.4] (18.4,35.9]
## 5 (23.5,46.1] (1.95,547]    (1,9.73] (19.9,33.4]  (35.9,343]
## 6 (23.5,46.1]  (547,777] (9.73,20.2]  (33.4,170]  (35.9,343]
```

Structure Learning and Model Averaging

However, HITON is still not working...

```
dag.hiton = si.hiton.pc(dsachs, test = "x2", undirected = FALSE)
unlist(compare(dag.hiton, dag.sachs))
## tp fp fn
##  0 17 10
```

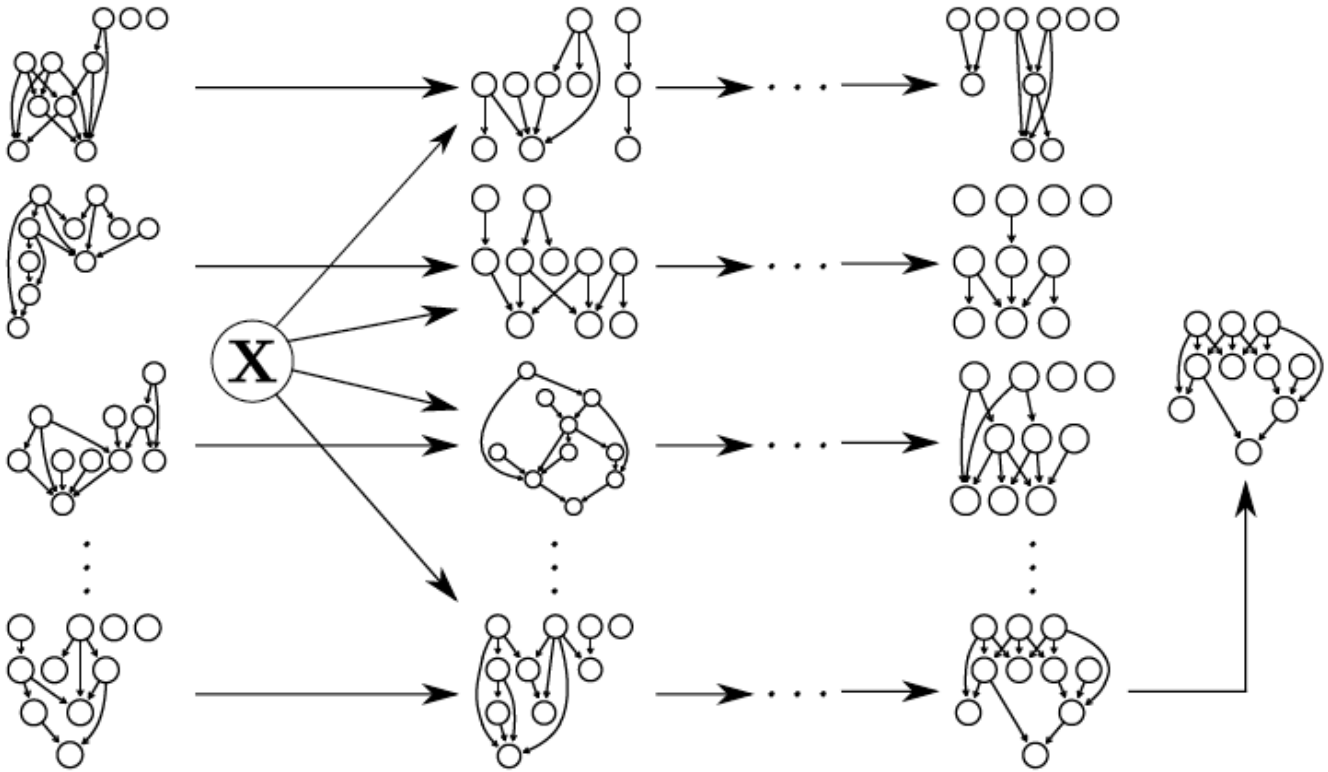
... so we switch to a score-based algorithm ...

```
dag.hc = hc(dsachs, score = "bde", iss = 10, undirected = FALSE)
unlist(compare(dag.hc, dag.sachs))
## tp fp fn
##  6 11  4
```

... and frequentist model averaging to remove spurious edges.

```
boot = boot.strength(dsachs, R = 500, algorithm = "hc",
algorithm.args = list(score = "bde", iss = 10))
head(boot[(boot$strength > 0.85) & (boot$direction >= 0.5), ], n = 3)
##    from   to strength direction
## 1   Raf  Mek    1.000     0.512
## 23 Plcg PIP2    0.998     0.510
## 24 Plcg PIP3    1.000     0.527
```

Learning Multiple DAGs from the Data



Seedgehing from different starting points increases our coverage of the space of the possible DAGs; the frequency with which an edge appears is a measure of the strength of the dependence.

Model Averaging from Multiple Seedgehes

While there is no function in **bnlearn** that does exactly this, we can combine random.graph() and sapply() to generate the random starting points and call hc() on each of them.

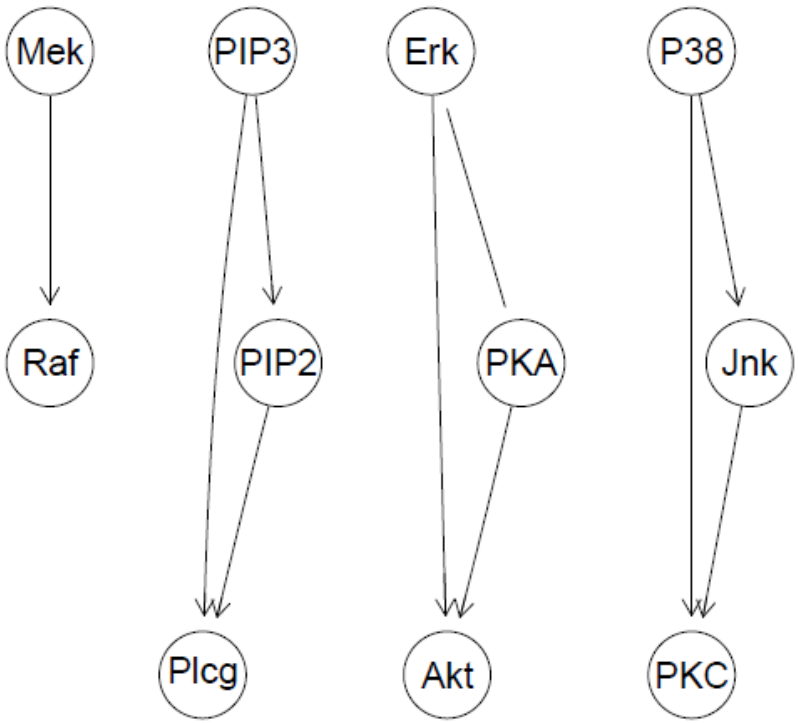
```
nodes = names(dsachs)
start = random.graph(nodes = nodes, method = "ic-dag",
num = 500, every = 50)
netlist = lapply(start,
function(net) {}
hc(dsachs, score = "bde", iss = 10, start = net)
}
)
```

Then we can take the resulting list and pass it to custom.strength() to compute edge strengths.

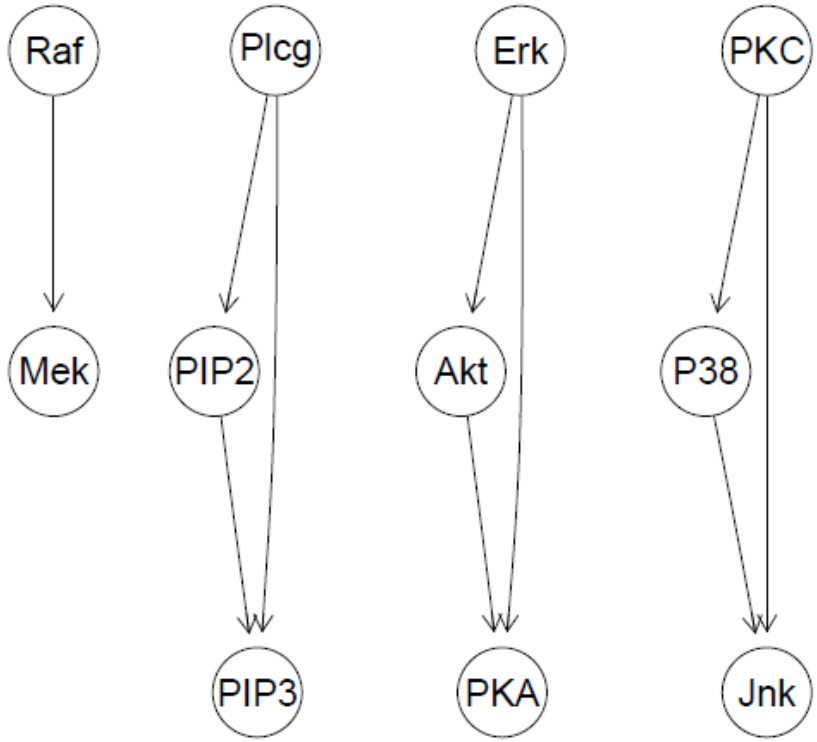
```
start = custom.strength(netlist, nodes = nodes)
```

Compare Both Approaches with the Validated Network

```
avg.start = averaged.network(start)
graphviz.plot(avg.start)
```



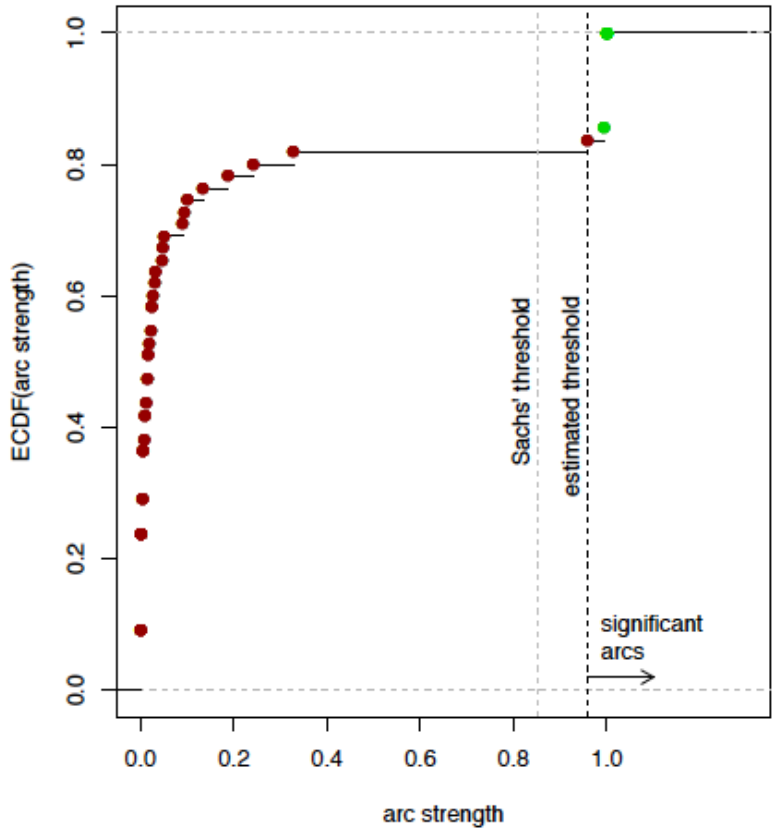

```
avg.boot = averaged.network(boot)
graphviz.plot(avg.boot)
```

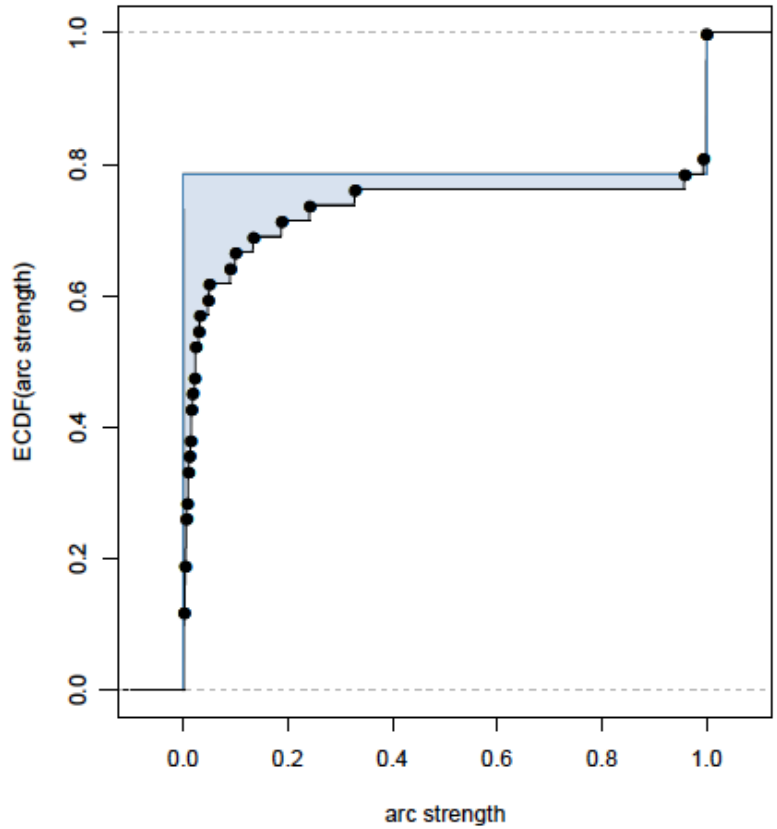



```
unlist(compare(avg.start, dag.sachs))
## tp fp fn
##  3 14  7
```

```
unlist(compare(avg.boot, dag.sachs))
## tp fp fn
##  6 11  4
```

Model Averaging for the Bootstrapped DAGs





edges with significant strength can be identified using a threshold estimated from the data by minimising the distance from the observed ECDF and the ideal, asymptotic one (the blue area in the right panel).

Taking the Interventions into Account

Both networks look nothing like the validated network, and in fact fall in the same equivalence class.

```
all.equal(cpdag(avg.boot), cpdag(avg.start))
## [1] TRUE
```

The only piece of information we have not taken into account yet are the stimulations and the inhibitions, that is, the interventions on the variables.

```
isachs = read.table("sachs.interventional.txt",
header = TRUE, colClasses = "factor")
```

With the discretised data, for each variable:

- an inhibition is an ideal intervention that sets the value to "low";
- a stimulations is an ideal intervention that sets the value to "high".

A Naive Approach with Whitelists

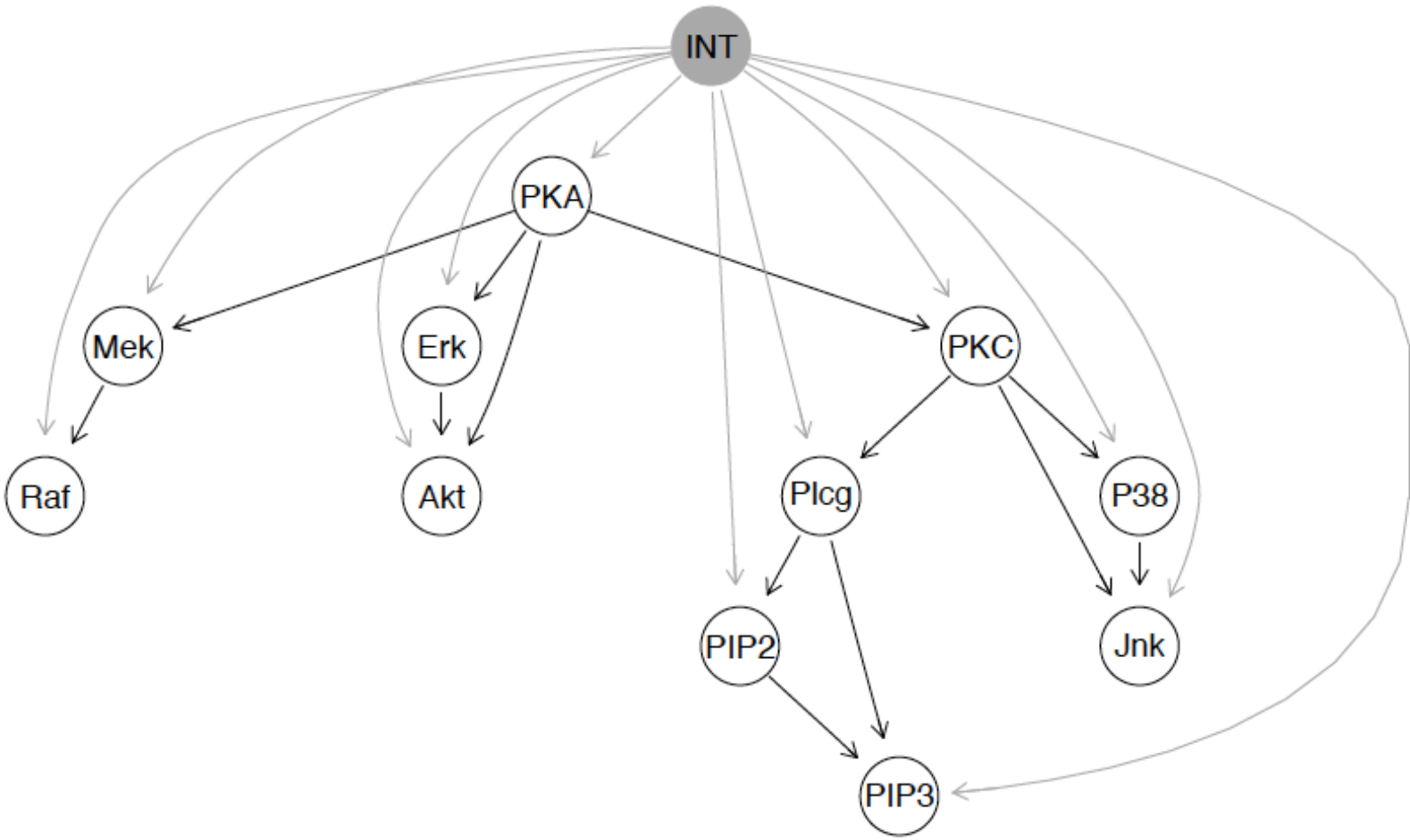
A naive approach to consider the intervention variable INT would be to include it as a node in the DAG and whitelist outgoing edges to all other variables to have different conditional probabilities depending on whether each observation is subject to an intervention.

```
wh = matrix(c(rep("INT", 11), names(isachs)[1:11]), ncol = 2)
dag.wh = tabu(isachs, whitelist = wh, score = "bde",
iss = 10, tabu = 50)
unlist(compare(subgraph(dag.wh, names(isachs)[1:11]), dag.sachs))
## tp fp fn
##  8  9  5
```

This works better than before, but we still do not get the validated network. Note that in this case we compare DAGs directly and not CPDAGs because the interventions break score equivalence by blocking the effect encoded by incoming edges for some combinations of nodes and observations.

A Naive Approach with Whitelists

```
graphviz.plot(dag.wh, highlight = list(nodes = "INT",
edges = outgoing.edges(dag.wh, "INT"), col = "darkgrey", fill = "darkgrey"))
```



Mixed Observational and Interventional Data

A more granular way of doing the same thing is to use the mixed observational and interventional data posterior score from Cooper & Yoo, which creates an implicit intervention binary node for each variable.

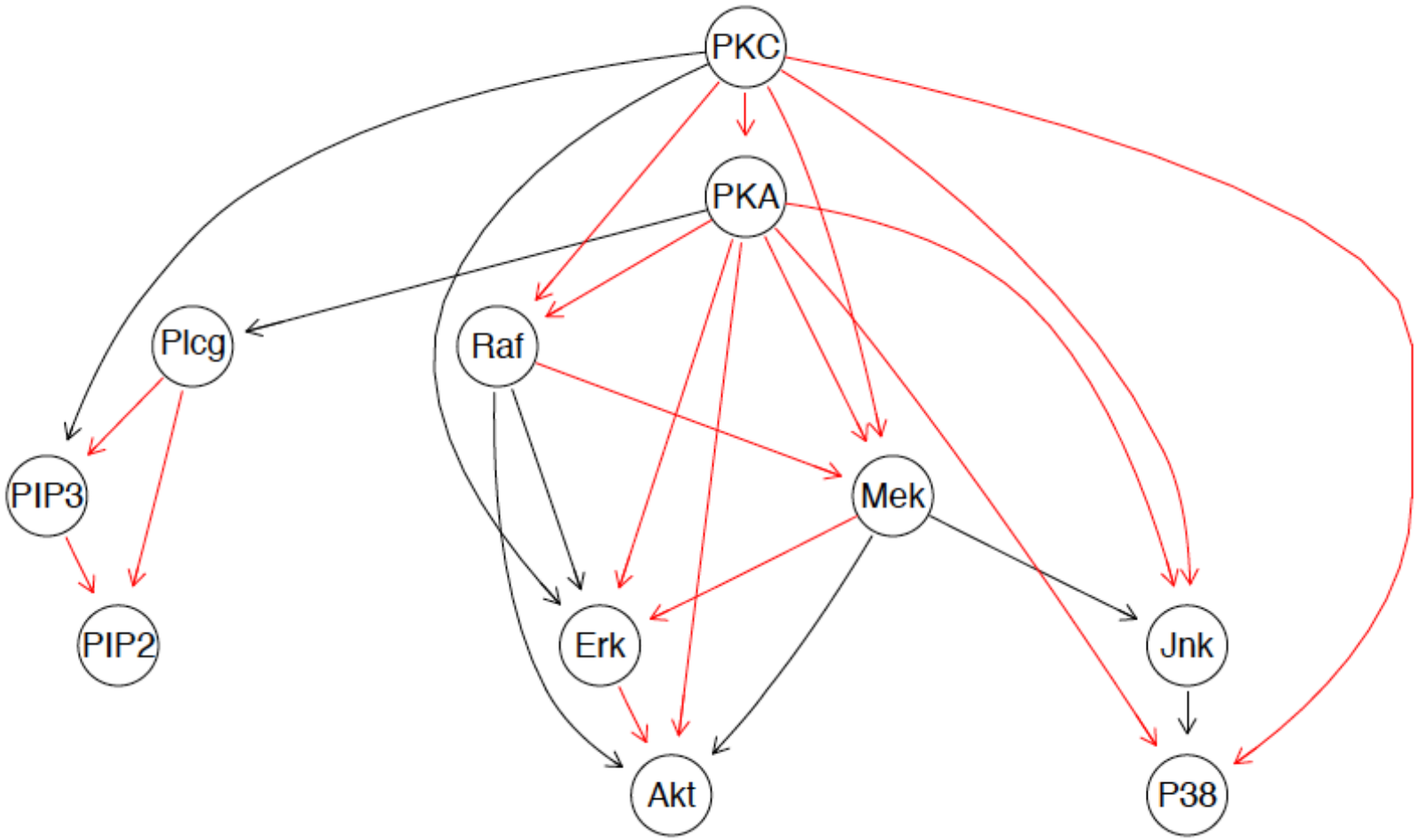
```
INT = sapply(1:11, function(x) which(isachs$INT == x) )
nodes = names(isachs)[1:11]
names(INT) = nodes
```

Then we perform model averaging of the resulting causal DAGs,with better results.

```
netlist = lapply(start, function(net) {
tabu(isachs[, 1:11], score = "mbde", exp = INT, iss = 1,
start = net, tabu = 50)
})
intscore = custom.strength(netlist, nodes = nodes, cpdag = FALSE)
dag.mbde = averaged.network(intscore)
unlist(compare(dag.sachs, dag.mbde))
## tp fp fn
## 17  8  0
```

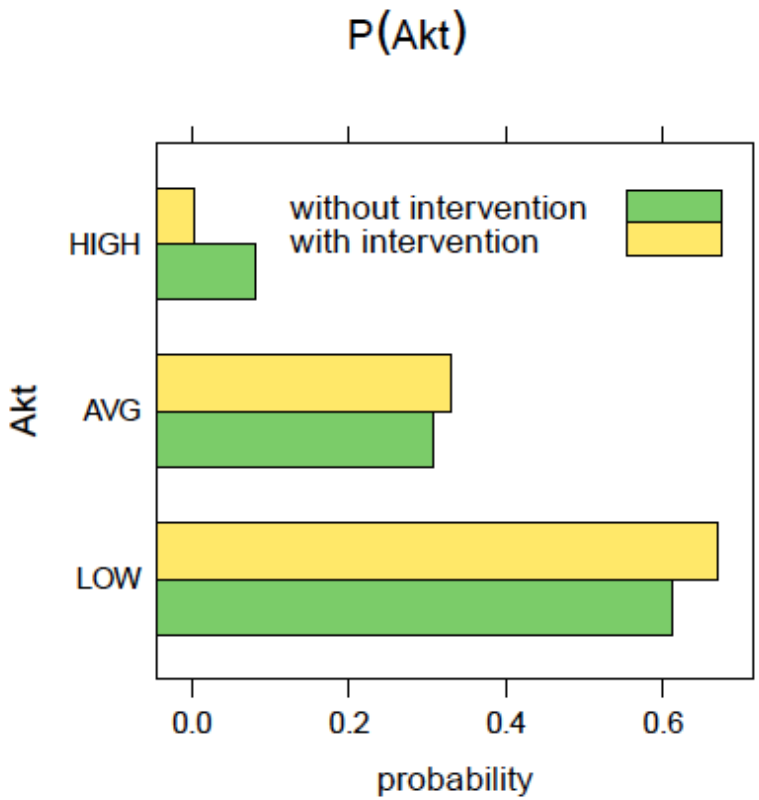
The Final DAG

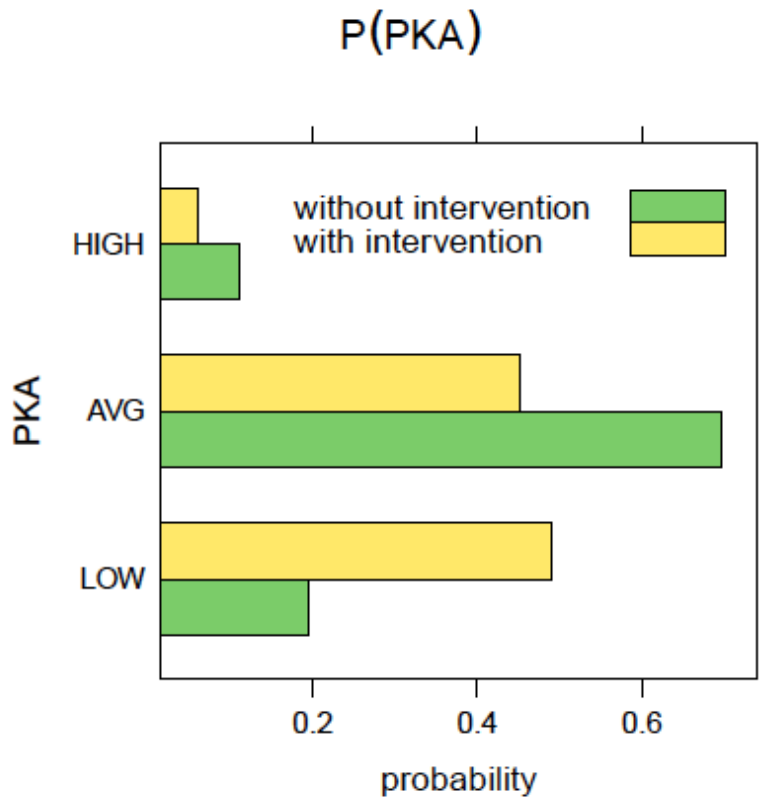
```
graphviz.plot(dag.mbde, highlight = list(edges = edges(dag.sachs)))
```



Using The Protein Network to Plan Experiments

This idea goes by the name of hypothesis generation: using a statistical model to decide which follow-up experiments to perform. BNs are especially easy to use for this because they automate the computation of arbitrary events.





Fitting the Parameters and Performing Queries

First, we need to learn the parameters of the BN given the DAG.

```
isachs = isachs[, 1:11]
for (i in names(isachs))
levels(isachs[, i]) = c("LOW", "AVG", "HIGH")
fitted = bn.fit(dag.sachs, isachs, method = "bayes")
```

Then we can proceed to perform queries using **gRain**, on the original BN

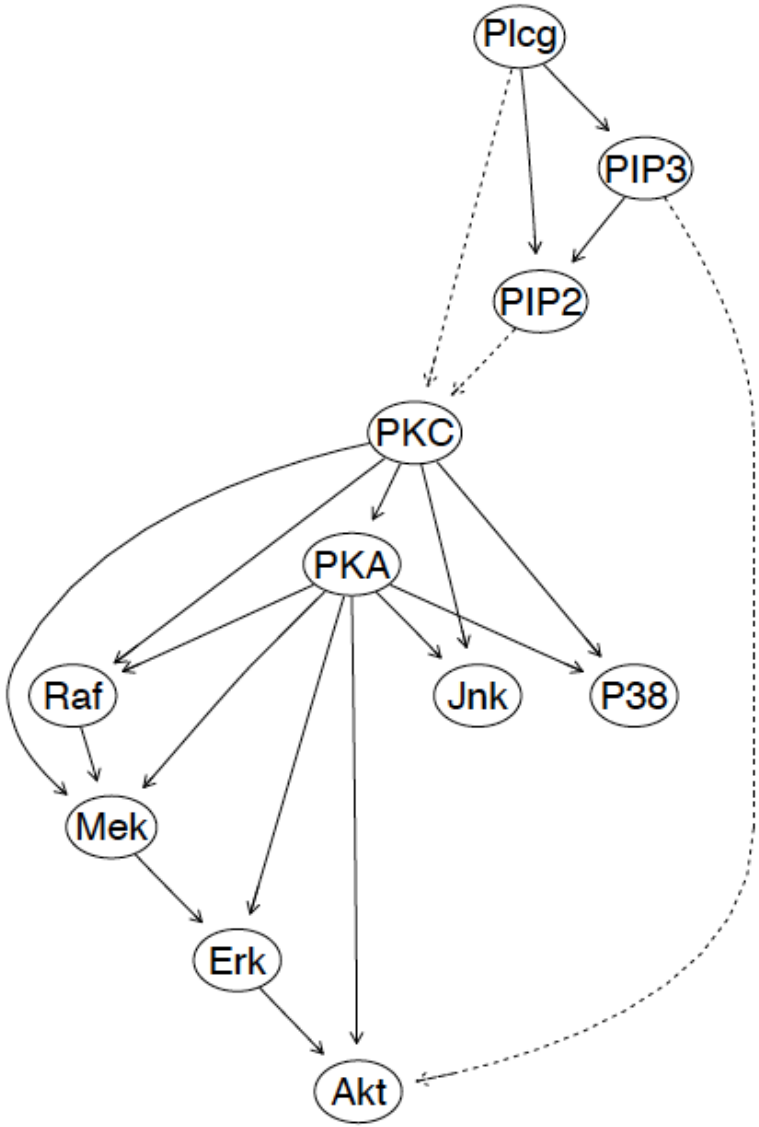
```
library(gRain)
jtree = compile(as.grain(fitted))
```

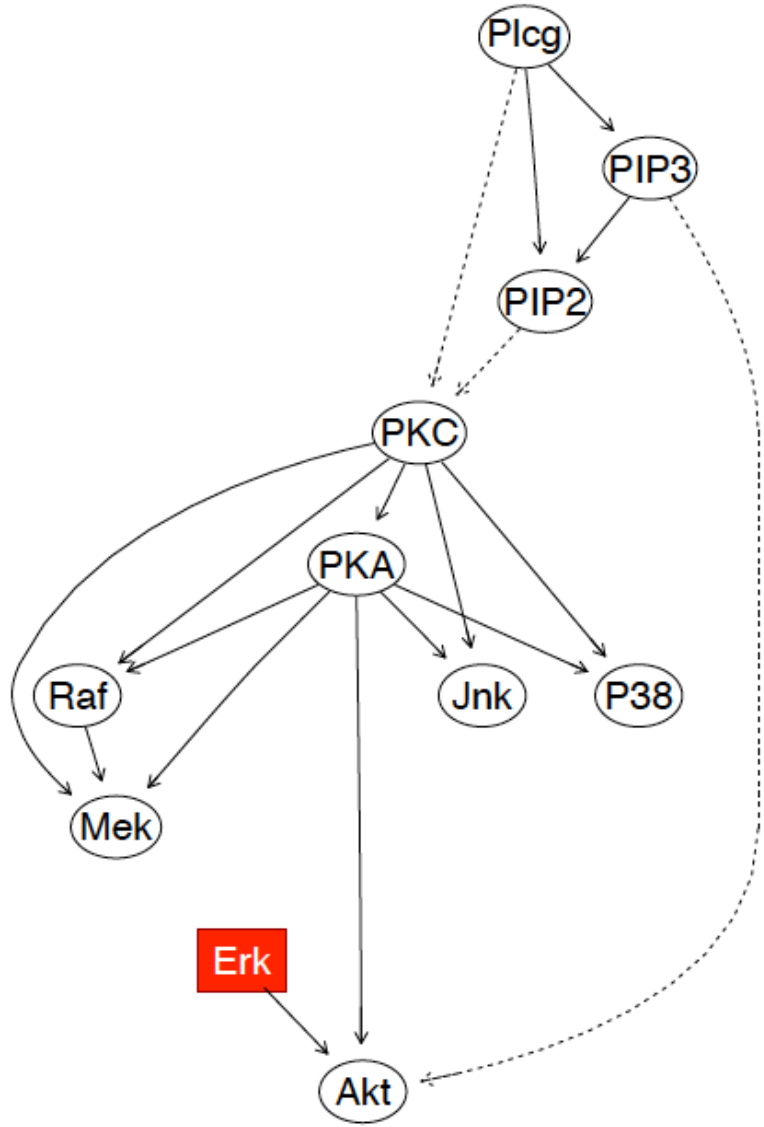
and on a mutilated BN in which we set Erk to LOW with an ideal intervention.

```
jlow = compile(as.grain(mutilated(fitted, evidence = list(Erk = "LOW"))))
```

In other words, we simulate a lab experiment in which we inhibit Erk (called a knock-out experiment). Much cheaper than actually doing it for real!

Interventions and Mutilated Graphs





Variables That are Downstream are Untouched

The marginal distribution of Akt changes depending on whether we take the evidence (intervention) into account or not.

```
querygrain(jtree, nodes = "Akt")$Akt
## Akt
##    LOW    AVG   HIGH
## 0.6089 0.3104 0.0807
querygrain(jlow, nodes = "Akt")$Akt
## Akt
##    LOW    AVG   HIGH
## 0.6671 0.3310 0.0019
```

The slight inhibition of Akt induced by the inhibition of Erk agrees with both the direction of the edge linking the two nodes and the additional experiments performed by Sachs et al. In causal terms, the fact that changes in Erk affect Akt supports the existence of a causal link from the former to the latter.

Causal Inference, Posterior Inference

If there is no causal link from the variable subject to intervention (Erk) to another variable (say PKA), the distribution of that variable will not be impacted by the intervention.

```
querygrain(jtree, nodes = "PKA")$PKA
## PKA
##   LO  W AVG  HIGH
## 0.194 0.696 0.110
querygrain(jlow, nodes = "PKA")$PKA
## PKA
##   LOW   AVG  HIGH
## 0.194 0.696 0.110
```

This is unlike posterior inference, because we do not remove Erk's parents in that case.

```
jlow = setEvidence(jtree, nodes = "Erk", states = "LOW")
querygrain(jlow, nodes = "PKA")$PKA
## PKA
##    LOW    AVG   HIGH
## 0.4891 0.4512 0.0597
```

Case Study: Plant Genetics

DNA data (e.g. SNP markers) is routinely used in statistical genetics to understand the genetic basis of human diseases, and to breed traits of commercial interest in plants and animals. Multiparent (MAGIC) populations are ideal for the latter. Here we consider a wheat population: 721 varieties, 16K genetic markers, 7 traits. (I ran the same analysis on a rice population, 1087 varieties, 4K markers, 10 traits, with similar results.) Phenotypic traits for plants typically include owering time,

height, yield, a number of disease scores. The goal of the analysis is to find key genetic markers controlling the traits; to identify any causal relationships between them; and to keep a good predictive accuracy.

Multiple Quantitative Trait Analysis Using Bayesian Networks

Medgeo Scutari, *et al.*, Genetics, **198**, 129-137 (2014); DOI: 10.1534/genetics.114.165704

Bayesian Networks in Genetics

If we have a set of traits and markers for each variety, all we need are the Markov blankets of the traits; most markers are discarded in the process.
Using common sense, we can make some assumptions:

- traits can depend on markers, but not vice versa;
- dependencies between traits should follow the order of the respective measurements (e.g. longitudinal traits, traits measured before and after harvest, etc.);
- dependencies in multiple kinds of genetic data (e.g. SNP + gene expression or SNPs + methylation) should follow the central dogma of molecular biology.

Assumptions on the direction of the dependencies allow to reduce Markov blankets learning to learning the parents and the children of each trait, which is a much simpler task.

Parametric Assumptions

In the spirit of classic additive genetics models, we use a Gaussian BN. Then the local distribution of each trait T_i is a linear regression model

$$\begin{aligned} T_i &= \boldsymbol{\mu}_{T_i} + \prod_{T_i} \boldsymbol{\beta}_{T_i} + \boldsymbol{\epsilon}_{T_i} \\ &= \boldsymbol{\mu}_{T_i} + \underbrace{T_j \beta_{T_j} + \ldots + T_k \beta_{T_k}}_{\text{traits}} + \underbrace{G_l \beta_{G_l} + \ldots + G_m \beta_{G_m}}_{\text{markers}} + \boldsymbol{\epsilon}_{T_i} \end{aligned}$$

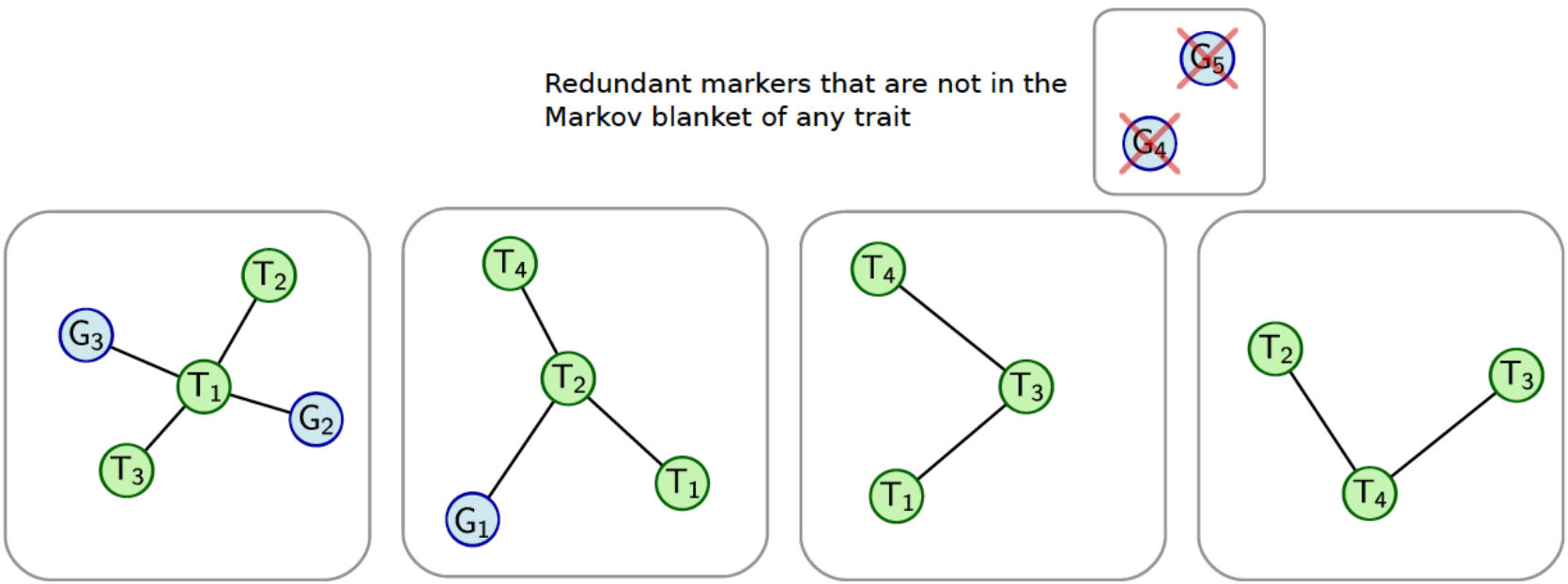
and the local distribution of each marker G_i is likewise

$$\begin{aligned} G_i &= \boldsymbol{\mu}_{G_i} + \prod_{G_i} \boldsymbol{\beta}_{G_i} + \boldsymbol{\epsilon}_{G_i} \\ &= \boldsymbol{\mu}_{G_i} + \underbrace{G_l \beta_{G_l} + \ldots + G_m \beta_{G_m}}_{\text{markers}} + \boldsymbol{\epsilon}_{G_i} \end{aligned}$$

in which the regressors ($\prod_{T_i}$ or $\prod_{G_i}$) are treated as fixed effects. $\prod_{T_i}$ can be interpreted as causal effects for the traits, $\prod_{G_i}$ as markers being in linkage disequilibrium with each other.

Learning the Bayesian Network (I)

1. Feature Selection.
   - 1.1 Independently learn the parents and the children of each trait with the SI-HITON-PC algorithm; children can only be other traits, parents are mostly markers, spouses can be either. Both are selected using the exact Student's t test for partial correlations.
   - 1.2 Drop all the markers that are not parents of any trait.



The Semi-Interleaved HITON-PC Algorithm

Input: each trait T_i in turn, other traits (T_j) and all markers (G_l), a significance threshold α.

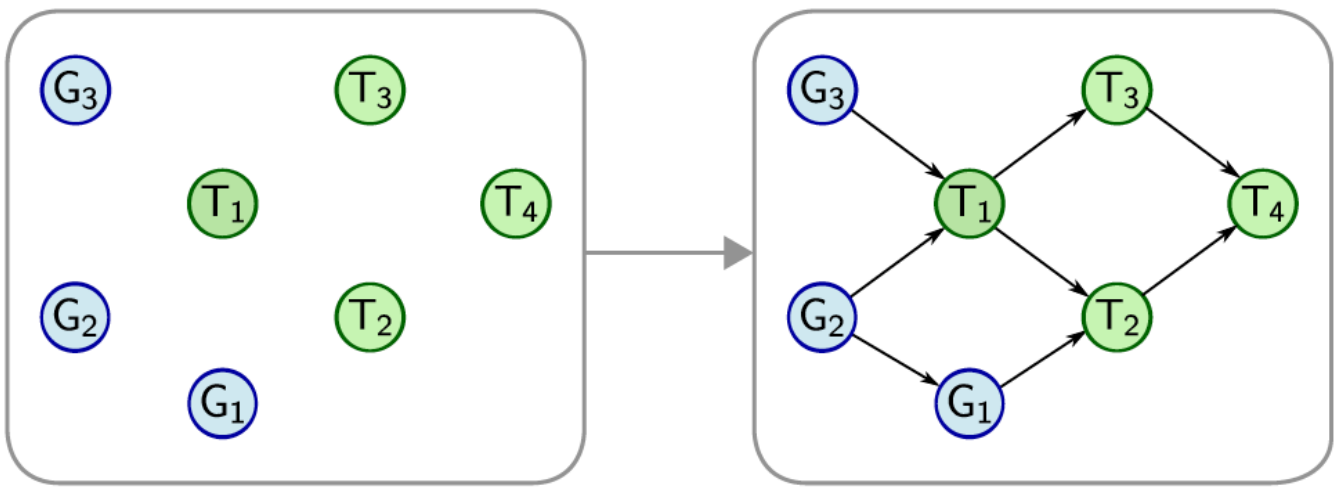
Output: the set CPC parents and children of T_i in the BN.

1. Perform a marginal independence test between T_i and each $T_j (T_i \perp\!\!\!\perp T_j)$ and $G_l (T_i \perp\!\!\!\perp G_l)$ in turn.
2. Discard all T_j and G_l whose p-values are greater than α.
3. Set $\mathbf{CPC} = \{\emptyset\}$.
4. For each the T_j and G_l in order of increasing p-value:
   - 4.1 Perform a conditional independence test between T_i and T_j / G_l conditional on all possible subsets $\mathbf{Z}$ of the current **CPC** ($T_i \perp\!\!\!\perp T_j \mid \mathbf{Z} \subseteq \mathbf{CPC}$ or $T_i \perp\!\!\!\perp G_l \mid \mathbf{Z} \subseteq \mathbf{CPC}$).
   - 4.2 If the p-value is smaller than α for all subsets then $\mathbf{CPC} = \mathbf{CPC} \cup \{T_j\}$ or $\mathbf{CPC} = \mathbf{CPC} \cup \{G_l\}$.

NOTE: the algorithm is defined for a generic independence test, you can plug in any test that is appropriate for the data.

Learning the Bayesian Network (II)

2. Structure Learning. Learn the structure of the network from the nodes selected in the previous step, setting the directions of the edges according to the assumptions above. The optimal structure can be identified with a suitable goodness-of-fit criterion such as BIC. This follows the spirit of other hybrid approaches (combining constraint-based and score-based learning) that have shown to be well-performing in the literature.

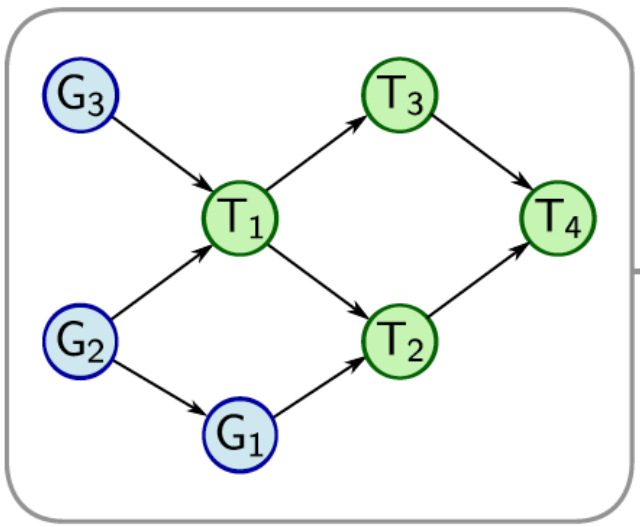


Empty network

Learned network

Learning the Bayesian Network (III)

3. Parameter Learning. LLearn the parameters: each local distribution is a linear regression and the global distribution is a hieredgehical linear model. Typically least squares works well because SI-HITON-PC selects sets of weakly correlated parents; ridge regression can be used otherwise.



Learned network

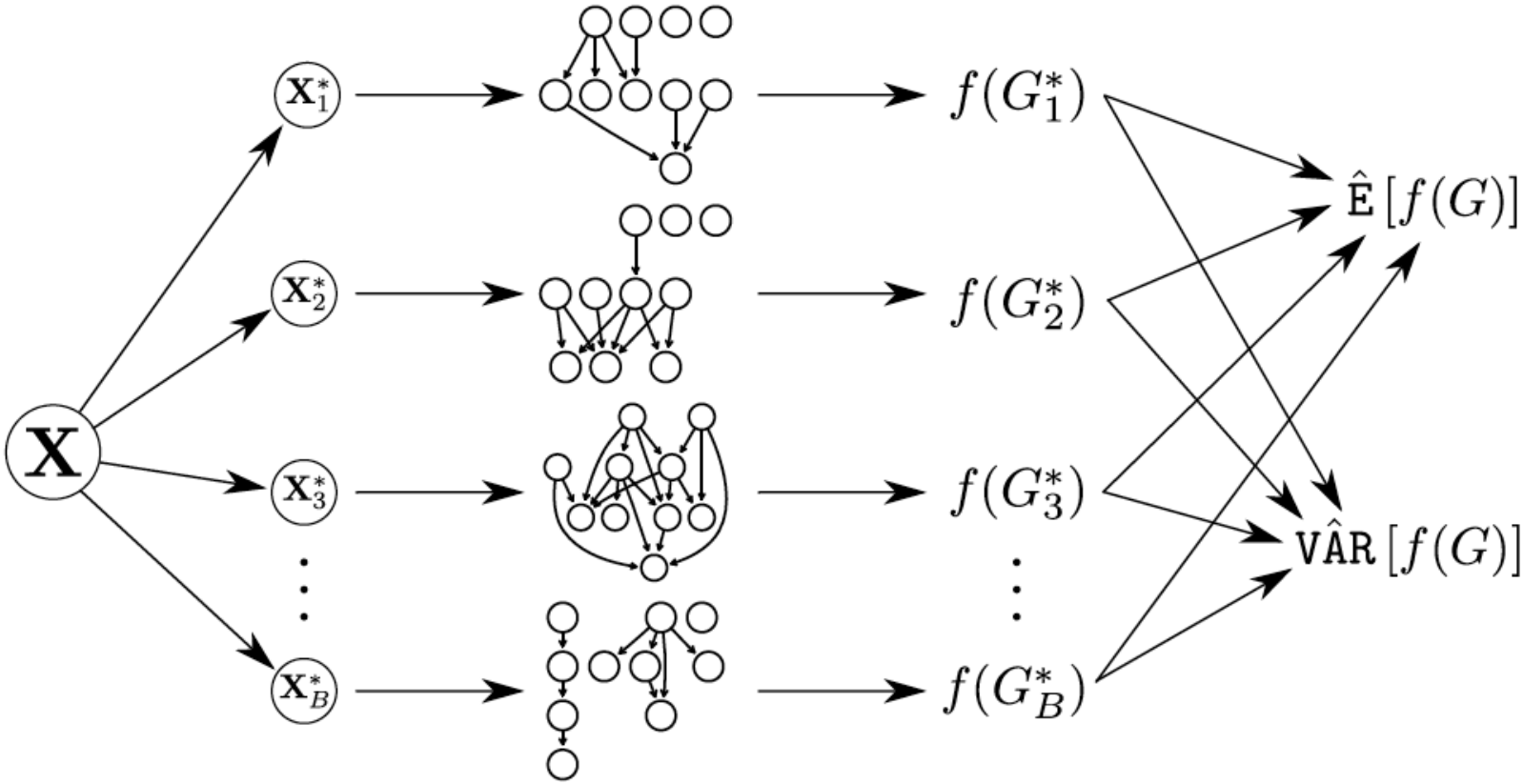
$$
\begin{aligned}
G_1 &= \mu_{G_1} + G_2\beta_{G_2} + \varepsilon_{G_1} \\
G_2 &= \mu_{G_2} + \varepsilon_{G_2} \\
G_3 &= \mu_{G_3} + \varepsilon_{G_3} \\
T_1 &= \mu_{T_1} + G_2\beta_{G_2} + G_3\beta_{G_3} + \varepsilon_{T_1} \\
T_2 &= \mu_{T_2} + T_1\beta_{T_1} + G_1\beta_{G_1} + \varepsilon_{T_2} \\
T_3 &= \mu_{T_3} + T_1\beta_{T_1} + \varepsilon_{T_3} \\
T_4 &= \mu_{T_4} + T_2\beta_{T_2} + T_3\beta_{T_3} + \varepsilon_{T_4}
\end{aligned}
$$

Local distributions

Learning The Structure

```
fit.the.model = function(data, traits, genes, alpha) {
qtls = vector(length(traits), mode = "list")
names(qtls) = traits
# find the parents of each trait among the genes.
for (q in seq_along(qtls)) {
# BLUP away the family structure.
m = lmer(as.formula(paste(traits[q], "~ (1|FUNNEL:PLANT)")), data = data)
data[!is.na(data[, traits[q]]), traits[q]] = data[, traits[q]] -
ranef(m)[[1]][paste(data$FUNNEL, data$PLANT, sep = ":"), 1]
# find out the parents.
qtls[[q]] = learn.nbr(data[, c(traits, genes)], node = traits[q],
method = "si.hiton.pc", test = "cor", alpha = alpha)
}#FOR
# yield has no children, and genes cannot depend on traits.
nodes = unique(c(traits, unlist(qtls)))
blacklist = tiers2blacklist(list(nodes[nodes %in% genes],
c("FT", "HT"),
traits[!(traits %in% c("YLD", "FT", "HT"))], "YLD"))
# build the overall network.
hc(data[, nodes], blacklist = blacklist)
}#FIT.THE.MODEL
```

Model Averaging and Assessing Predictive Accuracy



We perform all the above in 10 runs of 10-fold cross-validation to

- assess predictive accuracy with e.g. predictive correlation;
- obtain a set of DAGs to produce an averaged, de-noised consensus DAG with model averaging.

Performing Cross-Validation (Single Fold)

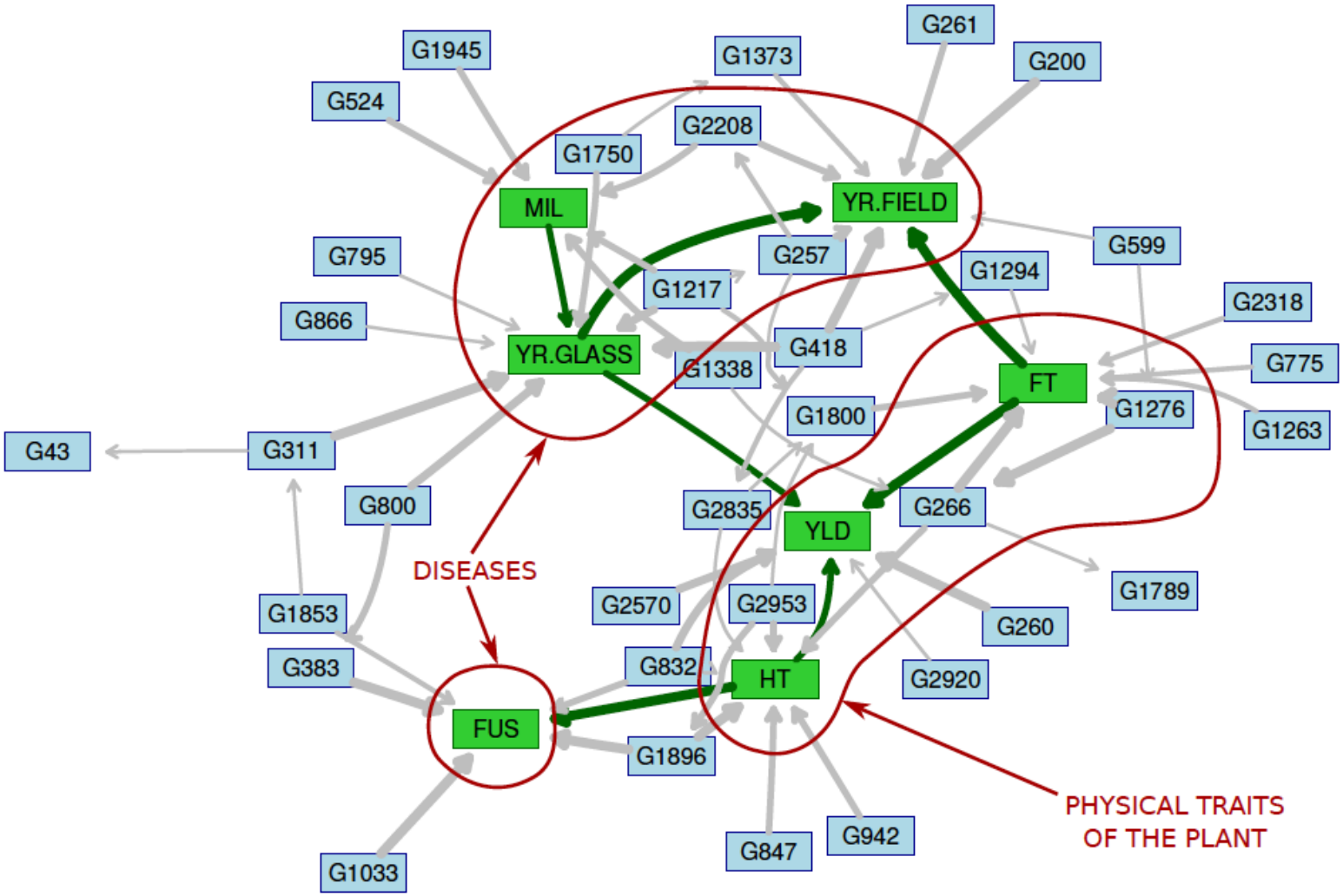
```
predicted = parLapply(kcv, cl = cluster, function(test) {
# create matrices to store the predicted values.
pred = matrix(0, nrow = length(test), ncol = length(traits))
post = matrix(0, nrow = length(test), ncol = length(traits))
colnames(pred) = colnames(post) = traits
# split training and test.
dtraining = data[-test, ]
dtest = data[test, ]
# fit the model on the training data.
model = fit.the.model(dtraining, traits, genes, alpha = alpha)
fitted = bn.fit(model, dtraining[, nodes(model)])
# subset the test data.
dtest = dtest[, nodes(model)]
# predict each trait in turn, given all the parents.
for (t in traits)
pred[, t] = predict(fitted, node = t, data = dtest[, nodes(model)])
# predict each trait in turn, given all the genes.
for (t in traits)
post[, t] = predict(fitted, node = t,
data = dtest[, names(dtest) %in% genes, drop = FALSE],
method = "bayes-lw", n = 1000)
return(list(model = fitted, pred = pred, post = post))
})
```

Averaging the Models from Cross-Validation

```
average.the.model = function(batch, data) {
# gather all the edge lists.
edgelist = list()
for (i in seq_along(batch)) {
# extract the models.
run = batch[[i]]$models
for (j in seq_along(run))
edgelist[[length(edgelist) + 1]] = edges(run[[j]])
}#FOR
# compute the edge strengths.
nodes = unique(unlist(edgelist))
str = custom.strength(edgelist, nodes = nodes)
# estimate the threshold and average the networks.
averaged = averaged.network(str)
# subset the network to remove isolated nodes.
relnodes = nodes(averaged)[sapply(nodes, degree, object = averaged) > 0]
averaged2 = subgraph(averaged, relnodes)
str2 = str[(str$from %in% relnodes) & (str$to %in% relnodes), ]
# save the fitted averaged network.
fitted = bn.fit(averaged2, data[, nodes(averaged2)])

return(list(model = averaged2, strength = str2, fitted = fitted))
}#AVERAGE.THE.MODEL
```

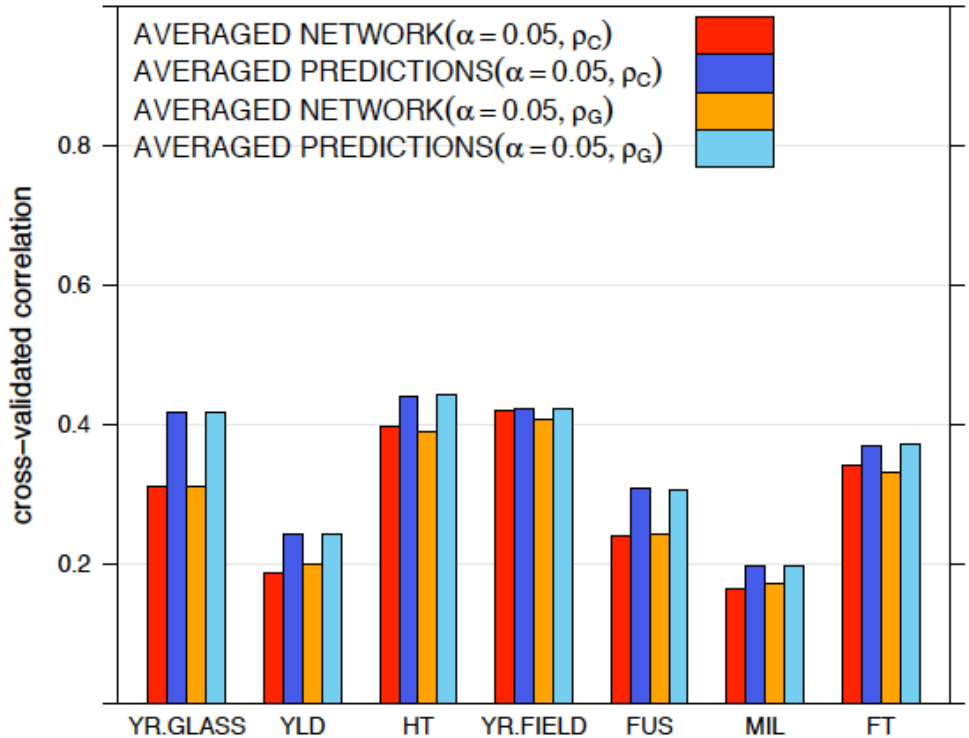
The Averaged Bayesian Network (44 nodes, 66 edges)



Predicting Traits for New Individuals

We can predict the traits:

1. from the averaged consensus network;
2. from each of the 10 $\times$ 10 networks we learn during cross-validation, and average the predictions for each new individual and trait.



Option 2. almost always provides better accuracy than option 1.; 10 $\times$ 10 networks capture more information, and we have to learn them anyway.

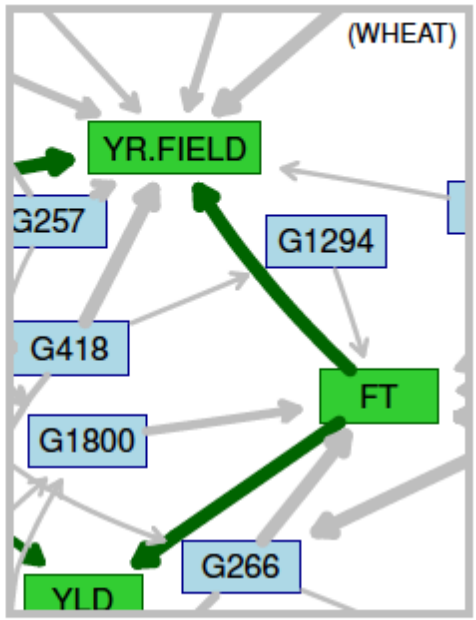
So: averaged network for interpretation, ensemble of networks for predictions.

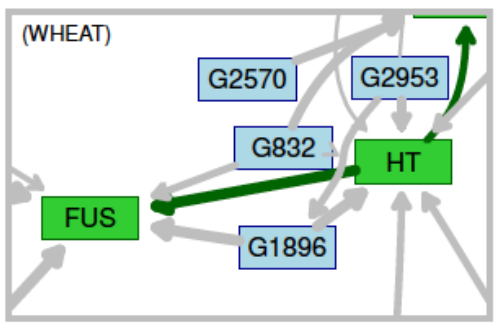
Causal Relationships Between Traits

One of the key properties of BNs is their ability to capture the direction of the causal relationships in the absence of latent confounders (the experimental design behind the data collection should take care of a number of them). Markers are causal for traits, but we do not know how traits in uence each other, and we want to learn that from the data.

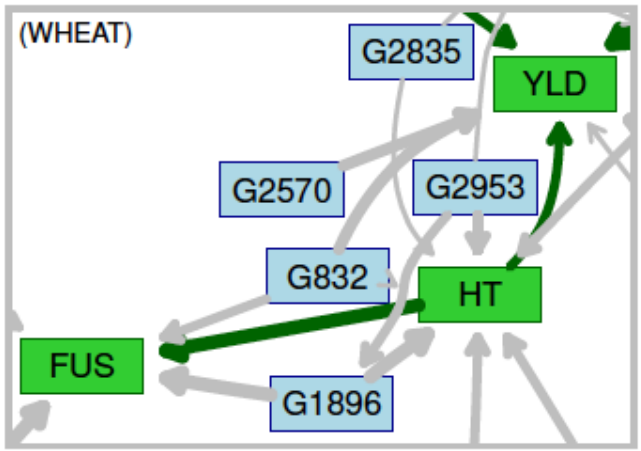
It works out because each trait will have at least one incoming edge from the markers, say $G_l \rightarrow T_j$, and then $(G_l \rightarrow) T_j \leftarrow T_k$ and $(G_l \rightarrow) T_j \rightarrow Tk$ are not probabilistically equivalent. So the network can

- suggest the direction of novel relationships;
- confirm the direction of known relationships, troubleshooting the experimental design and data collection.





Spotting Confounding Effects

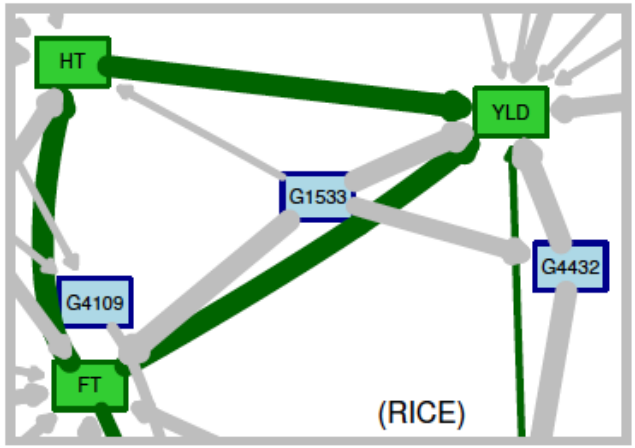


Traits can interact in complex ways that may not be obvious when they are studied individually, but that can be explained by considering neighbouring variables in the network. An example: in the WHEAT data, the difference in the mean YLD between the bottom and top quartiles of the FUS disease scores is +0.08.

So apparently FUS is associated with increased YLD! What we are actually measuring is the confounding effect of HT (FUS $\leftarrow$ HT $\rightarrow$ YLD); conditional on each quartile of HT, FUS has a negative effect on YLD ranging from -0.04 to -0.06. This is reassuring since it is known that susceptibility to fusarium is positively related to HT, which in turn affects YLD.

Disentangling Pleiotropic Effects (I)

When a marker is shown to be associated to multiple traits in a GWAS, we should separate its direct and indirect effects on each of the traits. (Especially if the traits themselves are linked!) Take for example G1533 in the RICE data set: it is putative causal for YLD, HT and FT.



- The difference in mean between the two homozygotes is +4.5cm in HT, +2.28 weeks in FT and +0.28 t/ha in YLD.
- Controlling for YLD and FT, the difference for HT halves (+2.1cm);
- Controlling for YLD and HT, the difference for FT is about the same (+2.3 weeks);
- Controlling for HT and FT the difference for YLD halves (+0.16 t/ha).

So, the model suggests the marker is causal for FT and that the effect on the other traits is partly indirect. This agrees from the p-values from an independent GWAS study (FT: 5.87e-28 $<$ YLD: 4.18e-10, HT:1e-11).

Disentangling Pleiotropic Effects (II)

```
control.ht = mutilated(bn.net(fitted), list("YLD" = 0, "FT" = 0))
control.ht = bn.fit(control.ht, indica[, nodes(control.ht)])
sim.aa = cpdist(control.ht, node = c("HT"), evidence = list(G1533 = 0),
method = "lw")
sim.AA = cpdist(control.ht, node = c("HT"), evidence = list(G1533 = 2),
method = "lw")
colMeans(sim.AA) - colMeans(sim.aa)

control.ft = mutilated(bn.net(fitted), list("YLD" = 0, "HT" = 0))
control.ft = bn.fit(control.ft, indica[, nodes(control.ft)])
sim.aa = cpdist(control.ft, node = c("FT"), evidence = list(G1533 = 0),
method = "lw")
sim.AA = cpdist(control.ft, node = c("FT"), evidence = list(G1533 = 2),
method = "lw")
colMeans(sim.AA) - colMeans(sim.aa)

control.yld = mutilated(bn.net(fitted), list("FT" = 0, "HT" = 0))
control.yld = bn.fit(control.yld, indica[, nodes(control.yld)])
sim.aa = cpdist(control.yld, node = c("YLD"), evidence = list(G1533 = 0),
method = "lw")
sim.AA = cpdist(control.yld, node = c("YLD"), evidence = list(G1533 = 2),
method = "lw")
colMeans(sim.AA) - colMeans(sim.aa)
```

Case Study:

INTED

Learning a Bayesian Structure to Model Attitudes Towards Business Creation at University

Ruiz-Ruano Gedgeia *et al.*, INTED, 5242-5249 (2014).

The main objective of this paper is to test a theoretical model of business creation based on the attitudes perspective:

> *The intention to create a new business would depend on attitudinal evaluation, if someone considers that creating a new business is a positive thing, he or she will be more prone to carry out the target behaviour. Additionally, intentions also depend on normative beliefs. That is to say, intentions depend on the perceived social pressure related with a particular behaviour.*

The data contains the answers to an electronic questionnaire from 1542 university professors from Andalusian universities (unfortunately with a response rate of $\approx 10\%$).

The Questionnaire

The questionnaire contained six sections:

1. demographic data;
2. questions directly related with entrepreneurship phenomena;
3. environment attitudes;
4. obstacles and facilitators;
5. an attitudinal scale;
6. comments and details.

To measure different aspect related with the entrepreneurial attitude we used scales about perceived obstacles, perceived facilitators, self-efficacy, locus of control, attitude towards business creation and normative beliefs. Scores in all scales were individually recoded into three levels of response (low, medium and high) using k-means.

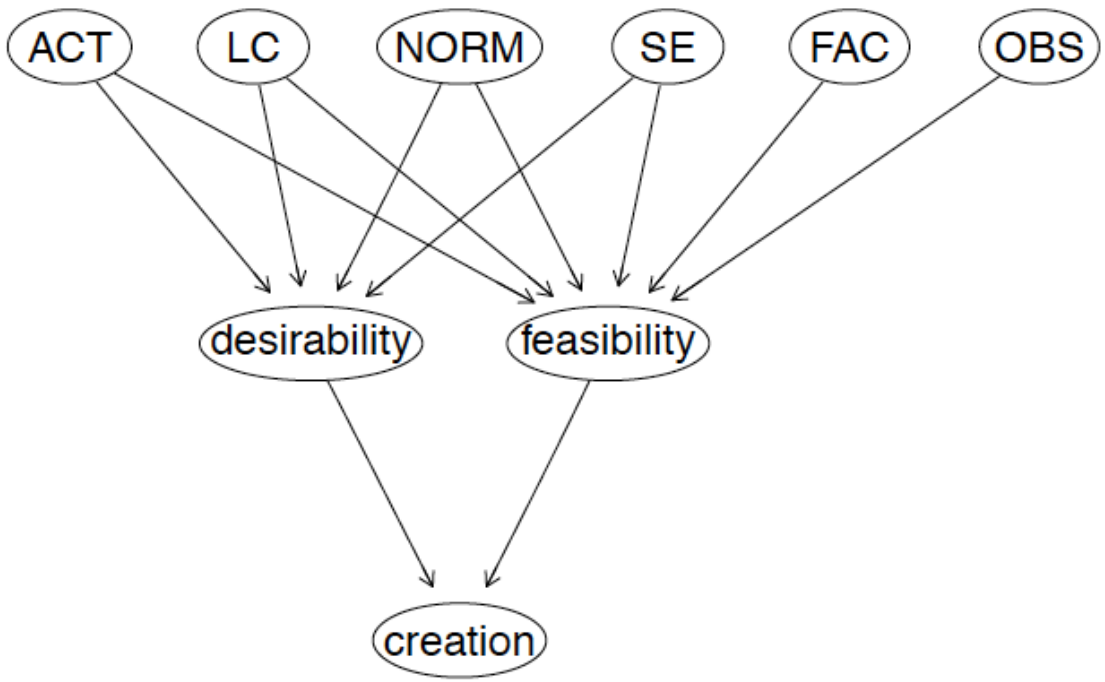
The Derived Scales

- perceived obstacles (OBS, out of 17): "Having to work too many hours", "Lack of experience", "Ignorance of activity sector", etc.
- perceived facilitators (FAC, out of 11): "Have perceived a need in the market", "The detection of a business opportunity" or "The availability of personal assets to invest", etc.
- self-efficacy (SE, 9 Likert items), the perceived difficulty to actually carry out a specific behaviour: "Working under continuous stress, pressure and conflict", "To form alliances or partnerships with other companies", etc.
- locus of control (LC, 3 Likert items): "If you want, you can easily be an entrepreneur and starting your own business", etc.
- attitude towards business creation (ACT, 6 Likert items): "To what extent do you believe that these elements are related with the creation of a new company?", "To what extent do you like assume it?", etc.
- normative beliefs (NORM, 4 Likert items): "Please, think in your family, closest friends and social environment and indicate the degree to which they are favourable to the idea that you create a company", etc.

A Prognostic Model

From the literature we assumed this prognostic BN for the data:

```
progn = model2network(
paste0("[creation|desirability:feasibility][desirability|LC:SE:ACT:NORM]",
"[feasibility|LC:SE:ACT:NORM:FAC:OBS][LC][FAC][OBS][SE][ACT][NORM]"))
graphviz.plot(progn, shape = "ellipse")
```



Running Out of Samples

The problems start when we try to learn the parameters of the BN from the data:

```
summary(inted)
## creation desirability                feasibility      LC
## Yes: 480 Yes:882      Very.little.feasible:378  High  :373
## No :1062 No :660      A.little.feasible   :672  Low   :544
##                       Feasible            :444  Medium:625
##                       A.lot.feasible      : 48
##     FAC         OBS          SE         ACT         NORM
## Low   :561 Low   :312 Medium:412 Medium:724 High  :318
## High  :259 Medium:793 Low   :774 Low   :226 Medium:452
## Medium:722 High  :437 High  :356 High  :592 Low   :772
##
```

A cursory examination suggests that the sample size is too small.

```
nparams(progn, inted)
## [1] 2288
nrow(inted)
## [1] 1542
```

Small n, Large p

If we learn the parameters with the classic maximum likelihood estimator, $\approx$ 40% of the CPT is missing values and another $\approx$ 40% is 0-1 distributions, which clearly is not ideal.

```
fitted.progn = bn.fit(progn, inted)
ldist = coef(fitted.progn$feasibility)
length(which(is.na(ldist))) / length(ldist)
## [1] 0.396
length(which(ldist %in% c(0, 1))) / length(ldist)
## [1] 0.397
```

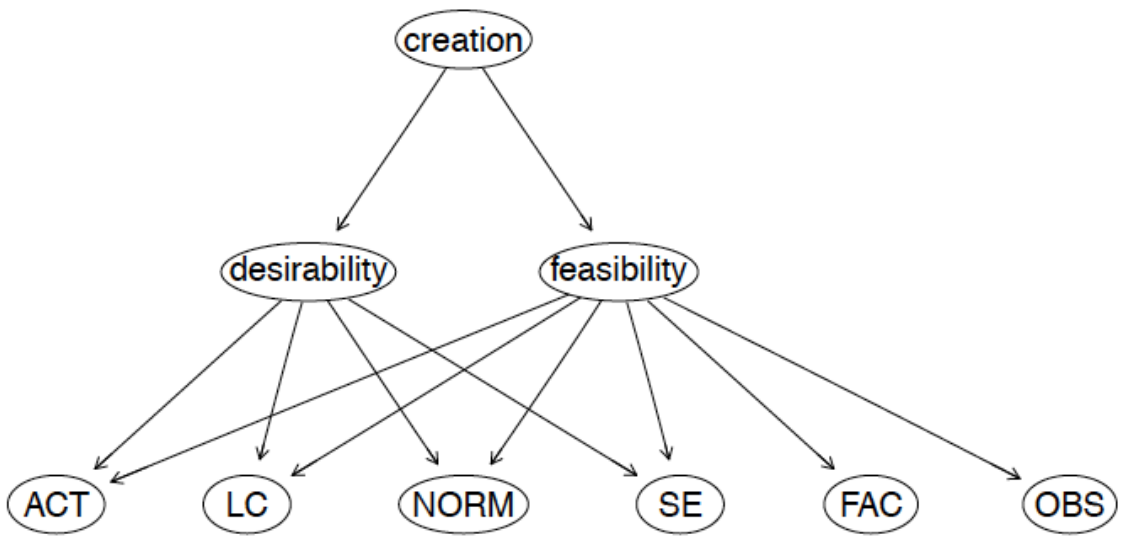
While we can paper over the problem by using posterior estimates...

```
fitted.progn = bn.fit(progn, inted, method = "bayes", iss = 1)
ldist = coef(fitted.progn$feasibility)
length(which(is.na(ldist))) / length(ldist)
## [1] 0
length(which(ldist %in% c(0, 1))) / length(ldist)
## [1] 0
```

... the BN would still lack statistical power.

A Diagonostic Model

```
diagn = model2network(
paste("[creation][desirability|creation][feasibility|creation]",
"[LC|desirability:feasibility][FAC|feasibility][OBS|feasibility]",
"[SE|desirability:feasibility][ACT|desirability:feasibility]",
"[NORM|desirability:feasibility]", sep = ""))
nparams(diagn, inted)
## [1] 89
graphviz.plot(diagn, shape = "ellipse")
```



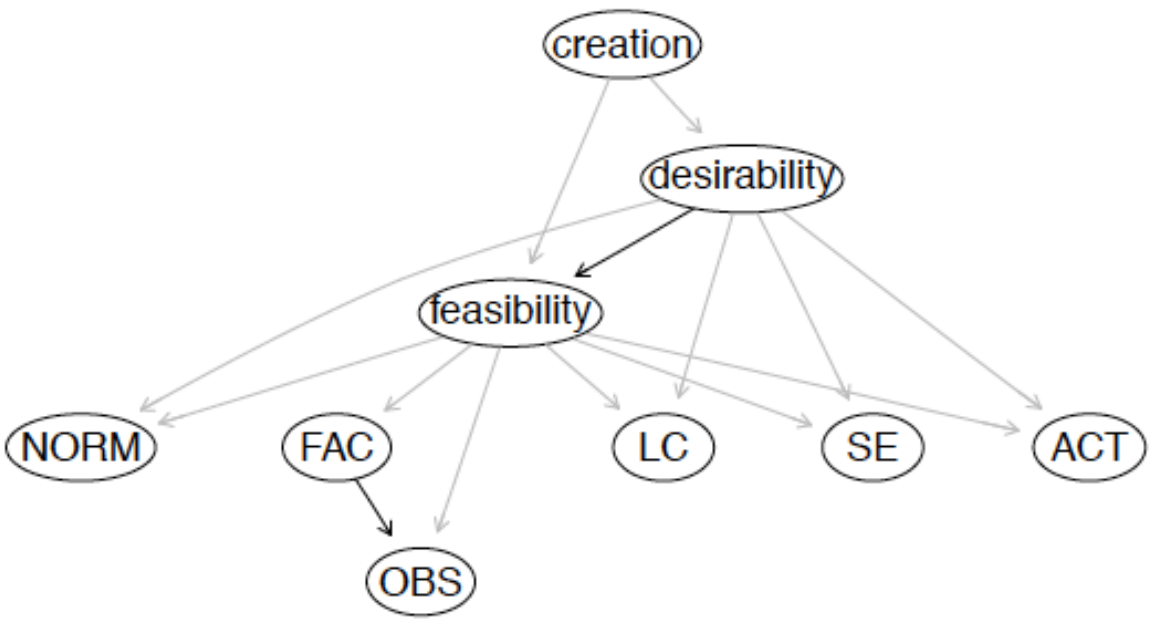
Developing the Model

The diagnostic BN has far fewer parameters, and we can estimate them with reasonable accuracy from the data.

```
fitted.diagn = bn.fit(diagn, inted)
```

Do the data support the any further edges we may have overlooked?

```
diagn2 = tabu(inted, whitelist = edges(diagn))
graphviz.plot(diagn2, highlight = list(edges = edges(diagn), col = "grey"),
shape = "ellipse")
```



Job Creation, Goodness of Fit

The three models we are considering fit the data equally well; the classi cation error for creation is about the same (≈ 0.274).

```
pred = predict(fitted.diagn, node = "creation", data = inted,
method = "bayes-lw")
ct = table(inted$creation, pred)
1 - sum(diag(ct)) / sum(ct)
## [1] 0.274
pred = predict(bn.fit(diagn2, inted), node = "creation", data = inted,
method = "bayes-lw")
ct = table(inted$creation, pred)
1 - sum(diag(ct)) / sum(ct)
## [1] 0.275
pred = predict(fitted.progn, node = "creation", data = inted,
method = "bayes-lw")
ct = table(inted$creation, pred)
1 - sum(diag(ct)) / sum(ct)
## [1] 0.273
```

Cross-Validation and Predictive Accuracy

Predictive accuracy is also similar; and note how we do not reuse diagn2 here but we re-estimate it to avoid using the data twice.

```
xval.diagn = bn.cv(inted, diagn, loss = "pred-lw", runs = 10,
loss.args = list(target = "creation"),
fit = "bayes", fit.args = list(iss = 1))
mean(sapply(xval.diagn, attr, "mean"))
## [1] 0.274
xval.diagn2 = bn.cv(inted, "tabu", loss = "pred-lw", runs = 10,
loss.args = list(target = "creation"),
algorithm.args = list(whitelist = edges(diagn)),
fit = "bayes", fit.args = list(iss = 1))
mean(sapply(xval.diagn2, attr, "mean"))
## [1] 0.276
xval.progn = bn.cv(inted, progn, loss = "pred-lw", runs = 10,
loss.args = list(target = "creation"),
fit = "bayes", fit.args = list(iss = 1))
mean(sapply(xval.progn, attr, "mean"))
## [1] 0.278
```

Scales and Predictive Accuracy

Interestingly, the summary variables desirability and feasibility (which d-separate creation from the six scales) improve the predictive accuracy.

```
from = c("ACT", "LC", "NORM", "SE", "FAC", "OBS")
xval.diagn = bn.cv(inted, diagn, loss = "pred-lw", runs = 10,
loss.args = list(target = "creation", from = from),
fit = "bayes", fit.args = list(iss = 1))
mean(sapply(xval.diagn, attr, "mean"))
## [1] 0.307
xval.diagn2 = bn.cv(inted, "tabu", loss = "pred-lw", runs = 10,
loss.args = list(target = "creation", from = from),
algorithm.args = list(whitelist = edges(diagn)),
fit = "bayes", fit.args = list(iss = 1))
mean(sapply(xval.diagn2, attr, "mean"))
## [1] 0.309
xval.progn = bn.cv(inted, progn, loss = "pred-lw", runs = 10,
loss.args = list(target = "creation", from = from),
fit = "bayes", fit.args = list(iss = 1))
mean(sapply(xval.progn, attr, "mean"))
## [1] 0.31
```

Learning and Interpretability

The BN proposed by tabu() as an extension of the diagnostic BN produces, at least,an interesting statistical model from the theoretical point of view. There are two new edges associating two nodes and this shed light to previously unexplored hypotheses.

- The edge desirability $\rightarrow$ feasibility makes sense because you will perceive more desirable to create a new business if it is considerate feasible.
- The edge FAC $\rightarrow$ OBS also makes sense because if you perceive few obstacles, you would perceive more facilitators to do a new venture.

This second edge is particularly interesting form a practical point of view in the context of entrepreneurship promotion. For example, it would be advisable to introduce laws or public-private incentives in order to reduce the subjective perception of difficulties in potential entrepreneurs.

Queries

Indeed increasing feasibility dramatically improves the attitude towards business creation.

```
fitted.diagn2 = bn.fit(diagn2, inted)
cpquery(fitted.diagn2, (creation == "Yes"),
evidence = list(feasibility = "A.lot.feasible"), method = "lw")
## [1] 0.798
cpquery(fitted.diagn2, (creation == "Yes"),
evidence = list(feasibility = "Very.little.feasible"), method = "lw")
## [1] 0.137
```

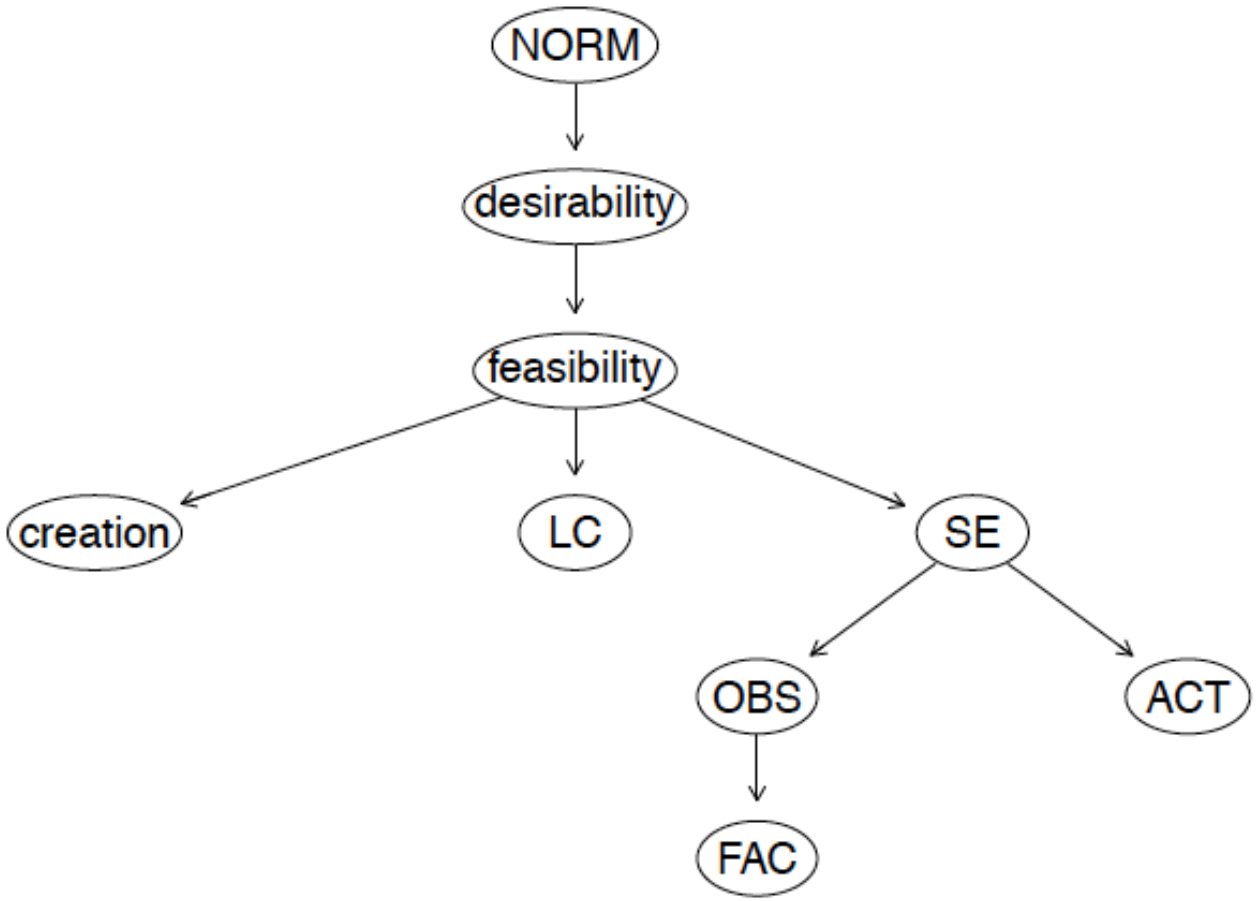
The same is true for decreasing OBS, but not as much; the reason is that OBS is farther away from creation so the effect of the conditioning is smaller.

```
cpquery(fitted.diagn2, (creation == "Yes"), evidence = list(OBS = "High"),
method = "lw")
## [1] 0.351
cpquery(fitted.diagn2, (creation == "Yes"), evidence = list(OBS = "Low"),
method = "lw")
## [1] 0.276
```

The DAG from Structure Learning is not Interpretable

On the other hand, we can learn a DAG directly from the data, but the result has no clear interpretation because the edges do not map well to what we know from the literature.

```
graphviz.plot(tabu(inted), shape = "ellipse")
```



That's It, Thanks!