Directed Graphical Models

Gunwoong Park

Lecture Note

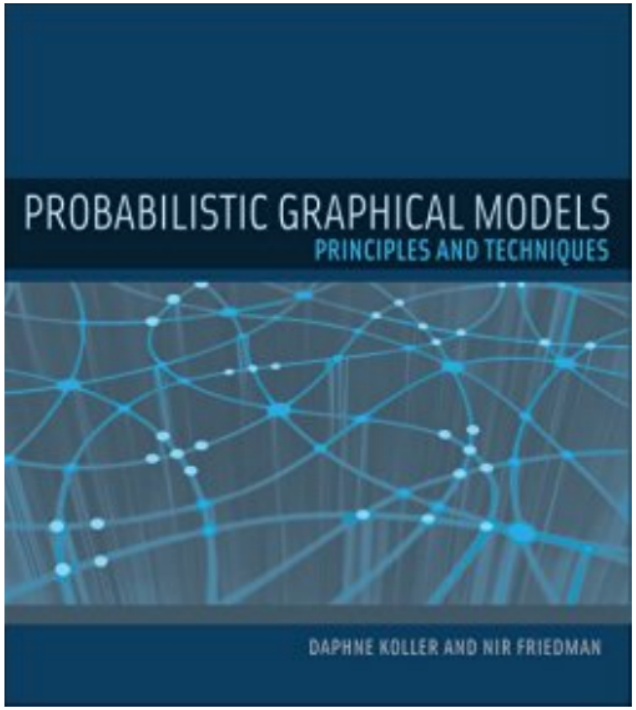
University of Seoul

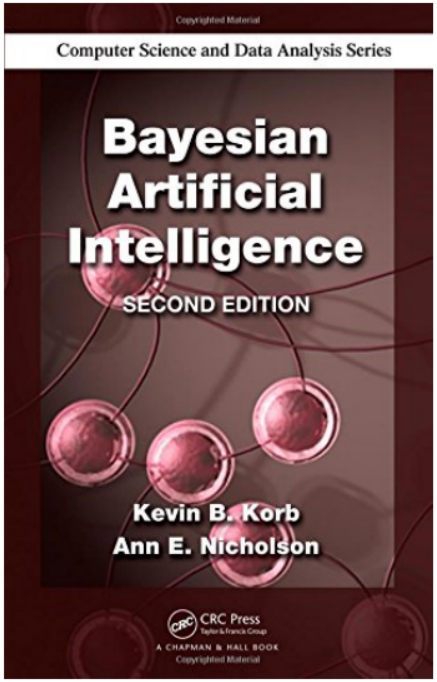
- Introduction
- Definitions and Useful Notations
- Causal Inference
- Structure Learning
    - ▷ Exponential Family DAG Models
    - ▷ GHD DAG Models
    - ▷ Identifiable Gaussian SEMs

Introductions

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:

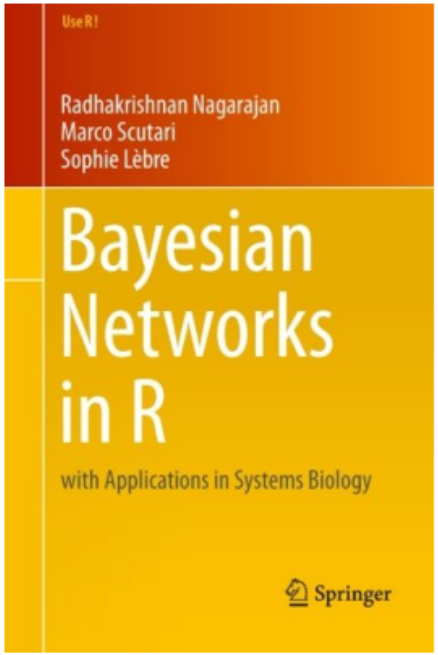
```
source("http://bioconductor.org/biocLite.R")
biocLite(c("graph", "Rgraphviz"))
```

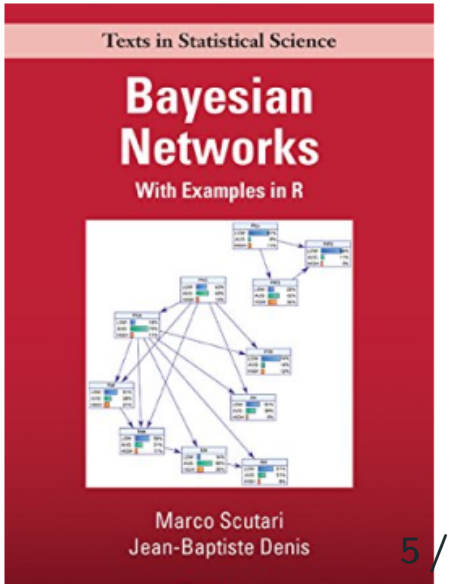
For exact inference on discrete Bayesian networks:

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

Other packages from CRAN:

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





Definition

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

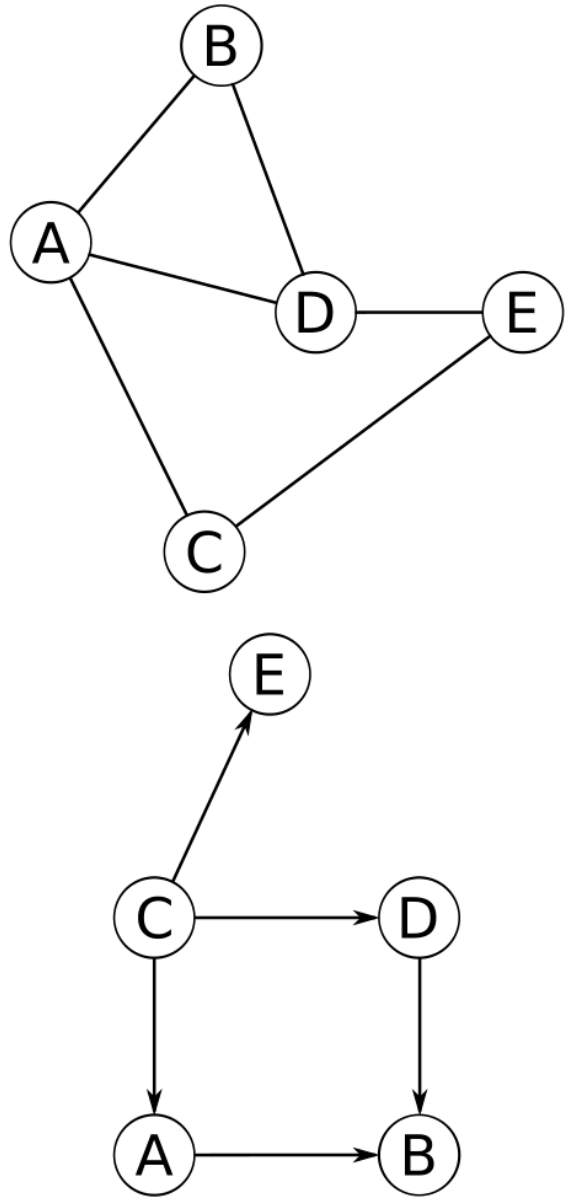
The first component of a BN is a graph. A graph G is a mathematical object with:

- 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.



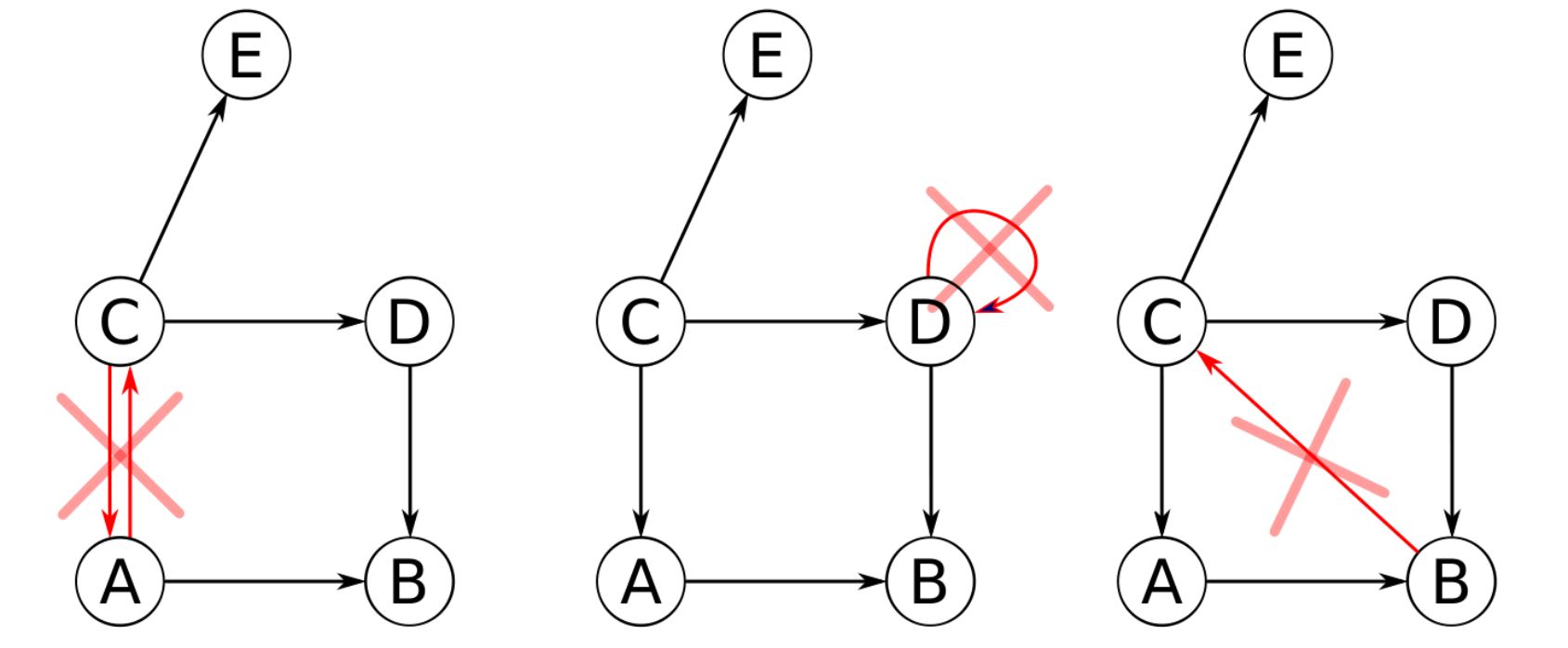
Graphs Notations

- V : A set of nodes, e.g. $V = \{X_1, X_2, X_3\}$.
- E : A set of directed edges, e.g. $E = \{(X_1, X_2), (X_2, X_3)\}$.
- 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

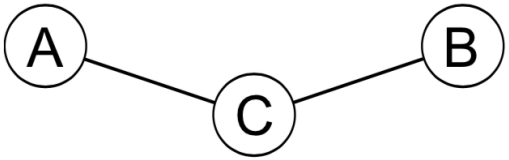
BNs use a specific kind of graph called a directed acyclic graph, that:

- 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)

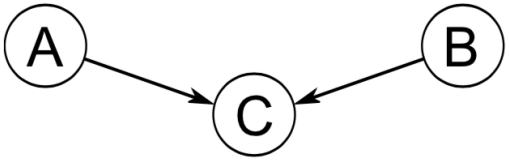
separation (undirected graphs)



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

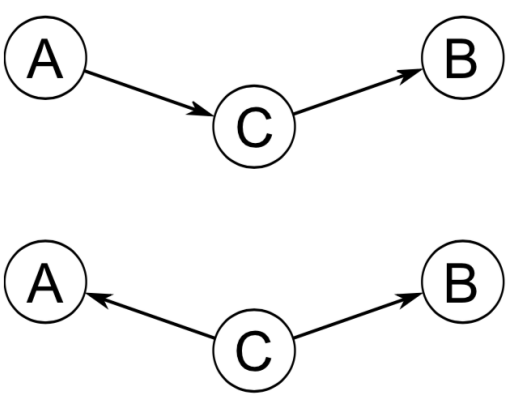
$$\mathrm{P}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \mathrm{P}(\mathbf{A} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{B} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{C})$$

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 (D-separation)

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.

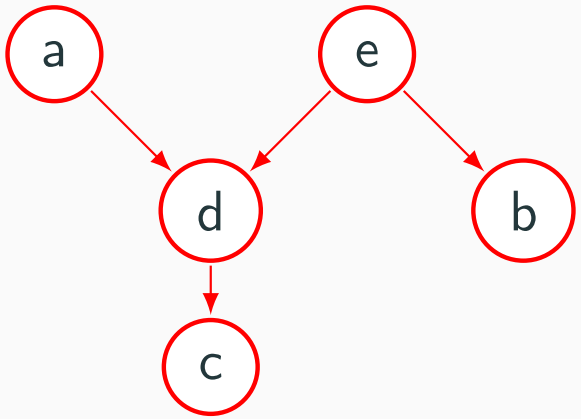
- If A, B and C are three disjoint subsets of nodes ($\subset V$) in G, then C is said to *d-separate* A from B, denoted $A \perp\!\!\!\perp_G B \mid C$, if along every path between a node in A and a node in B there is a node v satisfying one of the following two conditions:
  - v has converging edges (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.

Summary of D-separation

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

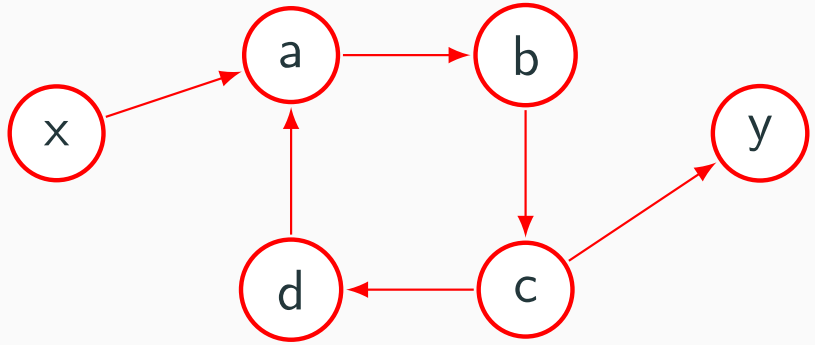
D-Separation Example

- 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.

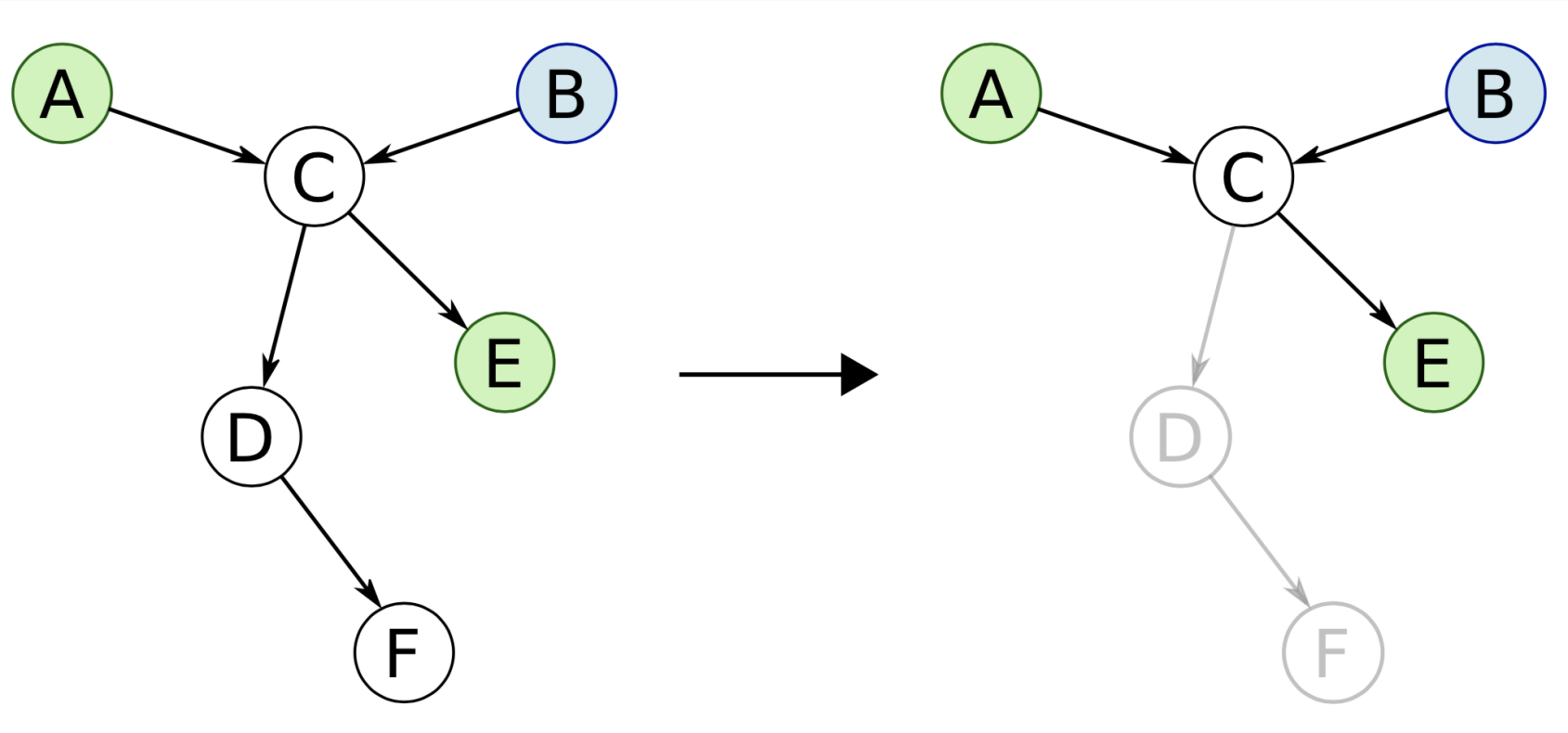


D-Separation Example in Directed Cyclic Graphs

- 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.

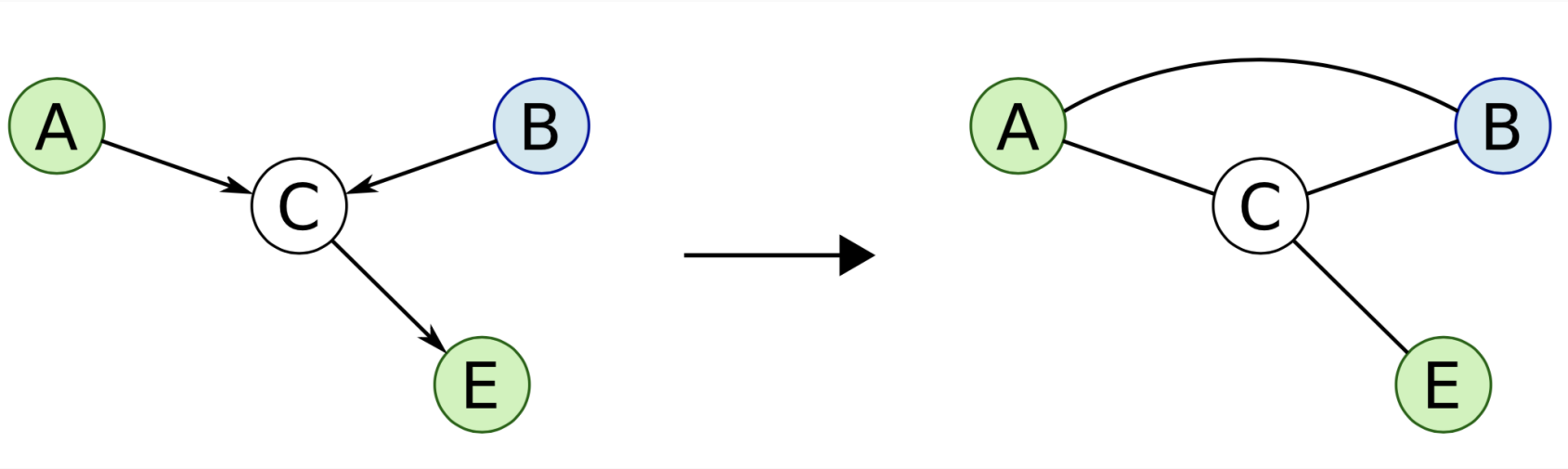


A Simple Algorithm to Check D-Separation (I)



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)

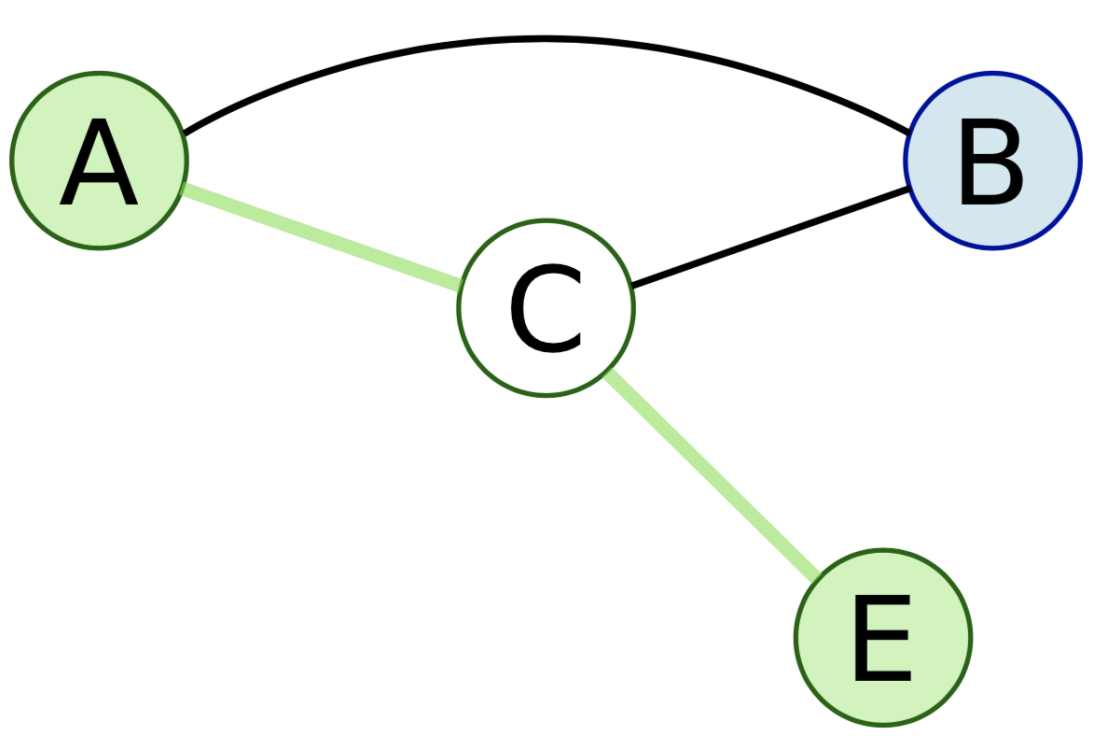


Transform the subgraph into its moral graph by

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.

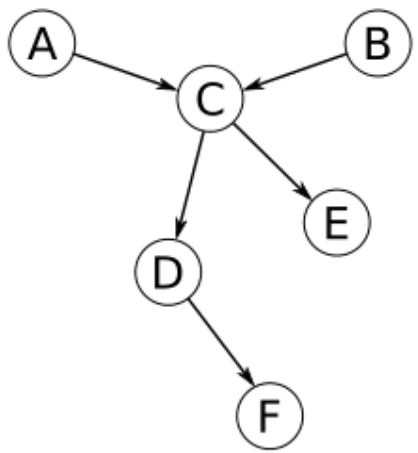
A Simple Algorithm to Check D-Separation (III)



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



Graphical separation

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

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

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

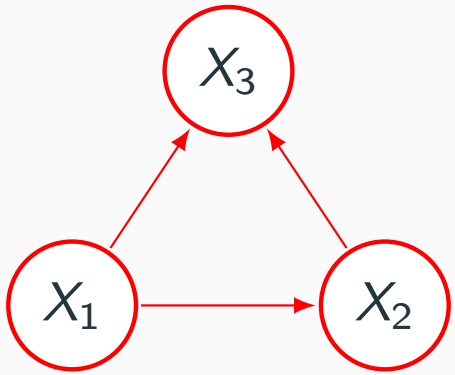
G is said to be a perfect map of M if it is both a D-map and an I-map, that is

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

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 $X_{\mathrm{Pa}(j)}$.

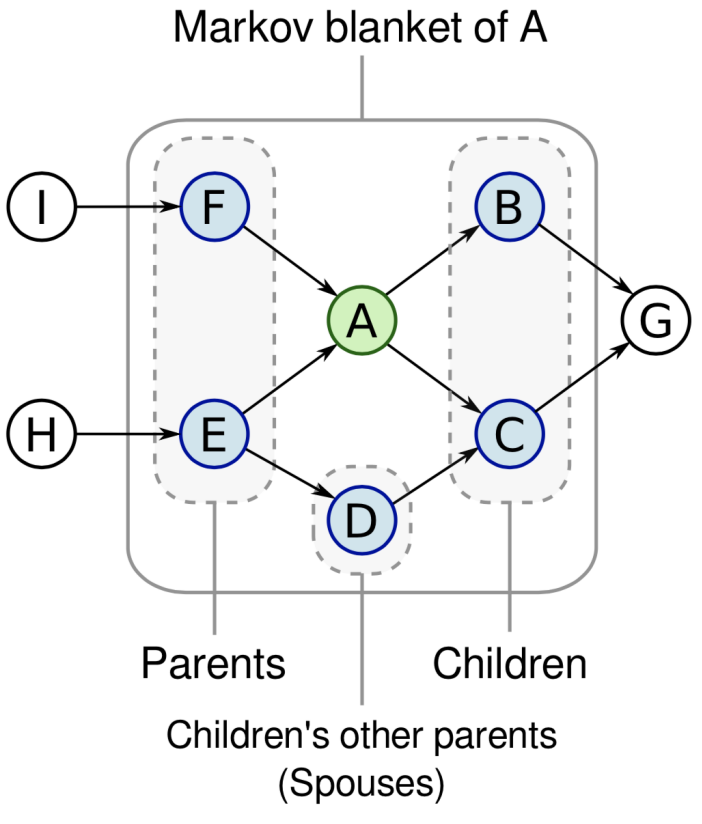
The Local Markov Property (II)

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

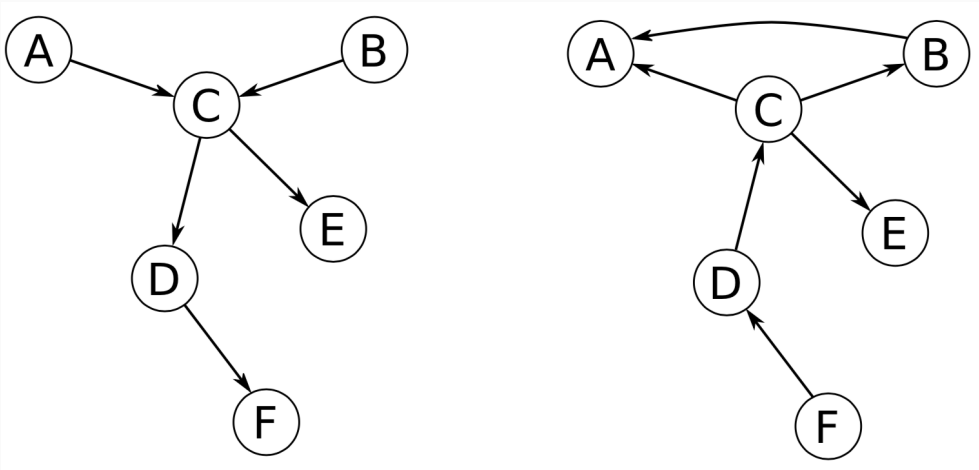
A DAG uniquely identifies a factorization of $P(X)$; the converse is Not necessarily true. Consider again the DAG on the left:

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

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).

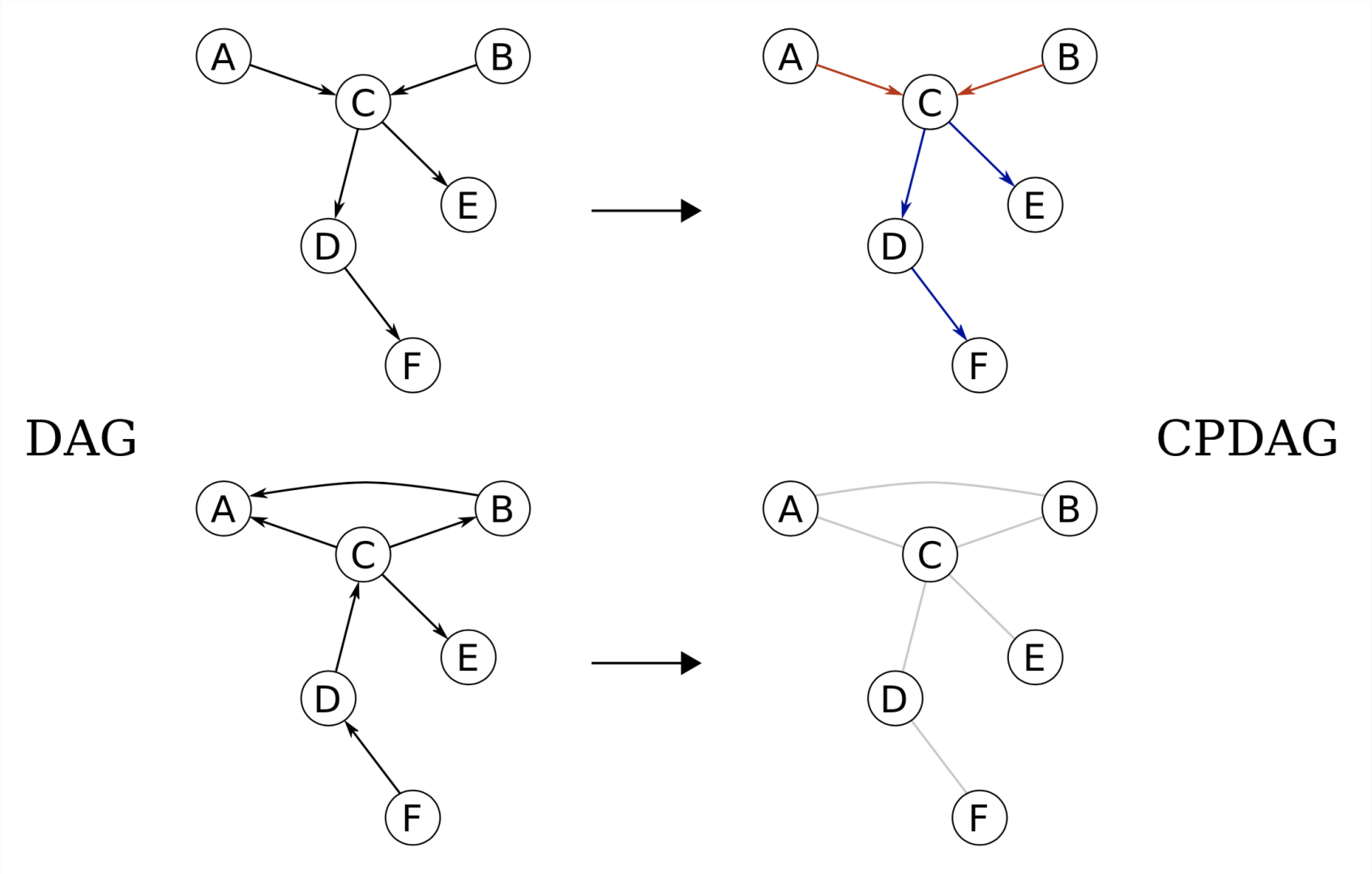
What Are V-Structures, and What Are Not

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:

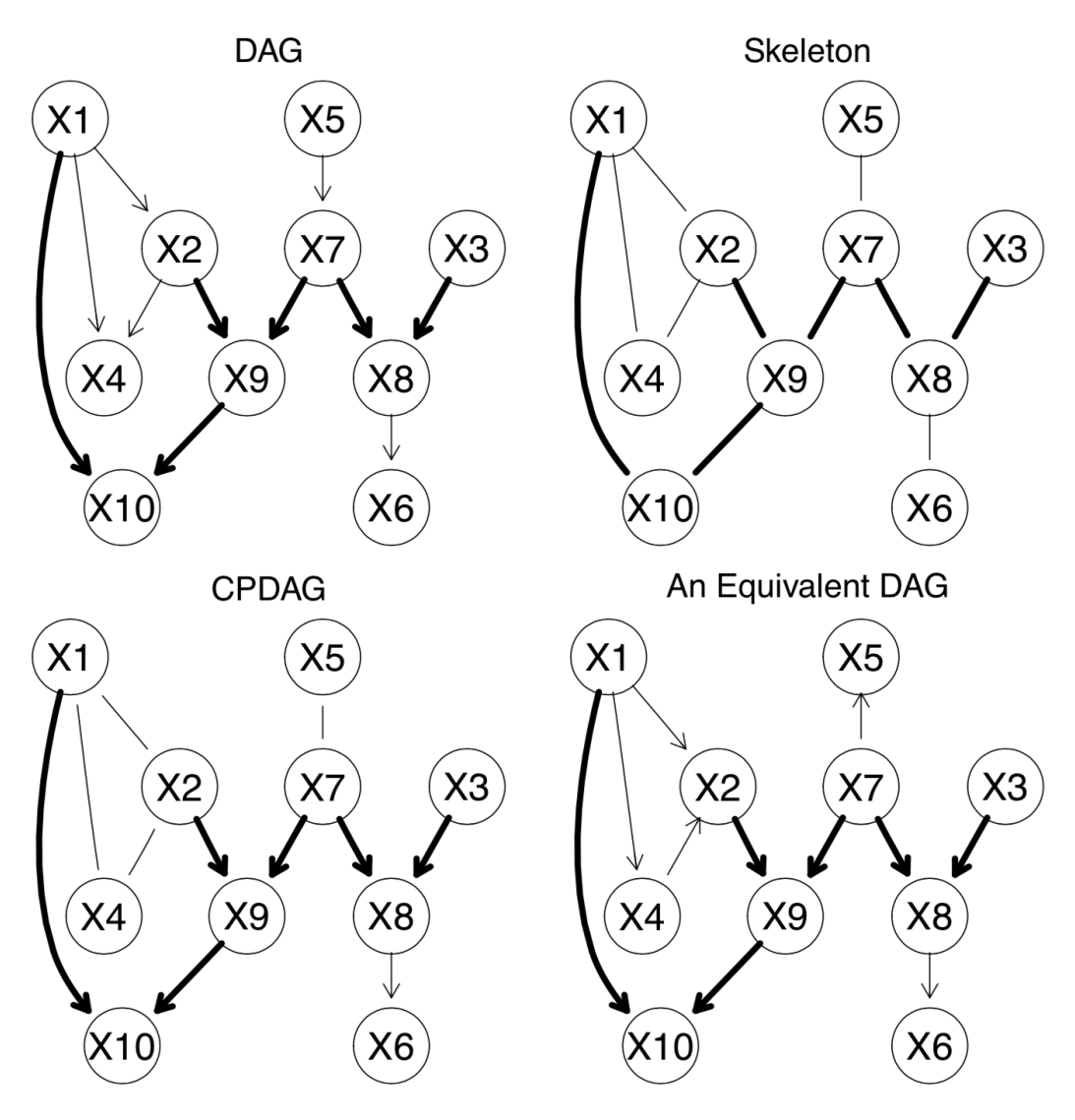
$$\underbrace{P(A)P(C \mid A)P(B \mid A, C)}_{A \to B \leftarrow C,\ A \to C} = \underbrace{P(A)P(C \mid B, A)P(B \mid A)}_{B \to C \leftarrow A,\ A \to B}.$$

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.

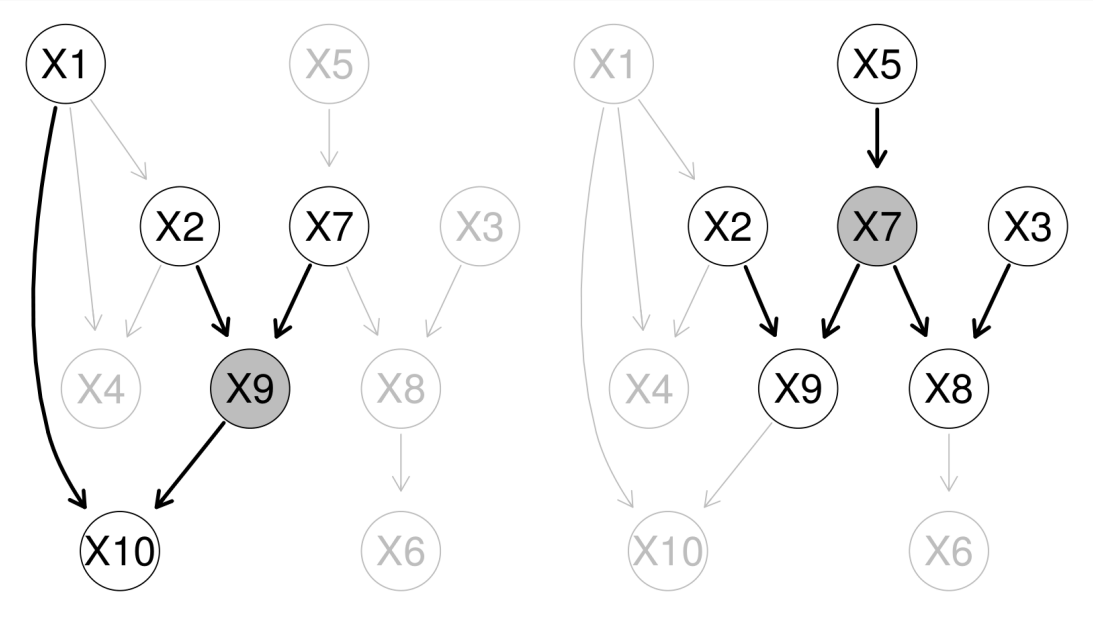
Completed Partially Directed Acyclic Graphs (CPDAGs)



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



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



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.

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:

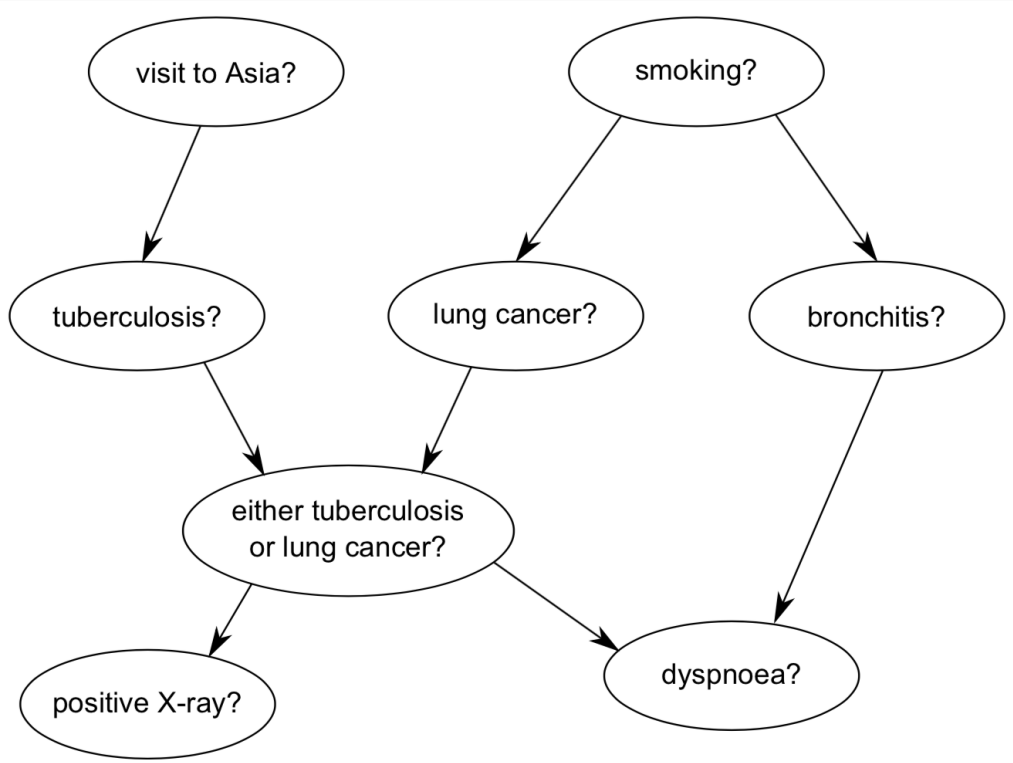
- can be learned efficiently from data;
- is flexible (distributional assumptions should not be too strict);
- is easy to query to perform inference.

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

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

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

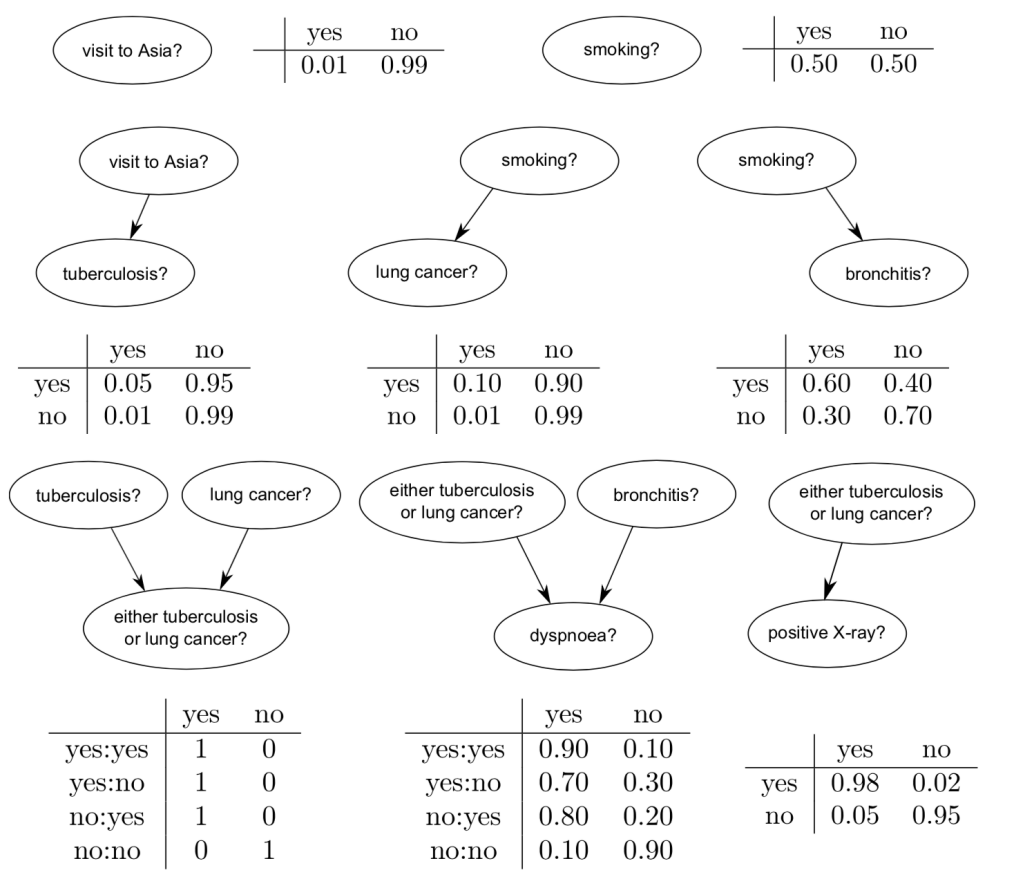
Discrete BNs



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 :
$2^8 - 1 = 255$

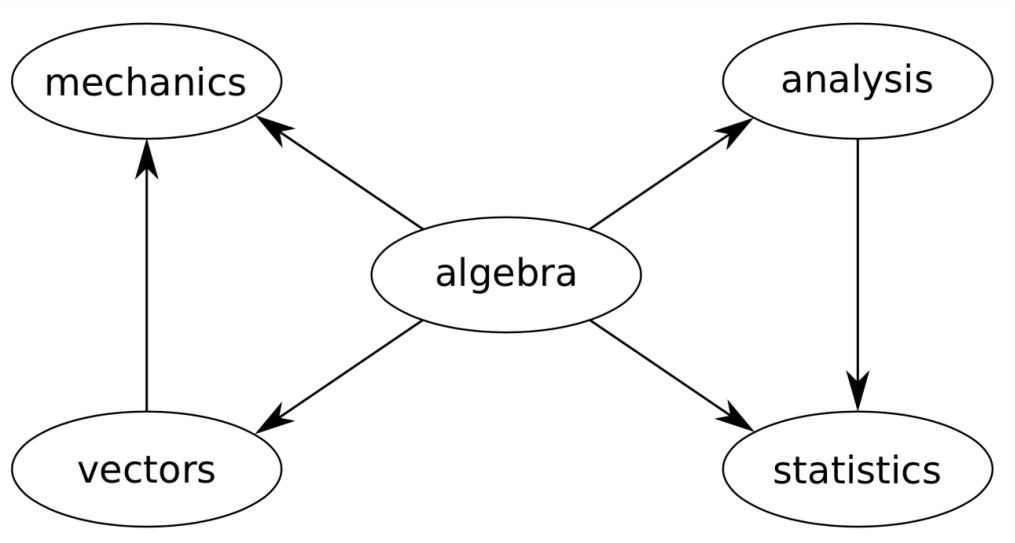
Conditional Probability Tables (CPTs)



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

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 $X_i \perp\!\!\!\perp_P X_j \mid X \backslash \{X_i, X_j\}$. The absence of an edge $X_i \to X_j$ in the DAG implies $X_i \perp\!\!\!\perp_G X_j \mid X \backslash \{X_i, X_j\}$, which in turn implies $X_i \perp\!\!\!\perp_P X_j \mid X \backslash \{X_i, X_j\}$.

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

Partial Correlations and Linear Regressions

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

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

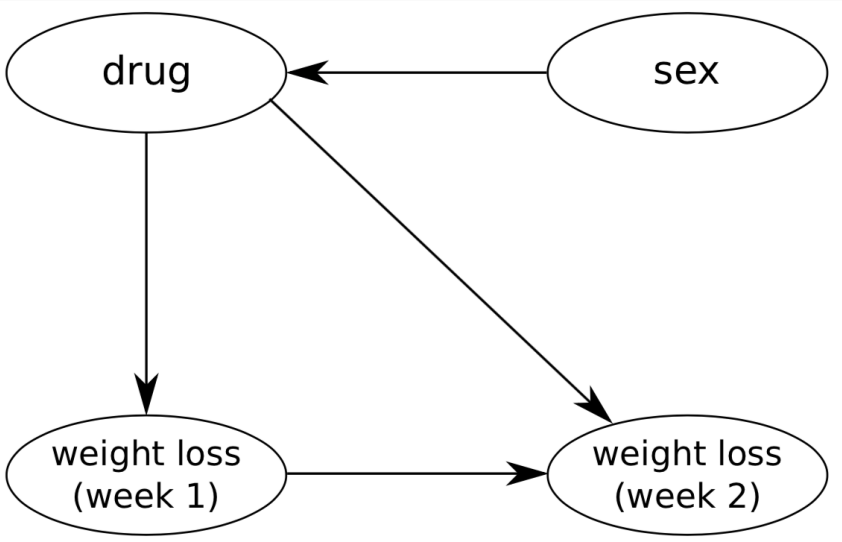
That is because $\Omega_{ij} \propto \beta_j$ for X_i, so $\beta_j > 0$ if and only if $\Omega_{ij} > 0$. Also $\Omega_{ij} \propto \rho_{ij}$, the partial correlation between X_i and X_j, so we are implicitly assuming all probabilistic dependencies are linear.

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

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.

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 $X_i \to X_j \to X_k$ and $X_i \leftarrow X_j \to X_k$ 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.

Causal Inference

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.

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).

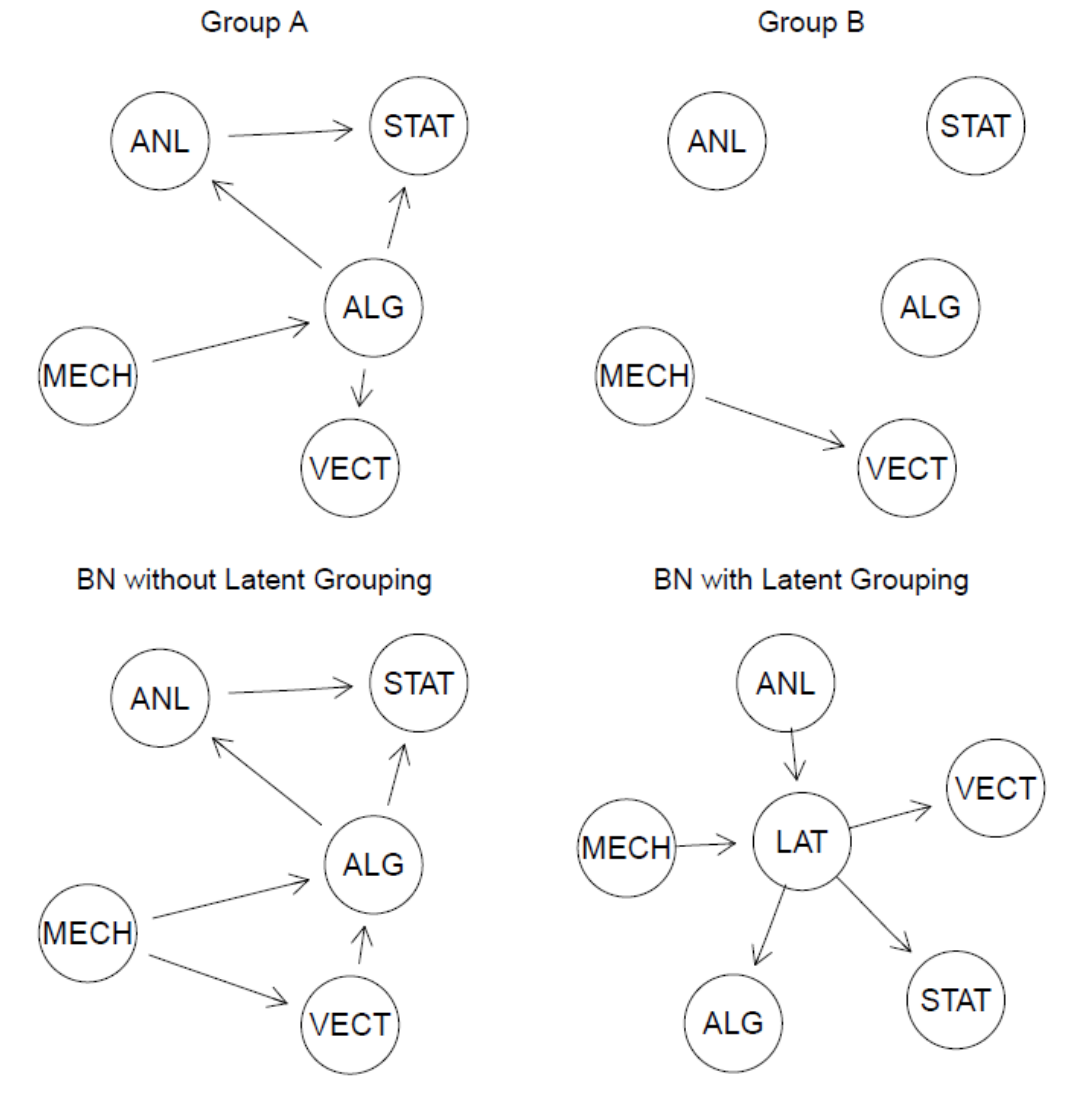
```
library(bnlearn)
head(marks)
## MECH VECT ALG ANL STAT
## 1 77 82 67 67 81
## 2 63 78 80 70 81
## 3 75 73 71 66 81
## 4 55 72 63 70 68
## 5 63 63 65 70 63
## 6 53 61 72 64 73
```

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]"
```

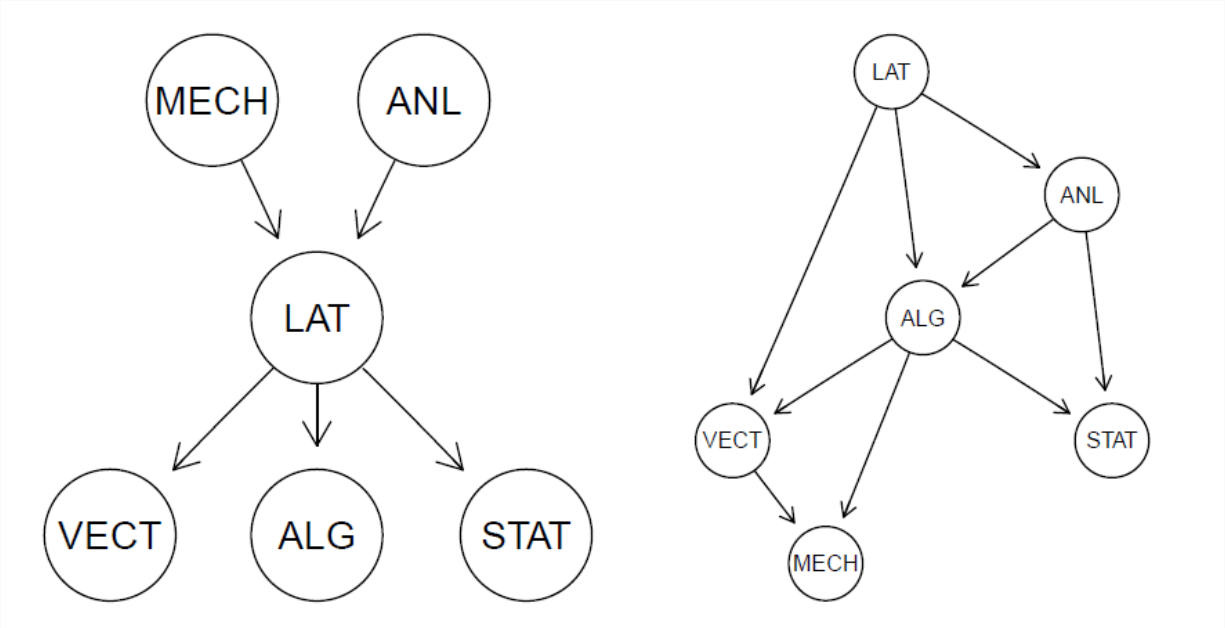
This BN seems to provide a simple interpretation of the relationships between the topics: the grades in mechanics and analysis can be used to infer which group a student belongs to, and that in turn influences the grades in the remaining topics.

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

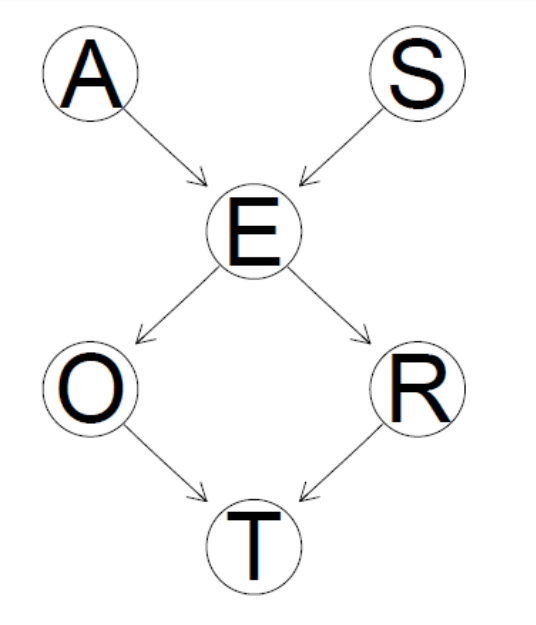
Once we have a causal BN we are happy with, we can again focus on using it to answer relevant questions. In the context of causal networks, we call this causal inference. Compared to the posterior inference we have seen in the previous lecture:

- 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

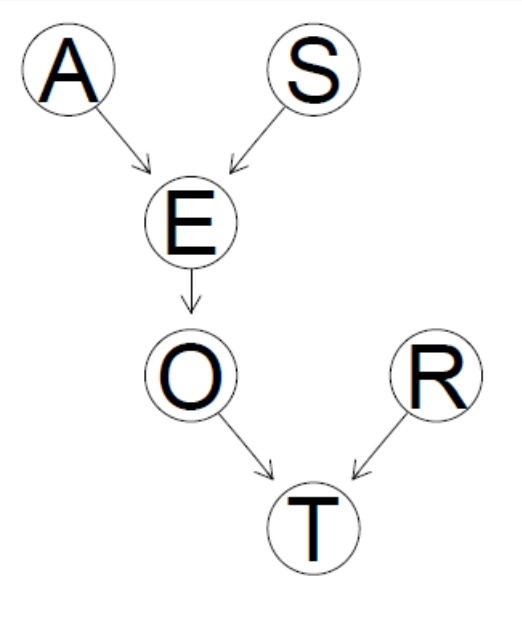
```
coef(survey.bn$R)
## E
## R high uni
## small 0.25 0.20
## big 0.75 0.80
```

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

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 difficult 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

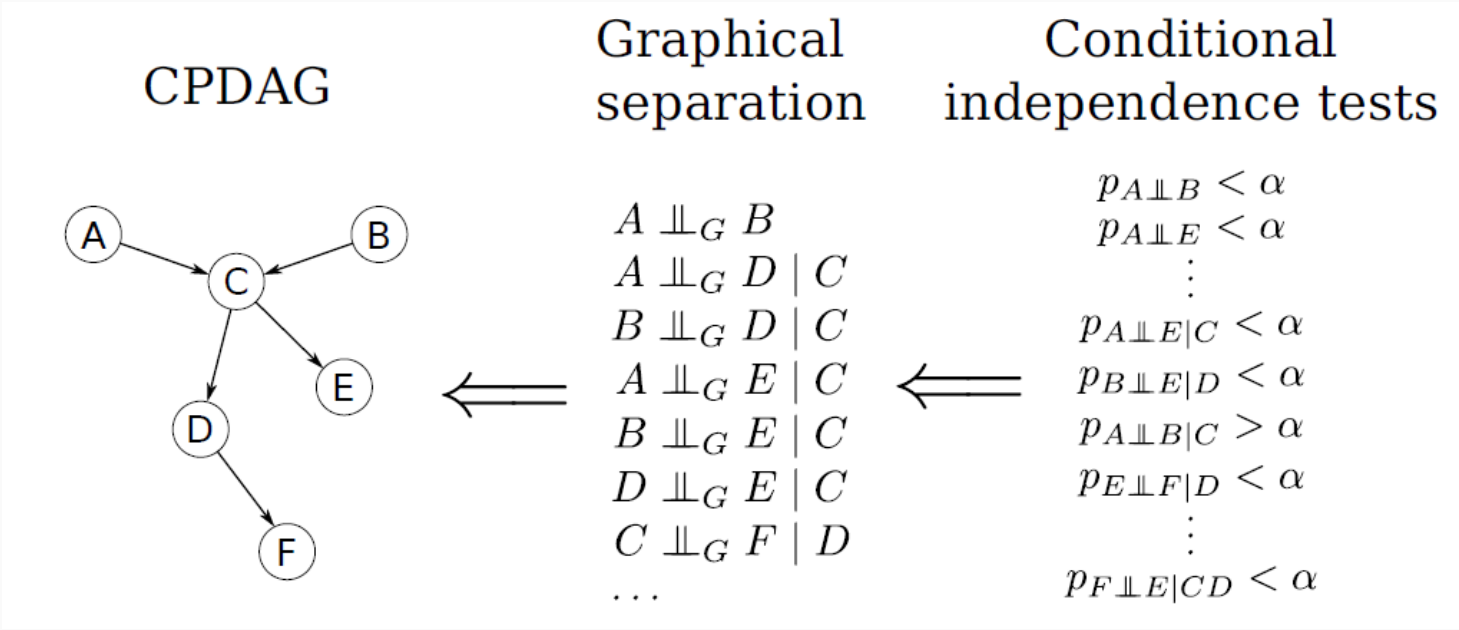
- There must be a one-to-one correspondence between the nodes in the DAG and the random variables in X; there must not be multiple nodes which are deterministic functions of a single variable.
- All the relationships between the variables in X must be conditional independencies, because they are by definition 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

DAG models 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;
- not all P(X) have a faithful DAG.

The Inductive Causation Algorithm

1. For each pair of variables A and B in X for set $S_{AB} \subset X$ 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 neighbor C, check whether $C \in S_{AB}$. If this is not true, set the direction of the edges $A - C - B$ to $A \rightarrow C \leftarrow B$.
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 regularize 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 regularize 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.

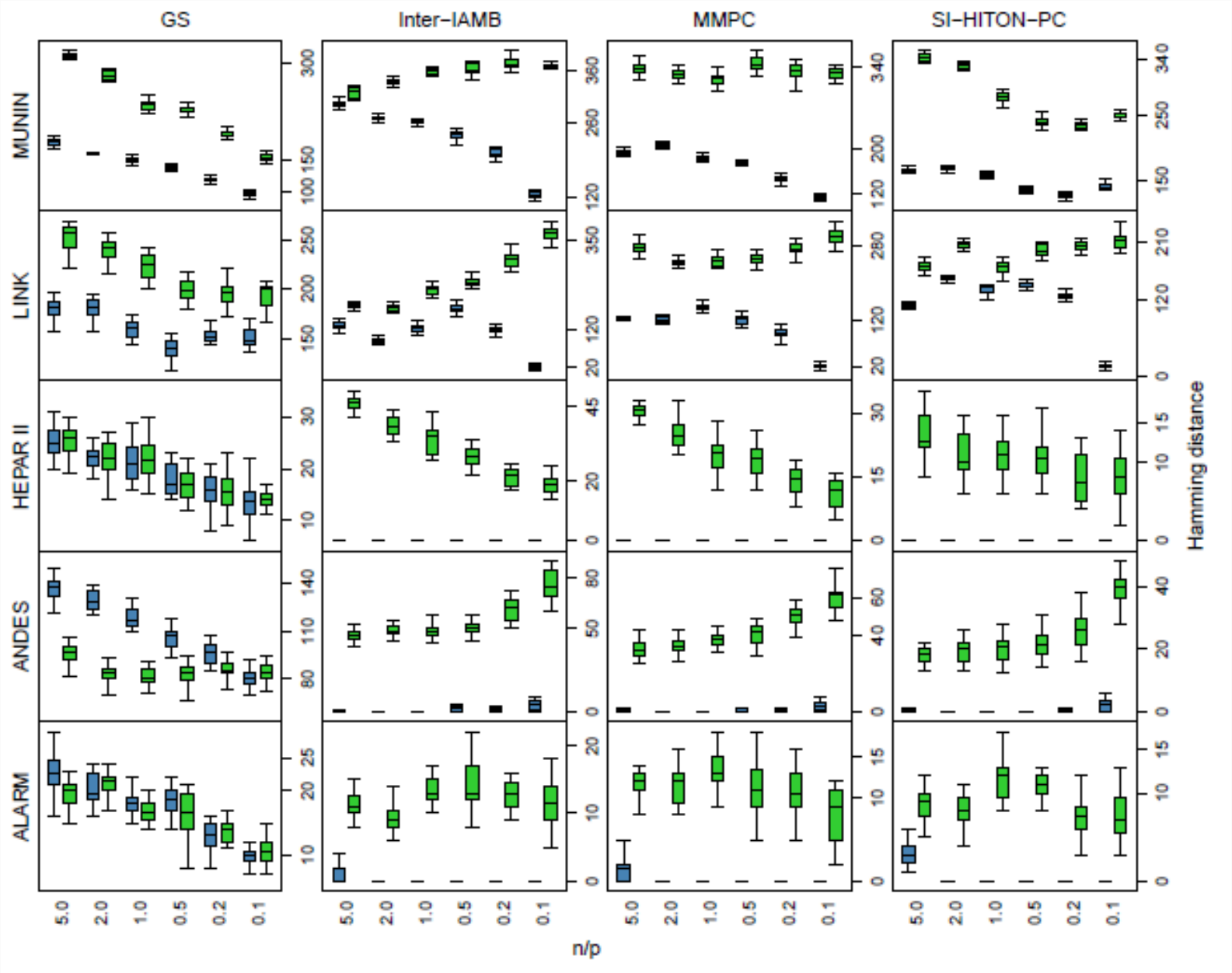
```
cpdag = si.hiton.pc(asia, undirected = FALSE)
cpdag
##
## Bayesian network learned via Constraint-based methods
##
## model:
## [partially directed graph]
## nodes: 8
## edges: 5
## undirected edges: 1
## directed edges: 4
## average markov blanket size: 1.75
## average neighbourhood size: 1.25
## 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

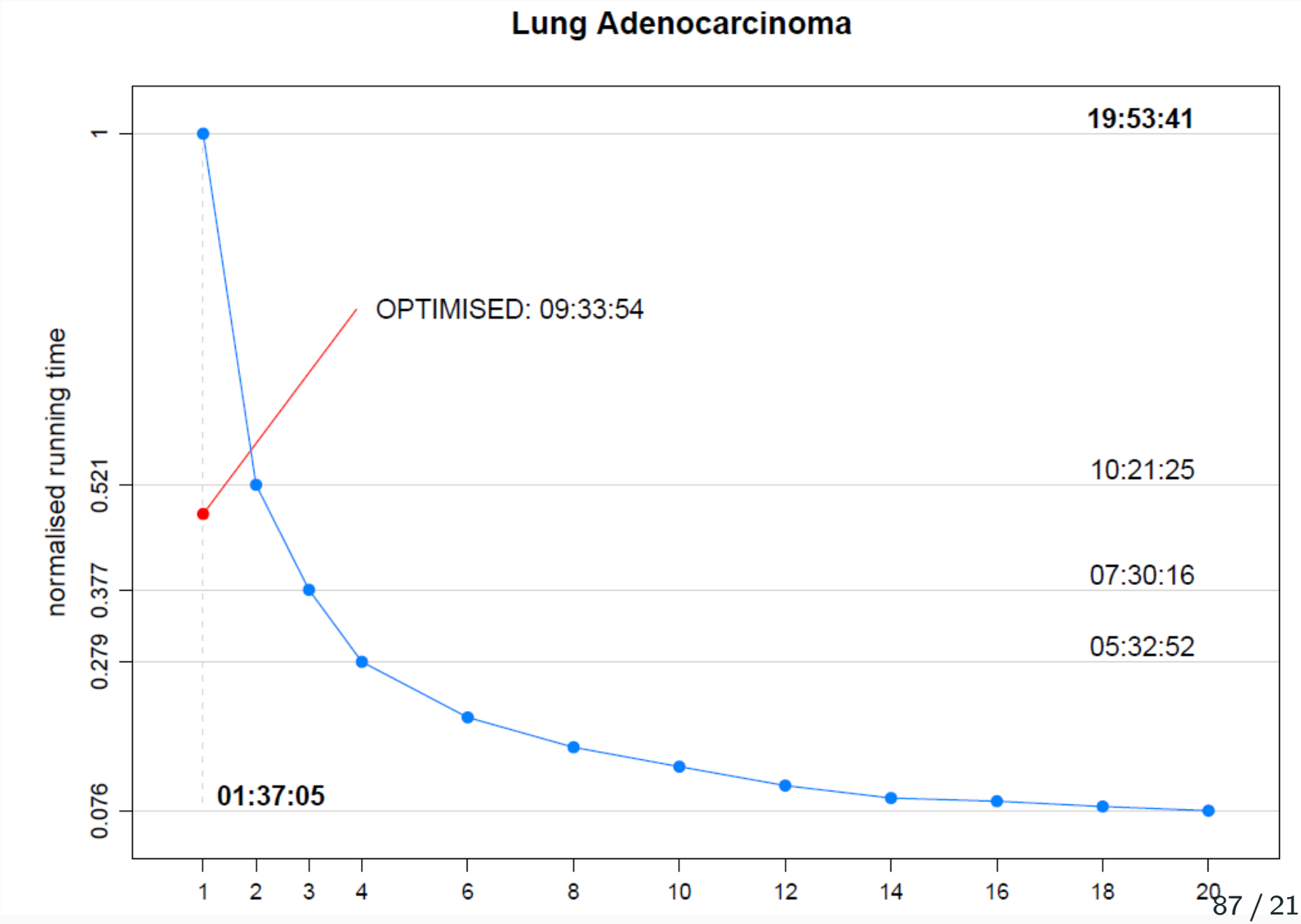
The arguments for the tuning parameters of constraint-based learning algorithms have the same names in the respective functions:

- 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...

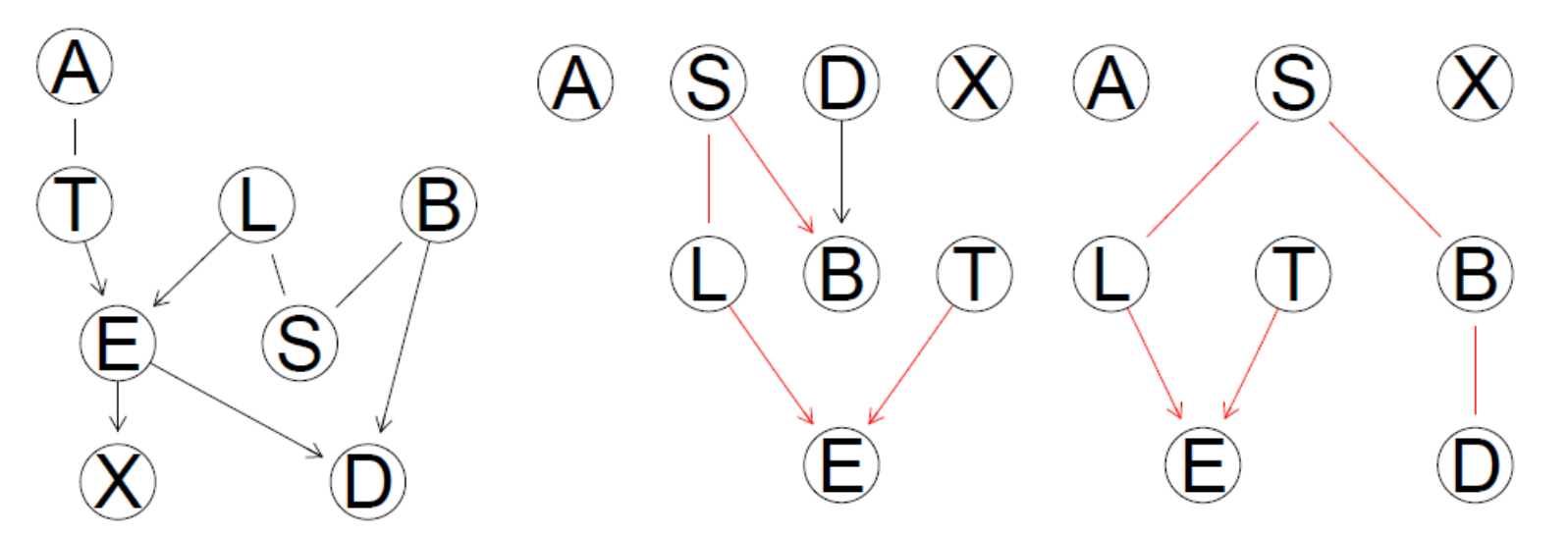


... Because Parallel Computing is Safer and Faster



bnlearn: With and Without Backtracking

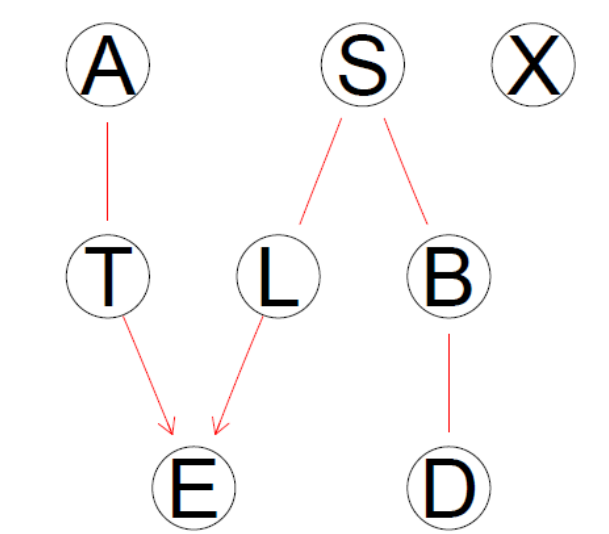
```
par(mfrow = c(1, 3))
true.dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
graphviz.plot(cpdag(true.dag))
graphviz.plot(cpdag, highlight = list(edges = edges(cpdag(true.dag))), )
cpdag2 = si.hiton.pc(asia, undirected = FALSE, optimized = FALSE)
graphviz.plot(cpdag2, highlight = list(edges = 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 search unfeasible in practice, regardless of the goodness-of-fit measure (called network score) used in the process. However, we can use heuristics in combination with decomposable scores, i.e.

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

such as

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

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

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* $\leftarrow$ *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 $\mathcal{G}^*$, $\text{Score}_{\mathcal{G}^*} \leftarrow \text{Score}(\mathcal{G}^*)$:
     - 4.1.2 if $\text{Score}_{\mathcal{G}^*} > \text{Score}_{\mathcal{G}}$, set $\mathcal{G} \leftarrow \mathcal{G}^*$ and $\text{Score}_{\mathcal{G}} \leftarrow \text{Score}_{\mathcal{G}^*}$.
   - 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 are independent);
- 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 evidence that this is a problematic choice:

- The prior is actually not uninformative.
- 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 DAG is determined by a complex interaction between α and $\mathcal{D}$.

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:

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

for GBNs:

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

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

$$LL(X_i, X_{\text{Pa}(i)}) = \prod_{m=1}^{n} N(x_m; \mu_{X_i,\delta_m} + \gamma_m \beta_{X_i,\delta_m}, \sigma^2_{X_i,\delta_m}),$$
$$|\Theta_{X_i}| = |\Delta_{X_i}| \times (|\Gamma_{X_i}| + 1).$$

bnlearn: Hill Climbing with BIC (MARKS)

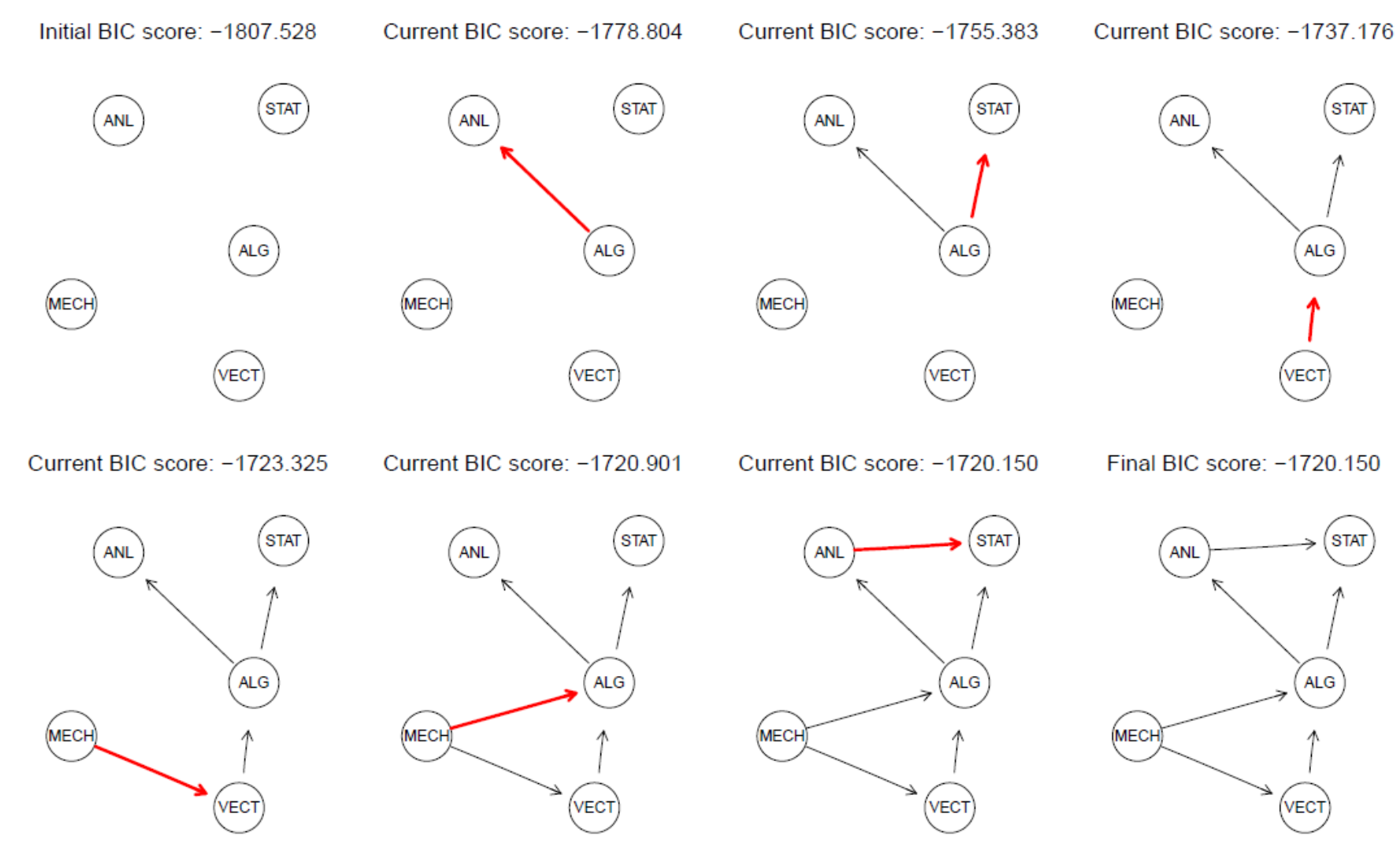
hc() implements hill-climbing with random restarts, and can use different scores much like functions implementing constraint-based algorithms can use different tests.

```
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)



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 Preseeded Networks

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

```
capture.output(hc(asia, score = "bde", iss = 1, debug = TRUE))[c(2, 6:7)]
## [1] "* starting from the following network:"
## [2] " model:"
## [3] " [A][S][T][L][B][E][X][D] "
```

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

```
capture.output(hc(asia, score = "bde", iss = 1,
start = random.graph(names(asia)), debug = TRUE))[c(2, 6:7)]
## [1] "* starting from the following network:"
## [2] " model:"
## [3] " [A][S][T|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 Search: hill-climbing over equivalence classes rather than graph structures; the search space is much smaller.
- Tabu Search: 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 Search

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

```
debugging.output =
  capture.output(tabu(asia, score = "bde", iss = 1, tabu = 10,
    max.tabu = 5, debug = TRUE))
head(grep("^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 search may get stuck in a local maximum.
- They are more stable than constraint-based algorithms.
- They require a definition of both the global and the local distributions, and a matching decomposable, network score. This means, for instance, that nobody can use them with ordinal variables because it is difficult to specify the global distribution. On the other hand, there are conditional independence test.
- 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 satisfies those constraints and define a new set of constraints to improve on.

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

Sparse Candidate and MMHC Algorithms

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 Comparison

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

```
true.dag = model2network("[A][S][T|A][L|S][b|S][D|b:E][E|T:L][X|E]")
unlist(compare(cpdag(asia.rsmax2), cpdag(true.dag)))
## tp fp fn
##  4  4  1
shd(asia.rsmax2, true.dag)
## [1] 4
unlist(compare(cpdag(asia.restart), cpdag(true.dag)))
## tp fp fn
##  7  1  0
shd(asia.restart, true.dag)
## [1] 1
unlist(compare(cpdag(cpdag2), cpdag(true.dag)))
## tp fp fn
##  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.

PC Algorithm

- How it works
- Theoretical Guarantees
- Computational Complexity

GES Algorithm

- How it works
- Theoretical Guarantees
- Computational Complexity

MMHC Algorithm

- How it works
- Theoretical Guarantees
- Computational Complexity

Constants Moments Ratio DAG Models

Moments related DAG models

- Poisson DAG models
- Quadratic Variance Function (QVF) - DAG models
- GHD DAG models
- Constants Moments Ratio (CMR) DAG Models

Poisson DAG Models, Park and Raskutti 2018

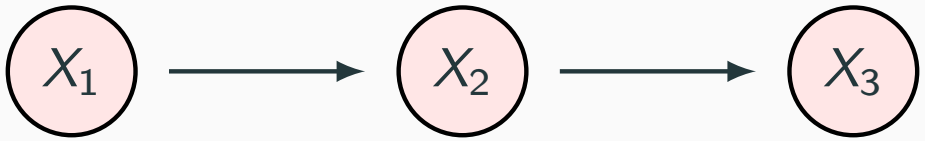
Definition of Poisson DAG models

Poisson DAG models

Poisson DAG models is that each conditional distribution is Poisson and the rate parameter depends only on its parents:

$$X_j \mid X_{\mathrm{Pa}(j)} \sim \mathrm{Poisson}(g_j(X_{\mathrm{Pa}(j)})),$$

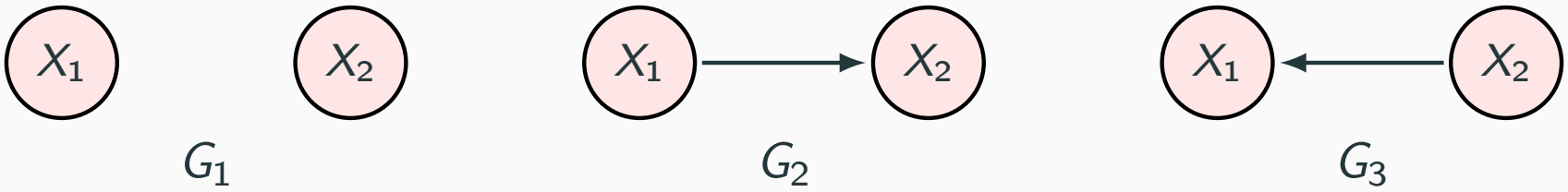
where g_j is an arbitrary positive function.



$$P(X_1) \sim Poi(\lambda_1), \quad P(X_2 \mid X_1) \sim Poi(g_2(X_1)), \quad P(X_3 \mid X_2) \sim Poi(g_3(X_2)).$$

Model Identifiability

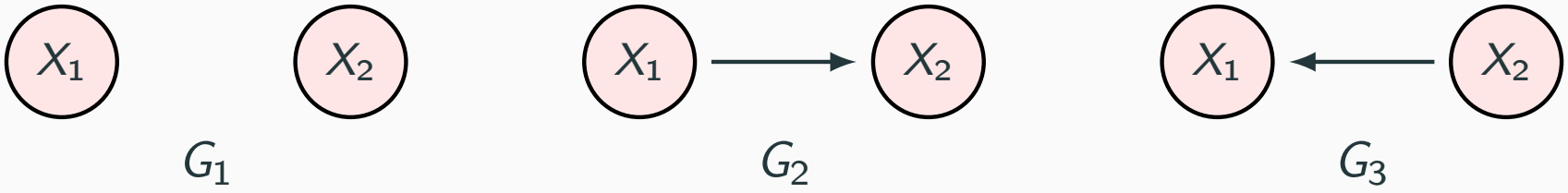
Is it possible to recover a graph from data? **Partially Yes.**



- We can distinguish G_2 and G_3 from G_1.
- We cannot identify the direction of an edge. Hence, we cannot distinguish G_2 from G_3.

Model Identifiability: Overdispersion

Is it possible to recover a graph from count data? Yes.



For Poisson random variable X, $\text{Var}(X) = \mathbb{E}(X)$ otherwise the variance is overdispersed relative to the mean.

- For G_1, $\text{Var}(X_1) = \mathbb{E}(X_1)$ and $\text{Var}(X_2) = \mathbb{E}(X_2)$.
- For G_2, $\text{Var}(X_1) = \mathbb{E}(X_1)$, while

$$\begin{aligned} \text{Var}(X_2) &= \mathbb{E}[\text{Var}(X_2 \mid X_1)] + \text{Var}[\mathbb{E}(X_2 \mid X_1)] \\ &> \mathbb{E}[\text{Var}(X_2 \mid X_1)] = \mathbb{E}(X_2), \end{aligned}$$

as long as $\text{Var}[\mathbb{E}(X_2 \mid X_1)] > 0$.

Identifiability for 3-node Chain Graph

$$X_1 = \epsilon_1, \quad X_2 = \beta_1 X_1 + \epsilon_2, \quad X_3 = \beta_2 X_2 + \epsilon_3$$

where $\epsilon_j \sim N(0, \sigma_j^2)$ for all $j \in \{1, 2, 3\}$.

Ordering Estimation: For π_1,

$$\text{Var}(X_2)/E(X_2) > \text{Var}(X_1)/\mathbb{E}(X_1)$$

Model Identifiability: Overdispersion

Theorem: Model Identifiability

For any node $j \in V$, non-empty $\text{Pa}_0(j) \subset \text{Pa}(j)$ and $S \subset \text{Pa}(j) \setminus \text{Pa}_0(j)$, if

$$\mathbb{E}\left(\text{Var}\left(\mathbb{E}(X_j \mid X_{\text{Pa}(j)}) \mid X_S\right)\right) > 0,$$

the DAG model is identifiable.

- All parents contribute to the variability of the rate parameter.
- The form of link functions (g_j) is not necessarily known.
- No restrictions on positive dependency between variables while Poisson undirected graphical models has (Yang et al. 2015)

Exponential Family DAG Models

Definition

Quadratic variance function (QVF) DAG models are DAG models where conditional distribution of each node given its parents $f_j(X_j \mid X_{\mathrm{Pa}(j)})$ satisfies the quadratic variance function:

$$\mathrm{Var}(X_j \mid X_{\mathrm{Pa}(j)}) = \beta_0 \mathbb{E}(X_j \mid X_{\mathrm{Pa}(j)}) + \beta_1 \mathbb{E}(X_j \mid X_{\mathrm{Pa}(j)})^2.$$

Examples:

- Poisson, Binomial, Negative Binomial, Gamma.

Distribution		β_0	β_1
Binomial	$\mathrm{Bin}(N, p)$	1	$-\frac{1}{N}$
Poisson	$\mathrm{Poi}(\lambda)$	1	0
Negative Binomial	$\mathrm{NB}(R, p)$	1	$\frac{1}{R}$
Gamma	$\mathrm{Gamma}(\alpha, \beta)$	0	$\frac{1}{\alpha}$

Lemma: Equidispersion Transformation

There exists a transformation $\mathbb{T}(X) := \omega \cdot X$ where $\omega = (\beta_0 + \beta_1 \mathbb{E}(X))^{-1}$ such that

$$\text{Var}(Y) = \mathbb{E}(Y).$$

Example:

- For $X \sim Bin(N, p)$, $Y = \frac{N}{N - \mathbb{E}(X)} X = \frac{X}{1-p}$.

$$\text{Var}(Y) = \mathbb{E}(Y) = \frac{N \cdot p}{1 - p}.$$

- For $X \sim Geo(p)$, $Y(X) = \frac{1}{1 + \mathbb{E}(X)} X$.

$$\text{Var}(Y) = \mathbb{E}(Y) = 1 - p.$$

Model Identifiability

Consider a QVF DAG model with quadratic variance coefficients (β_0, β_1).

Theorem: Model Identifiability

For any node $j \in V$, non-empty $\mathrm{Pa}_0(j) \subset \mathrm{Pa}(j)$ and $S \subset \mathrm{Pa}(j) \setminus \mathrm{Pa}_0(j)$, if $\beta_1 > -1$, and

$$\mathbb{E}\left(\mathrm{Var}\left(\mathbb{E}\left(\mathbb{T}_j(X_j) \mid X_{\mathrm{Pa}(j)}\right) \mid X_S\right)\right) > 0,$$

the DAG model is identifiable.

- Directed Ising models are not identifiable using our methods.

How to Use Overdispersion in Practice

- Overdispersion Score:

$$\widehat{\text{Var}}(X_j \mid X_S) - \widehat{\mathbb{E}}(X_j \mid X_S).$$

- Conditioning set for the j^{th} element of an ordering:
  - ▷ First $j-1$ elements of an ordering.
- Challenges:
  - ▷ Sample Complexity.
  - ▷ Computational Complexity.
- Space Reduction: **Moralized Graph, MEC**
  - ▷ Reduce the size of conditioning set.

Moralized Graph

- Moralized graph
  - ▷ Undirected graph representation of a DAG.
- Edges
  - ▷ Directed edges.
  - ▷ Additional edges between married nodes.
  - ▷ e.g., $X_1 \not\perp\!\!\!\perp X_2 \mid X_3$.

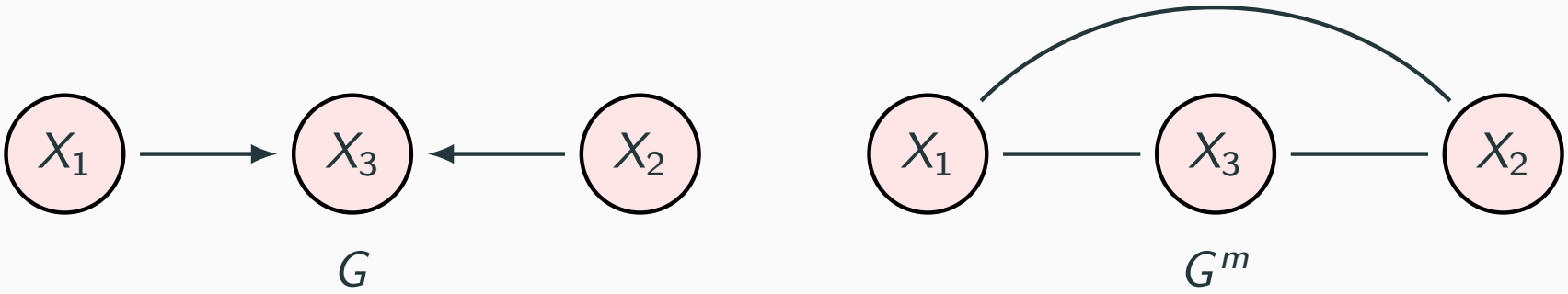
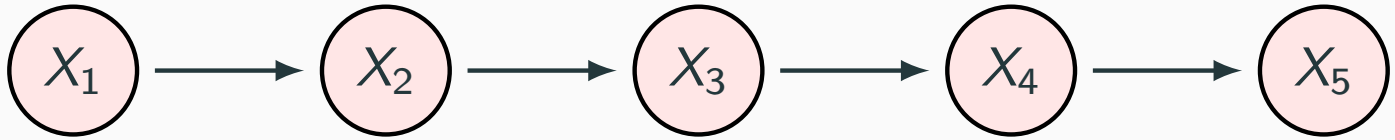


Figure 1: Moralized graph G^m for DAG G

Motivation for Space Reduction

A moralized graph provides **Candidate Parents**



- True ordering : $\pi^* = (X_1, X_2, X_3, X_4, X_5)$.

Suppose that the first 3 elements of the ordering are correctly estimated. In order to estimate the 4_{th} element of the ordering,

- Overdispersion score of X_4 given $\{X_1, X_2, X_3\}$.
- Overdispersion score of X_5 given $\{X_1, X_2, X_3\}$.

Space Reduction: Candidate Parents

Candidate parents set:

- Neighbors.
- All elements of an ordering before the node.

In order to estimate 4^{th} element of the ordering with the moralized graph given the partial ordering (X_1, X_2, X_3),

- Candidate parent set of X_4: X_3.
- Candidate parent set of X_5: $\emptyset$.

OverDispersion Scoring (ODS) Algorithm

STEP (1)

Space Reduction

- Regularized Regression.
- Graph learning algorithms.

STEP (2)

Ordering Estimation

- Overdispersion Score.

STEP (3)

Parents Estimation

- Regularized Regression.
- Existing DAG learning algorithms.

Assumptions

-

Assumptions

-

Consistency of the ODS Algorithm

Theorem: Consistency

For any $\epsilon > 0$, there exists a positive $C_\epsilon > 0$ such that if $n \geq C_\epsilon \left(\max \left\{ d_m^3 \log p^9, \log^{5+d} p \right\} \right)$,

$$P(G \neq \hat{G}) \leq \epsilon,$$

under the regularity conditions.

Notations

- $\hat{G}$: Estimated graph via the ODS algorithm.
- d_m: Maximum degree of the moralized graph.
- If the moralized graph is sparse, the algorithm works in the high-dimensional setting.

Sketch of the Proof I

-

Sketch of the Proof II

-

Sketch of the Proof III

-

Summary

- **Poisson DAG model**: For the multivariate count data.
- **Identifiability**: Overdispersion.
- **Algorithm**:
  - ▷ **Sample Complexity**: $\Omega\left(\max\left\{d_m^3 \log p^9, \log^{5+d} p\right\}\right)$.
  - ▷ **Computational Complexity**: $O\left(np^3\right)$.

Assumptions

-

Gaussian Linear SEMs

Gaussian Linear SEMs with Equal Variances

Algorithms

- Greedy DAG Search Algorithm: Peter and Buhlmann, 2014
- Gaussian SEM learning Algorithm using Inverse Covariance Matrix, Loh et al. 2014:
- High-dimensional Gaussian SEM learning Algorithm using Inverse Covariance Matrix: Ghoshal and Honorio, 2017
- Linear SEM learning Algorithm using Inverse Covariance Matrix: Ghoshal and Honorio, 2018
- Gaussian SEM learning Algorithm via element-wise regressions, Park 2018, 2019.

Greedy DAG Search Algorithm, Peter and Buhlmann, 2014

1. Identifiability for Gaussian SEM with equal error variances
2. Penalized maximum likelihood estimator
3. Greedy search algorithm
4. Experiments
   - 5.1 Existing methods.
   - 5.2 Random graphs.
   - 5.3 Deviation from equal error variances.
   - 5.4 Real data.

Gaussian SEM

- Gaussian SEM:

$$X_j = \Sigma_{k \in X_{Pa(j)}} \beta_{jk} X_k + N_j \quad (j = 1, \ldots, p).$$

  - V: a set of nodes in a graph, $V = \{1, \ldots, p\}$.
  - X_j: random variable, $j \in V$.
  - N_j: noise term, $N_j \sim^{IID} N(0, \sigma^2)$ with $\sigma^2 > 0$.
  - $\beta_{jk} \neq 0$ for all $k \in X_{Pa(j)}$, otherwise $\beta_{jk} = 0$.

Theorem & Assumptions

Theorem

Let $P(\mathcal{G})$ be generated from model Gaussian SEM,
Then $\mathcal{G}$ is identifiable from $P(\mathcal{G})$ and the coefficients β_{jk} can be reconstructed for all j and $k \in X_{Pa(j)}$.

- Assumptions: **Non-zero coefficient**, **Causal sufficiency**
  - ▷ Non-zero cofficient: $\beta_{jk} \neq 0$, $k \in X_{Pa(j)}$.
  - ▷ Causal sufficiency: all variables are observed.

LEMMA A1.

Let $(A_1, \ldots, A_m) \sim N\{(\mu_1, \ldots, \mu_m)^T, \Sigma\}$ with strictly positive definite Σ, and define $A_1^* = A_1 \mid_{(A_2, \ldots, A_m) = (a_2, \ldots, a_m)}$, in distribution. Then $var(A_1^*) \leq var(A_1)$ for all $(a_2, \ldots, a_m) \in \mathbb{R}^{m-1}$.

- **Proof.** Let us decompose Σ into

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \Sigma_{12}^T \\ \Sigma_{12} & \Sigma_{22} \end{pmatrix}$$

with an $(m-1) \times 1$ vector Σ_{12}. Since Σ_{22}^{-1} is positive definite,

$$var(A_1^*) = \sigma_1^2 - \Sigma_{12}^T \Sigma_{22}^{-1} \Sigma_{12} \leq \sigma_1^2 = var(A).$$

LEMMA A3.

Let $L(\mathcal{X})$ be generated by a structural equation model as in Gaussian SEMs with corresponding directed acyclic graph $\mathcal{G}$ and consider a variable $X \in \mathcal{X}$. If $S \subseteq ND_X^{\mathcal{G}}$, then $N_X \perp\!\!\!\perp S$.

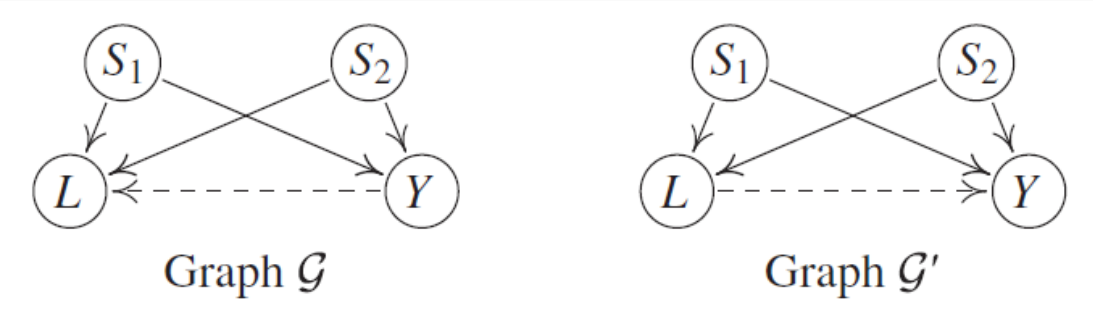
- **Example**

▷ $N_i \sim^{IID} N(0, \sigma_i^2)$, $i \in \{S, X\}$.

$$S = N_S.$$
$$X = \beta S + N_X.$$
$$\therefore\ N_X \perp\!\!\!\perp S.$$

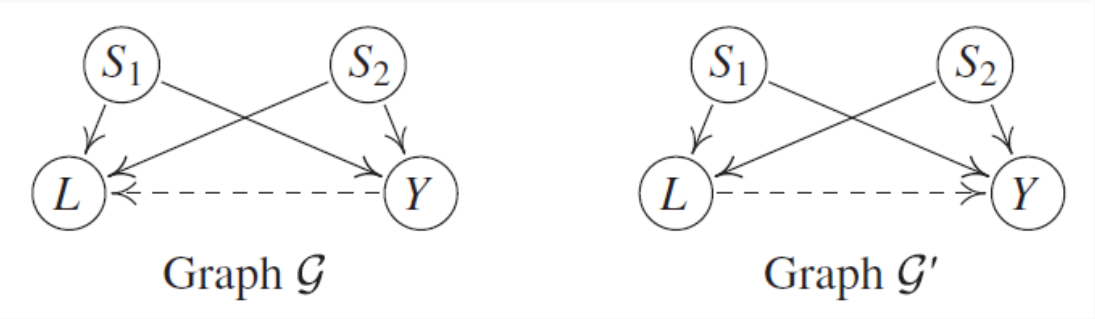
Identifiability for Gaussian SEM with equal error variances

- Suppose that there are two Gaussian SEMs with distinct graphs $\mathcal{G}$ and $\mathcal{G}'$ that lead to the same joint distribution.



$$\Rightarrow \quad " \ \mathcal{G} = \mathcal{G}' \ "$$

Proof



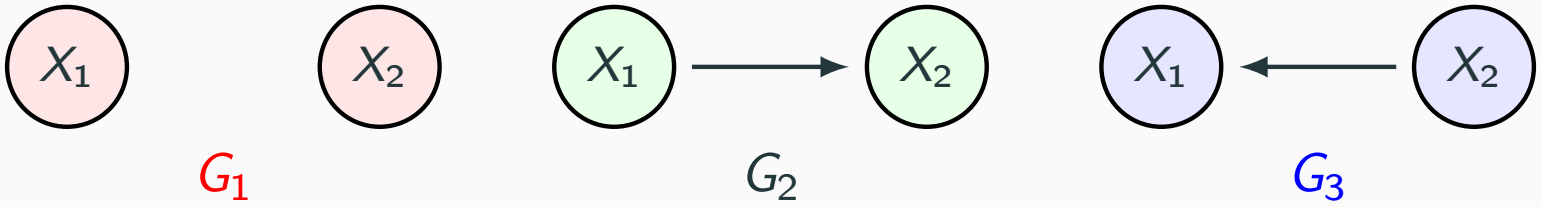
1. $\mathcal{G}$
   - $\triangleright$ $L^* = L\,|_{s_1=s_1, s_2=s_2}$, $Y^* = Y\,|_{s_1=s_1, s_2=s_2}$.
   - $\triangleright$ $L^* = \alpha_1 s_1 + \alpha_2 s_2 + \beta Y^* + N_L, N_L \sim N(0, \sigma_L^2)$.
   - $\triangleright$ $Y^* \perp\!\!\!\perp N_L$ by LEMMA A3.

   $\Rightarrow$ $Var(L^*) = \beta^2 Var(Y^*) + \sigma_L^2 > \sigma_L^2$. $\cdots(a)$
2. $\mathcal{G}'$
   - $\triangleright$ $L^* = L\,|_{s_1=s_1, s_2=s_2}$.

   $\Rightarrow$ $Var(L^*) \leq \sigma_L^2$ by LEMMA A1. $\cdots(b)$

$\therefore$ Equations (a) and (b) contradict each other.

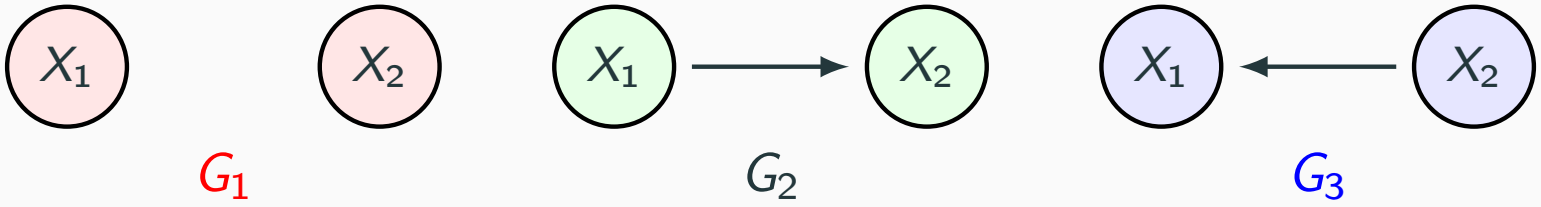
Example



- G_1: $X_1 \sim N(0, \sigma^2),\ X_2 \sim N(0, \sigma^2),\ X_1 \perp\!\!\!\perp X_2$.
- G_2: $N_1 \sim N(0, \sigma^2),\ N_2 \sim N(0, \sigma^2)$.

$$
\begin{aligned}
X_1 = N_1,\ X_2 &= \beta_1 X_1 + N_2. \\
var(X_2) &= var(\beta_1 X_1 + N_2). \\
&= var(\beta_1 X_1) + var(N_2) \text{ by LEMMA A3.} \\
&= \beta_1^2 \sigma^2 + \sigma^2. \\
&= (\beta_1^2 + 1)\sigma^2. \\
&> \sigma^2 = var(X_1). \\
&\therefore var(X_2) > var(X_1).
\end{aligned}
$$

Example



- G_3: $N_1 \sim N(0, \sigma^2)$, $N_2 \sim N(0, \sigma^2)$.

$$
\begin{aligned}
X_1 &= \beta_2 X_2 + N_1, \; X_2 = N_2. \\
var(X_1) &= var(\beta_2 X_2 + N_1). \\
&= var(\beta_2 X_2) + var(N_1) \text{ by LEMMA A3.} \\
&= \beta_2^2 \sigma^2 + \sigma^2. \\
&= (\beta_2^2 + 1)\sigma^2. \\
&> \sigma^2 = var(X_2). \\
&\therefore var(X_1) > var(X_2).
\end{aligned}
$$

Penalized maximum likelihood estimator

- Penalized maximum likelihood estimator:

$$\{\hat{\beta}(\lambda), \hat{\sigma}^{\mathbf{2}}(\lambda)\} = \arg\min_{\beta \in \mathcal{B}, \sigma^{\mathbf{2}} \in \mathbb{R}^+} -\ell\left(\beta, \sigma^{\mathbf{2}}; X^{(\mathbf{1})}, \ldots, X^{(n)}\right) + \lambda ||\beta||_{\mathbf{0}}, \qquad (1)$$

where

$$-\ell\left(\beta, \sigma^{\mathbf{2}}; X^{(\mathbf{1})}, \ldots, X^{(n)}\right) = \frac{np}{2}\log(2\pi\sigma^{\mathbf{2}}) + \frac{n}{2\sigma^{\mathbf{2}}}\text{tr}\{(I-B)^T(I-B)\hat{\Sigma}\}.$$

  - ▷ B: p$\times$ p matrix with $B_{jk} = \beta_{jk}$.
  - ▷ $\hat{\Sigma}$: sample covariance matrix.
  - ▷ σ^2: error variance.
  - ▷ $\lambda = \log(n)/2$: the objective function in equation (1) is the BIC score.

Convergence rate & Consistency

- Convergence rate:
  For $\lambda_n = \log(n)/2$, $n \to \infty$,

$$\Sigma_{j,k=1}^{p} \{\hat{\beta}_{jk}(\lambda_n) - \beta_{jk}\}^2 = O_p\{\log(n) n^{-1}\}.$$

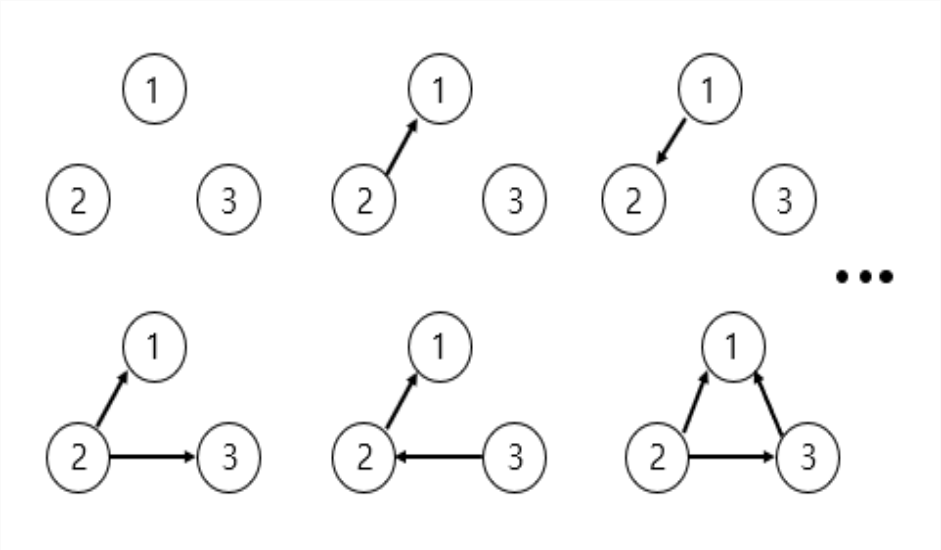
- Consistency:
  For $\lambda_n = \log(n)/2$, $n \to \infty$,

$$\text{pr}(\hat{\mathcal{G}}_n = \mathcal{G}) \to 1.$$

* Reference: Van de Geer & B̈uhlmann (2013),
$\ell_{\mathbf{0}}$-penalized maximum likelihood for sparse DAG. (Theorem 5.1)

Computational complexity

- Because the optimization is over the space of all DAG, the estimator is hard to compute.
  - ▷ p=3



  - ▷ p=20: $2 \cdot 3 \times 10^{72}$ DAG $\mathcal{G}$

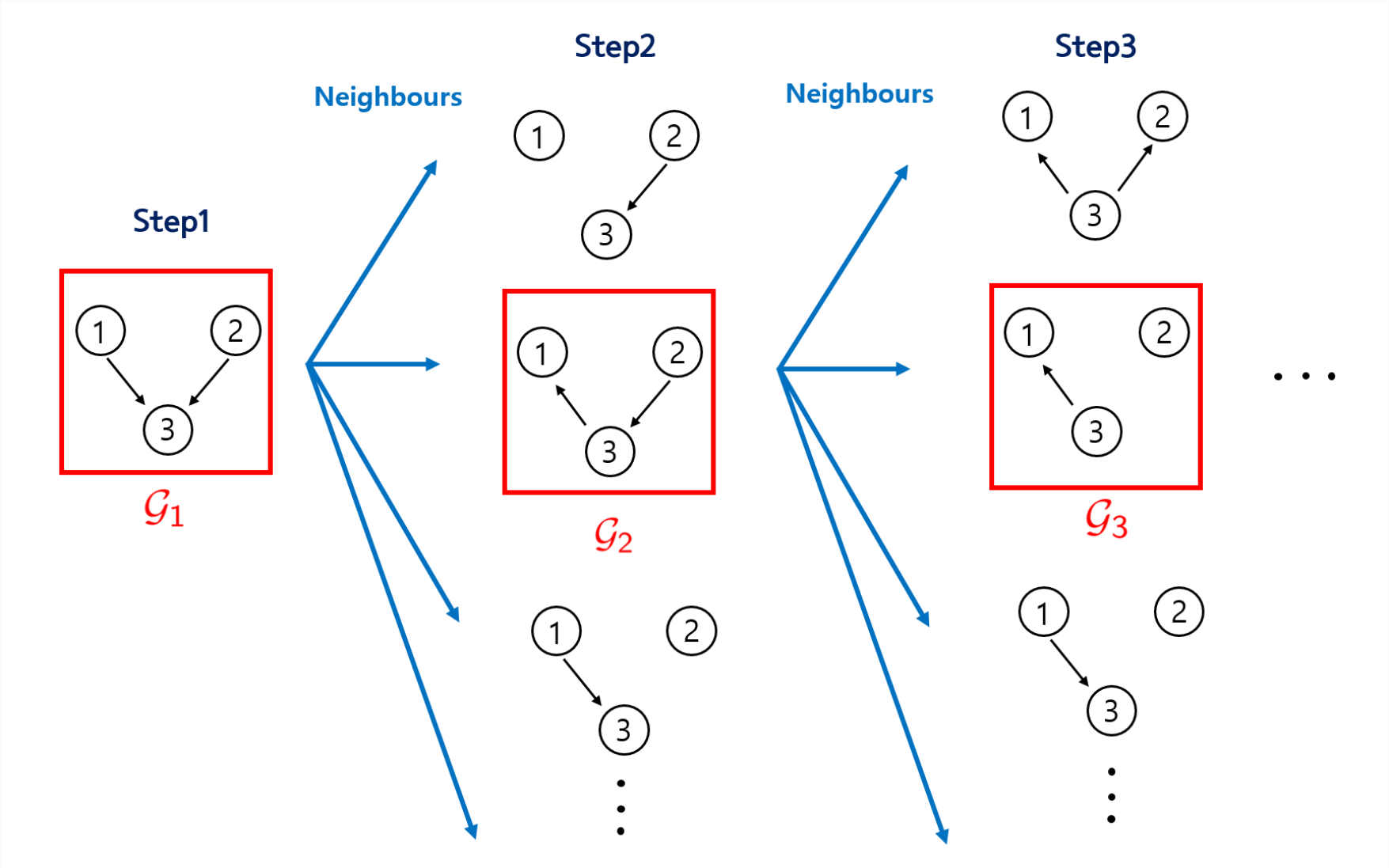
$\Rightarrow$ **Greedy search algorithm**

Greedy Directed Acyclic Graph Search(GDS)

- Initial graph: Empty graph or Random graph.
- We consider at least k neighbours.
- If the neighbouring $\mathcal{G}_t$ have a lower BIC score than $\mathcal{G}_{t-1}$, $\mathcal{G}_{t-1}$ move to $\mathcal{G}_t$.
- If all neighbours have a higher BIC score than $\mathcal{G}_t$, the algorithm terminates.
- $k = p,\ k = 2p,\ k = 3p,\ k = 5p$ and $k = 300$.

* Neighbour: if they can be transformed into each other by one edge addition, removal or reversal.

Greedy Directed Acyclic Graph Search(GDS)



Compare three methods

- Compare three methods:
  PC-algorithm, GES, **GDS**.
- Evaluation:
  The structural Hamming distance between the true and estimated DAG.

* Hamming distance: this assigns a distance of 2 for each pair of reversed edges. all other edge mistakes count as 1.

Random graphs

- p=5, 20, 40.
- n=100, 500, 1000.
- β_{jk}: uniformly $[-1, -0.1] \cup [0.1, 1]$.
- Sparse setting: 3p/4 edges.
- Dense setting: $3p(p-1)/20$ edges.

Sparse setting

- Sparse setting

		$n = 100$			$n = 500$			$n = 1000$		
p		GDS_{EEV}	PC	GES	GDS_{EEV}	PC	GES	GDS_{EEV}	PC	GES
5	DAG	1·5	3·9	3·6	0·5	2·9	2·8	0·4	3·0	2·5
	CPDAG	1·5	2·9	2·3	0·5	1·4	1·2	0·3	1·0	0·7
20	DAG	12·2	14·1	18·0	4·5	11·1	10·3	2·7	10·1	8·7
	CPDAG	13·9	10·9	17·0	5·2	7·7	7·6	3·0	6·9	5·6
40	DAG	44·7	29·6	53·0	15·7	22·6	26·1	10·7	20·1	21·9
	CPDAG	50·0	24·4	53·1	18·9	15·9	23·4	13·4	13·3	17·5

Average structural Hamming distance

  - ▷ Except for $p = 40$ and $n = 100$,
    GDS method are closer to the true DAG.

Dense setting

- Dense setting

		$n = 100$			$n = 500$			$n = 1000$		
p		GDS_{EEV}	PC	GES	GDS_{EEV}	PC	GES	GDS_{EEV}	PC	GES
5	DAG	1·2	2·9	3·0	0·6	2·4	2·2	0·3	2·1	2·1
	CPDAG	1·3	2·1	1·9	0·5	1·2	0·7	0·2	0·8	0·5
20	DAG	30·0	56·6	63·9	12·5	55·7	66·3	8·2	57·6	69·1
	CPDAG	31·0	56·1	63·2	13·1	55·5	66·2	8·8	57·5	68·5
40	DAG	216·1	242·8	323·1	185·2	247·2	430·4	172·0	248·9	470·6
	CPDAG	217·1	242·4	323·0	185·7	247·0	430·1	172·2	248·5	470·4

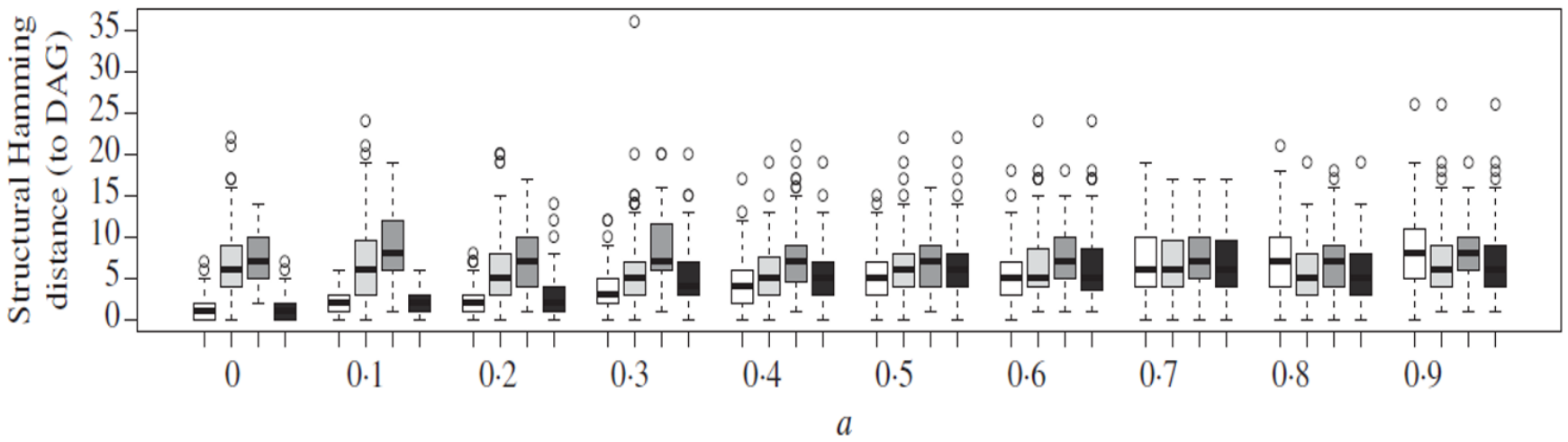
Average structural Hamming distance

  - ▷ GDS method are closer to the true DAG.

Deviation from equal error variances

- $p = 10$.
- $n = 500$.
- β_{jk}^0: uniformly $[-1, \ -0.1] \cup [0.1, \ 1]$.
- Setting: p edges (between the sparse and dense settings).
- σ_j^2: uniformly $[1 - a, \ 1 + a], \ a = (0.1, 0.2, \cdots, 0.9)$.

Deviation from equal error variances



1: GDS, 2: GES, 3: PC, 4: Best

- For large values of a,
  the method does not perform worse than the PC-algorithm.
- GDS is relatively robust as the parameter a changes.

* Bestscore method: the result of GDS or GES depending on which method obtained the better score.

Ghoshal and Honorio, 2017

Ghoshal and Honorio, 2017

- SangJun.

Ghoshal and Honorio, 2018

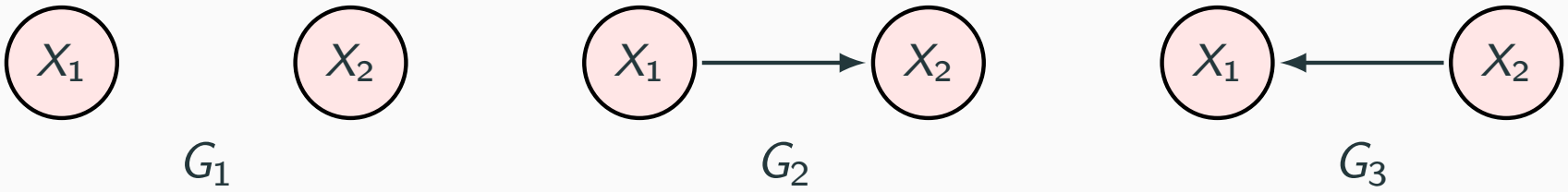
Ghoshal and Honorio, 2018

- GW

Gaussian Linear Structural Equation Model, Park 2019

Model Identifiability

Is it possible to recover a graph from data? **Partially Yes.**



- We can distinguish G_2 and G_3 from G_1 using dependence.
- We cannot identify the direction of an edge. Hence, we cannot distinguish G_2 from G_3.

Gaussian Linear SEM

- Form of Gaussian Linear SEMs:
  - ▷ Node-wide: $X_j = \sum_{k \in \mathrm{Pa}(j)} \beta_{kj} X_k + \epsilon_j \quad , \forall j = 1, \ldots, p,$
  - ▷ Matrix:

$$(X_1, X_2, ..., X_p)^T = B_0 + B(X_1, ..., X_p)^T + (\epsilon_1, ..., \epsilon_p)^T$$

- Assumptions:
  - ▷ Gaussian Noise: $\epsilon_j \sim_{iid} N(0, \sigma_j^2)$ with $\sigma_j^2 > 0$.
  - ▷ Causal Minimality: $\beta_{kj} \neq 0$ for all $k \in \mathrm{Pa}(j)$, otherwise $\beta_{jk} = 0$.
  - ▷ Causal Sufficiency: all variables are observed.

- Joint Distribution:

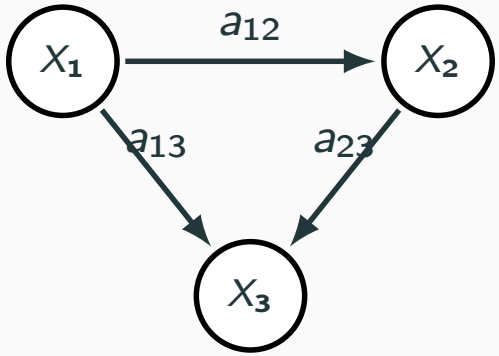
$$f(G) = \frac{1}{\sqrt{(2\pi)^p \det(\Theta^{-1})}} \exp\Big(-\frac{1}{2}(x_1, ..., x_p)\Theta(x_1, ..., x_p)^T \Big).$$

where $\Theta = (I_p - B)^T \Sigma_\epsilon^{-1} (I_p - B) \succ 0$ and Σ_ϵ is a $\mathrm{diag}(\sigma_1^2, ..., \sigma_p^2)$.

λ-strong faithfulness assumption

- Under the faithfulness assumption, the Markov equivalence class can be recovered.
- In finite sample settings, Geometry of λ-strong faithfulness assumption
  - ▷ partial correlation hypersurfaces
  - ▷ fixed $\lambda > 0$: lower bounds on tube volumes

3-node Linear structural equation model



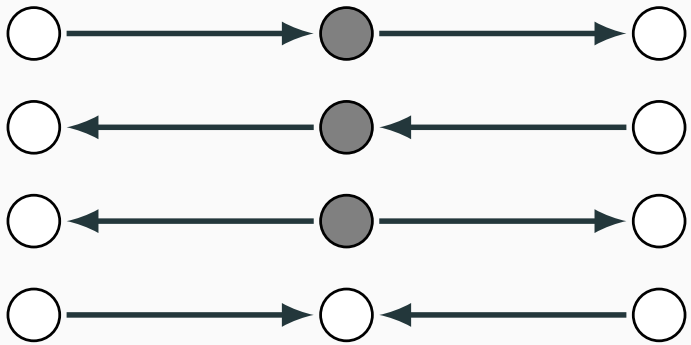
$$X_1 = \epsilon_1$$
$$X_2 = a_{12}X_1 + \epsilon_2$$
$$X_3 = a_{13}X_1 + a_{23}X_2 + \epsilon_3$$

- $\epsilon \sim \mathbb{N}(0, D)$ indep. noise $\rightarrow X \sim \mathbb{N}(0, (I - A)D^{-1}(I - A^T))$

Partial correlation and D-separation

Definition

i is d-separated from j given $S \subset V \setminus \{i, j\}$ if every path between i and j is blocked by S



- If $(i, j) \notin E$ then there exists $S \subset V \setminus \{i, j\}$ such that i is d-separated from j given S
- In the Gaussian setting: $| \, corr(i, j \mid S) \, | = | \, \frac{det(K_{iR,jR})}{\sqrt{det(K_{iR,iR}) \cdot det(K_{jR,jR})}} \, |$
- $det(K_{iR,jR})$ is linear combination of all non-blocked paths from i to j
- $corr(i, j \mid S) \equiv 0 \Leftrightarrow i$ is d-separated from j given S

Strong-faithfulness

Definition

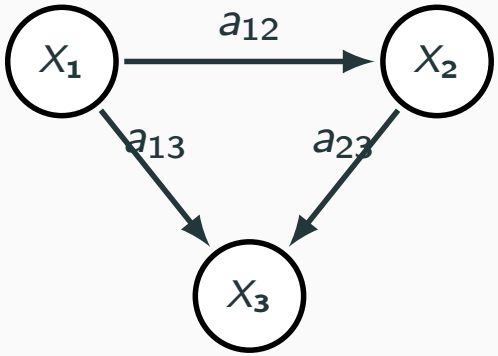
A multivariate Gaussian distribution P is λ-strongly faithful to a DAG G for $\lambda \in (0,1)$ if for any $j, k \in V$ and any $S \subset V \setminus \{j,k\}$:

$$| \, corr(j,k) \, | \leq \lambda \iff j \text{ is d-separated from } k \text{ given } S.$$

Question: What is the proportion of distributions that don't satisfy the strong-faithfulness condition?

- $Tube_{corr(i,j|S)}(\lambda) = \{(a_{ij}) \in [-1,1]^E \ : \ | \, corr(i,j \mid S) \, | \leq \lambda\}$
- $Vol_{corr(i,j|S)}(\lambda) = \int_{Tube_{corr(i,j|S)}(\lambda)} \phi(\omega) d\omega$

Unfaithful distributions: 3-node example



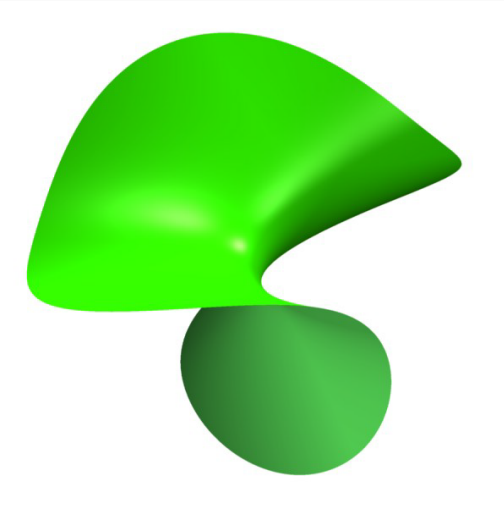
$$\Sigma^{-1} = \begin{pmatrix} 1 + a_{12}^2 + a_{13}^2 & a_{13}a_{23} - a_{12} & -a_{13} \\ a_{13}a_{23} - a_{12} & 1 + a_{23}^2 & -a_{23} \\ -a_{13} & -a_{23} & 1 \end{pmatrix}$$

Faithfulness is **NOT** satisdied if any of the following relations hold:

- $X_1 \perp\!\!\!\perp X_2 \iff a_{12} = 0$
- $X_1 \perp\!\!\!\perp X_3 \iff a_{13} + a_{12}a_{23} = 0$
- $X_2 \perp\!\!\!\perp X_3 \iff a_{12}^2 a_{23} + a_{12}a_{13} + a_{23} = 0$
- $X_1 \perp\!\!\!\perp X_2 \mid X_3 \iff a_{13}a_{23} - a_{12} = 0$
- $X_1 \perp\!\!\!\perp X_3 \mid X_2 \iff -a_{13} = 0$
- $X_2 \perp\!\!\!\perp X_3 \mid X_1 \iff -a_{23} = 0$

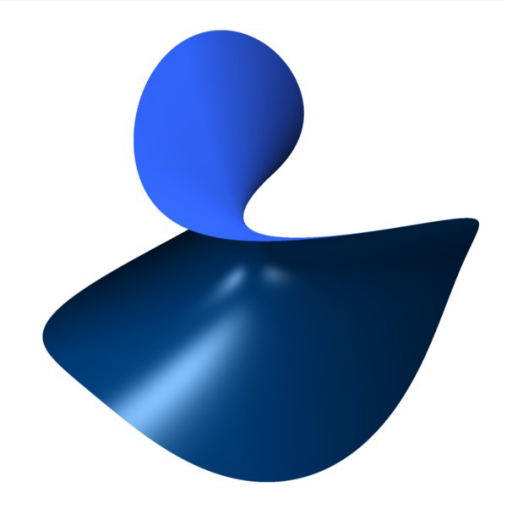
$\Rightarrow$ Faithfulness not satisfied on collection of hypersurfaces in $\mathbb{R}^{|E|}$

Unfaithful distributions: 3-node example



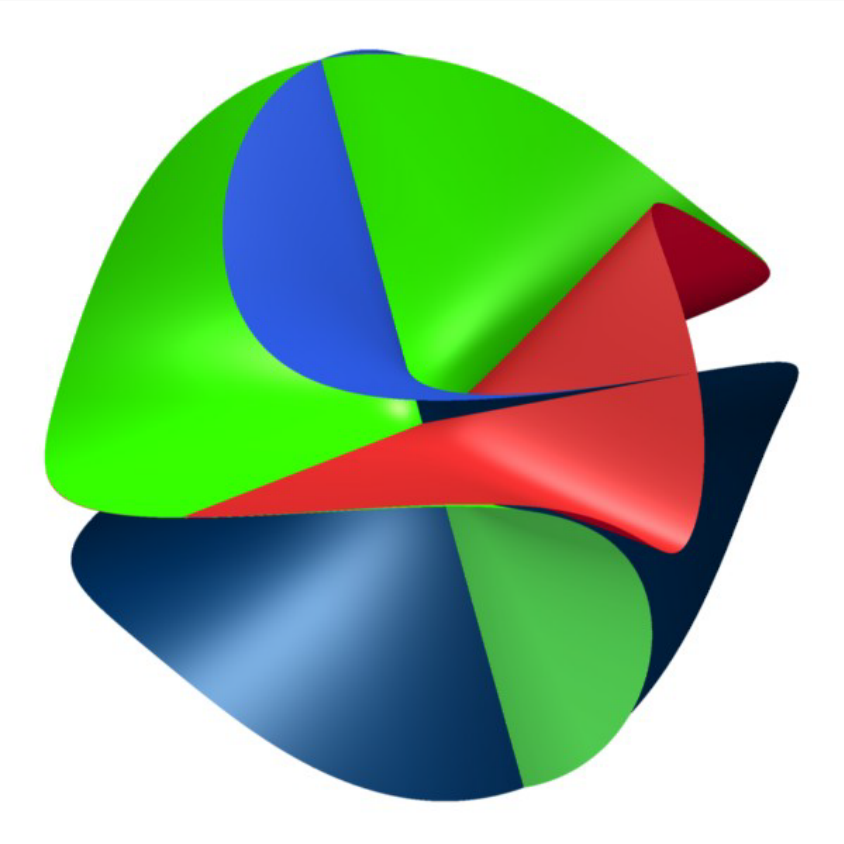
$a_{13} + a_{12}a_{23} = 0$

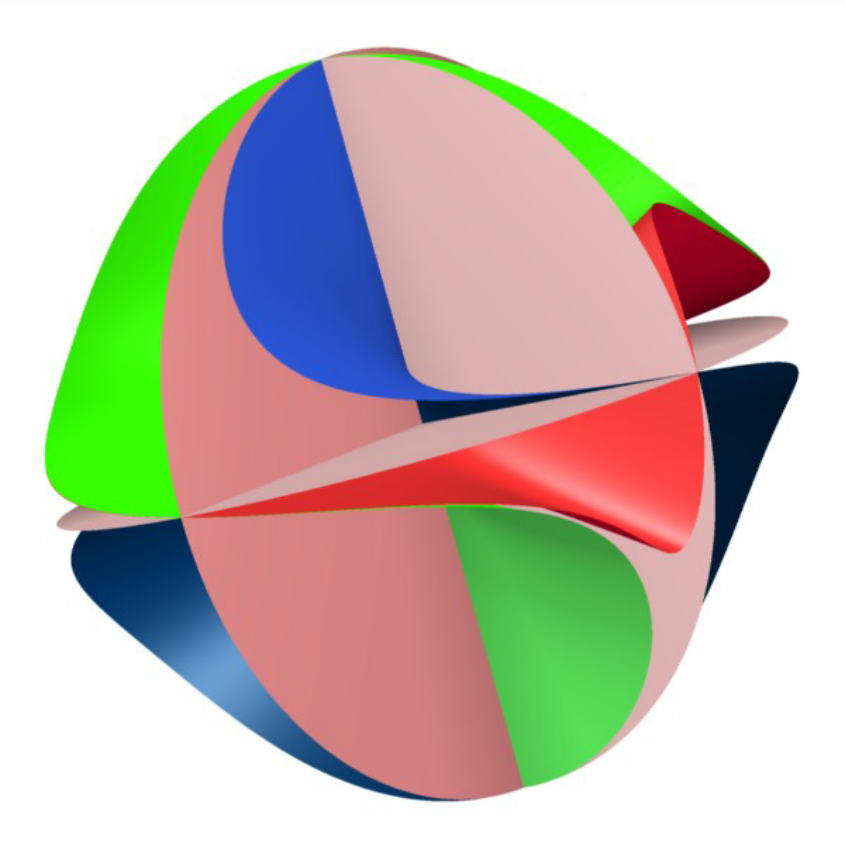
$a_{12}^2 a_{23} + a_{12}a_{13} + a_{23} = 0$



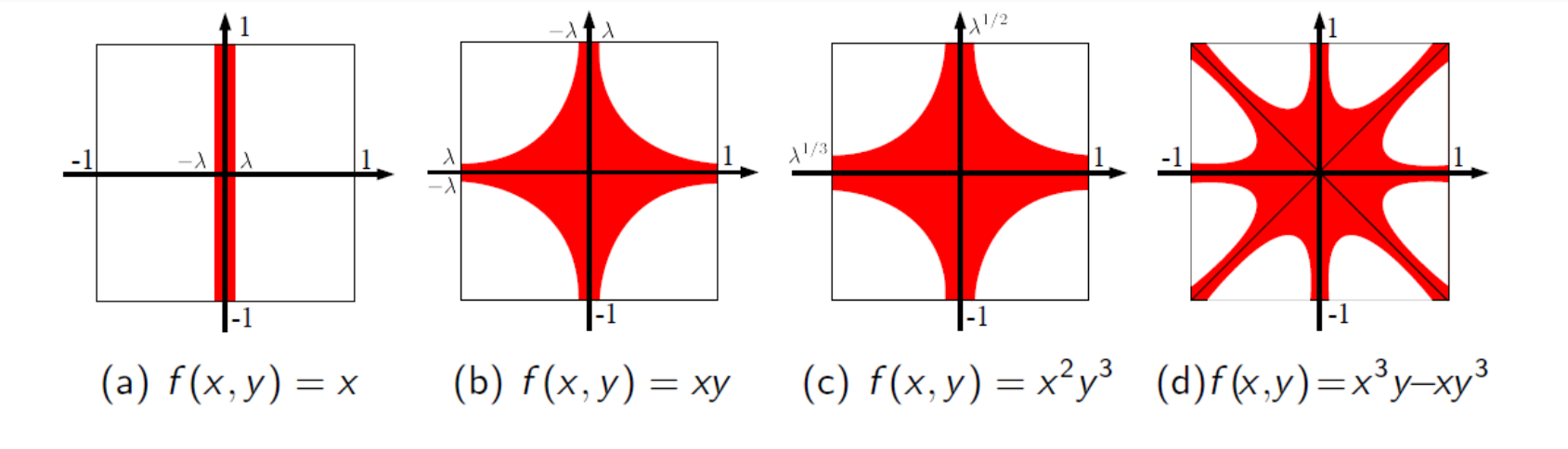
$a_{13}a_{23} - a_{12} = 0$

Unfaithful distributions: 3-node example





Main question

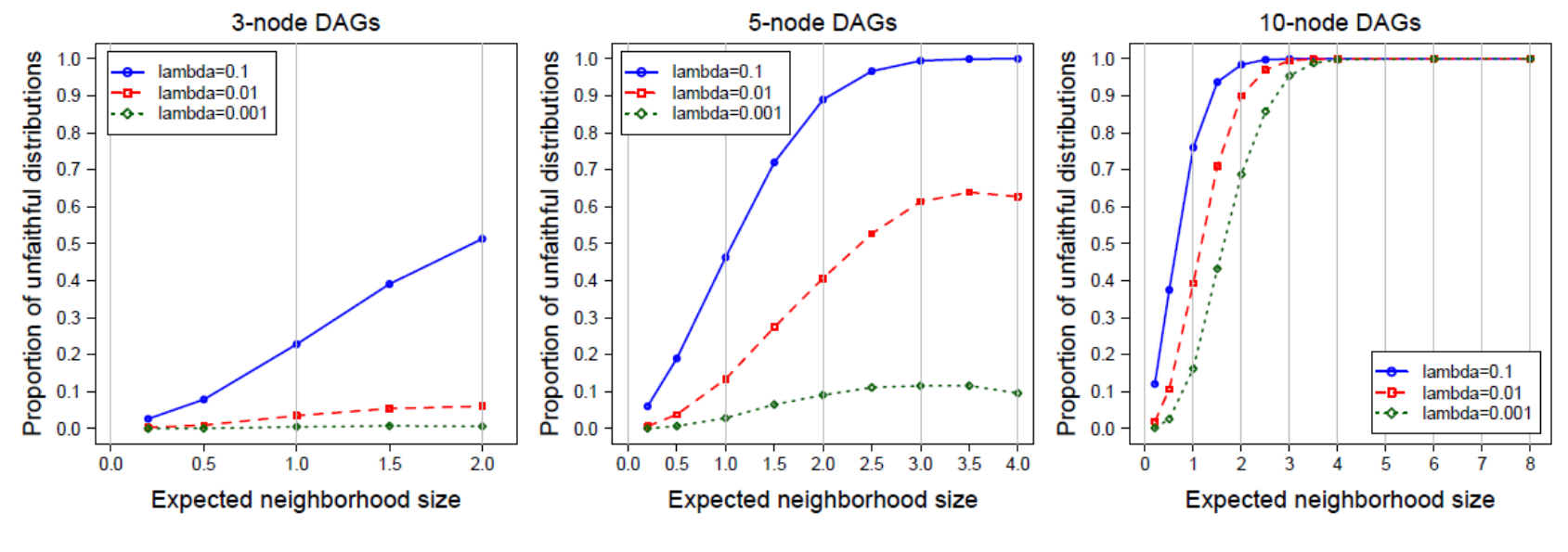


(a) $f(x,y) = x$ (b) $f(x,y) = xy$ (c) $f(x,y) = x^2y^3$ (d) $f(x,y) = x^3y - xy^3$

Problem: Given a DAG G, what proportion of distributions does not satisfy strong-faithfulness?

- $\mathcal{P}^{\lambda}_{j,k|S} = \{(a_{st}) \in [-1,+1]^{|E|} :| \; corr(X_j, X_k \mid X_S) \; | \leq \lambda\}$
- $\mathcal{M}_{G,\lambda} = \cup_{j,k|S \;\; not \;\; \mathbf{d}-separated} \mathcal{P}^{\lambda}_{j,k|S}$

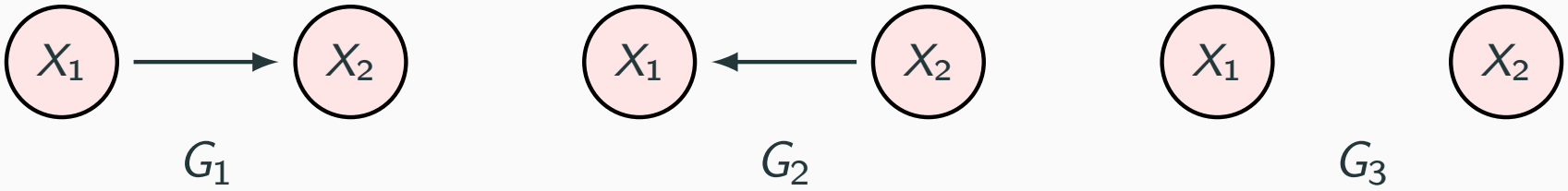
lambda-strong faithfulness for DAGs



- Proportion of strongly unfaithful distributions increases with λ ($\asymp 1/\sqrt{n}$)
- Proportion of strongly unfaithful distributions increases with graph density and size

Motivations of Identifiability: Ordering Recovery

Is it possible to recover a graph from a Gaussian SEM? Yes.



For G_1 where $X_1 = \epsilon_1$, $X_2 = \beta_1 X_1 + \epsilon_2$, and $\epsilon_j \sim N(0, \sigma_j^2)$

- Forward Selection:

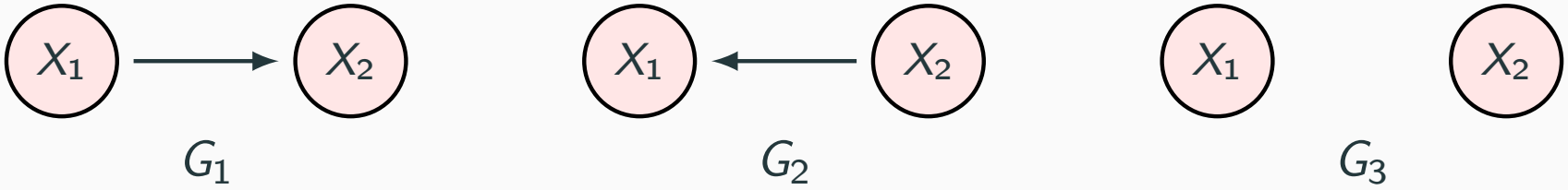
$$\text{Var}(X_2) = \mathbb{E}(\text{Var}(X_2 \mid X_1)) + \text{Var}(\mathbb{E}(X_2 \mid X_1)) = \sigma_2^2 + \beta_1^2 \sigma_1^2 > \sigma_1^2 = \text{Var}(X_1),$$

- Backward Elimination:

$$\mathbb{E}(\text{Var}(X_1 \mid X_2)) = \text{Var}(X_1) - \text{Var}(\mathbb{E}(X_1 \mid X_2))$$

$$= \sigma_1^2 - \frac{\beta_1^2 \sigma_1^4}{\beta_1^2 \sigma_1^2 + \sigma_2^2} < \sigma_2^2 = \mathbb{E}(\text{Var}(X_2 \mid X_1)).$$

Motivations of Identifiability: Parent Recovery



Causal Minimality

A joint distribution is not Markov with respect to a strict sub-graph of the true graph. In the linear SEM settings, it is equivalent to the following for any node $j \in V$ and one of its parents $k \in \mathrm{Pa}(j)$:

$$X_j \not\perp\!\!\!\perp X_k \mid X_S, \quad \forall \ \mathrm{Pa}(j) \setminus \{k\} \subset S \subset \mathrm{Nd}(j) \setminus \{k\}.$$

Suppose that $\sigma_1^2 < \sigma_2^2$. Then, for G_3,

$$\mathrm{Var}(X_2) = \sigma_2^2 > \sigma_1^2 = \mathrm{Var}(X_1).$$

Under the minimality condition,

$$G_1 : X_1 \not\perp\!\!\!\perp X_2 \quad \text{and} \quad G_3 : X_1 \perp\!\!\!\perp X_2$$

Identifiability for 3-node Chain Graph

$$X_1 = \epsilon_1, \quad X_2 = \beta_1 X_1 + \epsilon_2, \quad X_3 = \beta_2 X_2 + \epsilon_3$$

where $\epsilon_j \sim N(0, \sigma_j^2)$ for all $j \in \{1, 2, 3\}$.

Ordering Estimation: For π_1,

$$\mathrm{Var}(X_2) = \mathbb{E}(\mathrm{Var}(X_2 \mid X_1)) + \mathrm{Var}(\mathbb{E}(X_2 \mid X_1)) = \sigma_2^2 + \beta_1^2 \sigma_1^2 > \sigma_1^2 = \mathrm{Var}(X_1)$$
$$\mathrm{Var}(X_3) = \mathbb{E}(\mathrm{Var}(X_3 \mid X_2)) + \mathrm{Var}(\mathbb{E}(X_3 \mid X_2)) = \sigma_3^2 + \beta_2^2 \sigma_2^2 + \beta_2^2 \beta_1^2 \sigma_1^2 > \sigma_1^2 = \mathrm{Var}(X_1).$$

as long as $\sigma_2^2/\sigma_1^2 > (1 - \beta_1^2)$ and $\sigma_3^2/\sigma_1^2 > (1 - \beta_2^2)$.

Identifiability for Three-node Chain Graph

Ordering Estimation: For π_2,

$$\begin{aligned}\mathbb{E}(\mathrm{Var}(X_3 \mid X_1)) &= \mathbb{E}(\mathbb{E}(\mathrm{Var}(X_3 \mid X_2) \mid X_1)) + \mathbb{E}(\mathrm{Var}(\mathbb{E}(X_3 \mid X_2) \mid X_1)) \\ &= \sigma_3^2 + \beta_2^2\sigma_2^2 > \sigma_2^2 = \mathbb{E}(\mathrm{Var}(X_2 \mid X_1)).\end{aligned}$$

as long as $\sigma_3^2/\sigma_2^2 > (1 - \beta_2^2)$.

Parent Estimation: Under the minimality and the Markov condition,

$$X_1 \not\perp\!\!\!\perp X_2, \quad X_1 \perp\!\!\!\perp X_3 \mid X_2, \quad X_2 \not\perp\!\!\!\perp X_3 \mid X_1$$

Theorem: Identifiability

Let $P(G)$ be generated from a Gaussian linear SEM.

- Π_G is a set of true orderings of graph G.
- $X_{1,2,..,j} = \{X_{\pi_1}, X_{\pi_2}, ..., X_{\pi_j}\}$.

Theorem

The DAG G is uniquely identifiable, if there exists $\pi \in \Pi_G$ satisfying either of the two following conditions: For any node $m \in V$, let $j = \pi_m$, $k \in \mathrm{De}(j)$, and $\ell \in \mathrm{An}(j)$,

(A) Forward Selection:

$$\sigma_j^2 < \sigma_k^2 + \mathbb{E}(\mathrm{Var}(\mathbb{E}(X_k \mid X_{\mathrm{Pa}(k)}) \mid X_{1:(j-1)}))$$

(B) Backward Elimination:

$$\sigma_j^2 > \sigma_\ell^2 + \mathbb{E}(\mathrm{Var}(\mathbb{E}(X_\ell \mid X_{1:(j-1)}) \mid X_{\mathrm{Pa}(\ell)})).$$

Prior Identifiability Assumptions for Gaussian SEMs

Identifiability Assumption (Ghoshal and Horino, 2018)

Let $P(X)$ be generated from a Gaussian linear SEM with directed acyclic graph G. For any $m \in \{1, 2, ..., p\}$, let $j = \pi_m$ and $\ell \in \mathrm{De}(j)$.

$$\frac{1}{\sigma_j^2} < \frac{1}{\sigma_\ell^2} + \sum_{k \in \mathrm{Ch}(\ell)} \frac{\beta_{\ell k}^2}{\sigma_k^2}.$$

- The main idea of the proof is based on the property of precision matrix.
- It allows heterogeneous error variances.

Sketch of the Proof I

- The precision matrix can be expressed as:

$$\Theta = (I_p - B)^T \Sigma_\epsilon^{-1} (I_p - B)$$

$$= \begin{bmatrix} 1 & 0 & \dots & 0 \\ -\beta_{12} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ -\beta_{1p} & -\beta_{2p} & \dots & 1 \end{bmatrix}^T \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \sigma_p^2 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & \dots & 0 \\ -\beta_{12} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & 0 \\ -\beta_{1p} & -\beta_{2p} & \dots & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{\sigma_1^2} + \sum_{j \in \mathrm{Ch}(1)} \frac{\beta_{1j}^2}{\sigma_j^2} & \dots & \dots & \dots \\ \dots & \frac{1}{\sigma_2^2} + \sum_{j \in \mathrm{Ch}(2)} \frac{\beta_{2j}^2}{\sigma_j^2} & \dots & \dots \\ \vdots & \vdots & \vdots & 0 \\ \dots & \dots & \dots & \frac{1}{\sigma_p^2} \end{bmatrix}$$

- Hence we can choose the last element of the ordering.

Sketch of the Proof II

- For the next element of the ordering, $(\Sigma_{-p,-p})^{-1}$ is required:

$$\Theta^{'} = (I_{p-1} - B_{-p,-p})^T \Sigma_{\epsilon}^{-1} (I_{p-1} - B_{-p,-p})$$

$$= \begin{bmatrix} 1 & 0 & \dots & 0 \\ -\beta_{12} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ -\beta_{1p-1} & -\beta_{2p-1} & \dots & 1 \end{bmatrix}^T \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \sigma_{p-1}^2 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & \dots & 0 \\ -\beta_{12} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & 0 \\ -\beta_{1p-1} & -\beta_{2p-1} & \dots & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{\sigma_1^2} + \sum_{j \in Ch(1) \setminus p} \frac{\beta_{1j}^2}{\sigma_j^2} & \dots & \dots & \dots \\ \dots & \frac{1}{\sigma_2^2} + \sum_{j \in Ch(2) \setminus p} \frac{\beta_{2j}^2}{\sigma_j^2} & \dots & \dots \\ \vdots & \vdots & \vdots & 0 \\ \dots & \dots & \dots & \frac{1}{\sigma_{p-1}^2} \end{bmatrix}$$

How to update the precision matrix?

- It is much faster to use a precision matrix for the update.

$$\Theta = (I_p - B)^T \Sigma_\epsilon^{-1} (I_p - B)$$

$$= \begin{bmatrix} 1 & 0 & \dots & 0 \\ -\beta_{12} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ -\beta_{1p} & -\beta_{2p} & \dots & 1 \end{bmatrix}^T \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \sigma_p^2 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & \dots & 0 \\ -\beta_{12} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & 0 \\ -\beta_{1p} & -\beta_{2p} & \dots & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \dots & -\frac{\beta_{12}}{\sigma_2^2} + \sum_{j \in 3:p} \frac{\beta_{1j}\beta_{2j}}{\sigma_j^2} & -\frac{\beta_{13}}{\sigma_3^2} + \sum_{j \in 4:p} \frac{\beta_{1j}\beta_{3j}}{\sigma_j^2} & \dots \\ \dots & \dots & -\frac{\beta_{23}}{\sigma_3^2} + \sum_{j \in 4:p} \frac{\beta_{2j}\beta_{3j}}{\sigma_j^2} & \dots \\ \vdots & \vdots & \vdots & 0 \\ \dots & -\frac{\beta_{2p}}{\sigma_p^2} & -\frac{\beta_{3p}}{\sigma_p^2} & \frac{1}{\sigma_p^2} \end{bmatrix}$$

How to update the precision matrix?

$$
\begin{aligned}
\Theta^{'} &= (I_{p-1} - B_{-p,-p})^T \Sigma_{\epsilon}^{-1} (I_{p-1} - B_{-p,-p}) \\
&= \Theta_{-p,-p} - \frac{1}{\sigma_p^2} (B_{,-p})^T (B_{,-p}) \\
&= \Theta_{-p,-p} - \Theta_{p,p} (B_{,-p})^T (B_{,-p}) \\
&= \Theta_{-p,-p} - \Theta_{p,p} \left(\frac{\Theta_{p,-p}}{\Theta_{p,p}} \right)^T \left(\frac{\Theta_{p,-p}}{\Theta_{p,p}} \right)
\end{aligned}
$$

Identifiability for Linear SEMs

Theorem

The DAG G is uniquely identifiable, if there exists $\pi \in \Pi_G$ satisfying either of the two following conditions: For any node $m \in V$, let $j = \pi_m$, $k \in \mathrm{De}(j)$, and $\ell \in \mathrm{An}(j)$,

(A) Forward Selection:

$$\sigma_j^2 < \sigma_k^2 + \sum_{k' \in \mathrm{Pa}(k) \setminus 1:(j-1)} \beta_{k' \to k}^2 \sigma_{k'}^2.$$

where $\beta_{k' \to k}$ is a sum of all paths from k' to k, that is,

$$\begin{aligned}
\beta_{k' \to k} &= \beta_{k',k'+u} && \cdots \text{ the path of length 1} \\
&+ \beta_{k',k'+1}\beta_{k'+1,k'+u} + \beta_{k',k+2}\beta_{k'+2,k'+u} + \cdots + \beta_{k',k'+u-1}\beta_{k'+u-1,k'+u} && \cdots \text{ all paths of length 2} \\
&+ \beta_{k',k'+1}\beta_{k'+1,k'+2}\beta_{k'+2,k'+u} + \cdots + \beta_{k',k'+1}\beta_{k'+1,k'+2}\beta_{k'+u-2,k'+u} && \cdots \text{ all paths of length 3} \\
&+ \cdots && \vdots \\
&+ \beta_{k',k'+1}\beta_{k'+1,k'+2}\beta_{k'+u-1,k'+u} && \cdots \text{ the path of length u}
\end{aligned}$$

where $k = k' + u$.

Identifiability for Linear SEMs

Theorem

The DAG G is uniquely identifiable, if there exists $\pi \in \Pi_G$ satisfying either of the two following conditions: For any node $m \in V$, let $j = \pi_m$, $k \in \mathrm{De}(j)$, and $\ell \in \mathrm{An}(j)$,

(B) Backward Elimination:

$$\frac{1}{\sigma_j^2} < \frac{1}{\sigma_\ell^2} + \sum_{k' \in \mathrm{Ch}(\ell)} \frac{\beta_{\ell k'}^2}{\sigma_{k'}^2}.$$

Sketch of the Proof for Forward Selection I

Using the inverse matrix property,

$$\begin{aligned}[(\Sigma_{S\cup j,S\cup j})^{-1}]_{jj} &= \Sigma_{jj} - \Sigma_{jS}\Sigma_{SS}^{-1}\Sigma_{Sj} = \mathbb{E}(\mathrm{Var}(X_j \mid X_S)) \\ &= \sigma_j^2 + \mathbb{E}(\mathrm{Var}(\mathbb{E}(X_j \mid X_{\mathrm{Pa}(j)}) \mid X_S)).\end{aligned}$$

The covariance matrix is can be expressed as:

$$\Sigma = (I - B)^{-T}\Sigma_\epsilon(I - B)^{-1}.$$

Using the Neumann power series, we obtain

$$(I - B)^{-1} = (I + B + B^2 + B^3 + ... + B^p).$$

Then, the covariance matrix is:

$$\Sigma = (I-B)^{-T}\Sigma_\epsilon(I-B)^{-1} = (I+A+A^2+...+A^p)\Sigma_\epsilon(I+B+B^2+...+B^p)$$

where A is a transpose of B.

Sketch of the Proof for Forward Selection II

For ease of notation, we define

$$D^T = \Sigma_\epsilon^{1/2}(1 + B + B^2 + ... + B^p)$$

Using the path interpretation, $[D]_{jk}$ corresponds to sum of all directed paths from j to k.

The matrix D can be partitioned into four blocks, it can be inverted blockwise as follows:

$$D = \begin{bmatrix} D_1 & 0 \\ D_3 & D_4 \end{bmatrix}$$

Sketch of the Proof for Forward Selection III

Then, the inverse matrix of partial covariance matrix is as follows:

$$[(\Sigma_{S\cup j,S\cup j})^{-1}]_{j,j} = \left(D_3 {D_3}^T + D_4 {D_4}^T - D_3 D_1 (D_1 D_1^T)^{-1} {D_1}^T {D_3}^T\right)^{-1}$$

Since D_1 is a lower triangular matrix, we have

$$[(\Sigma_{S\cup j,S\cup j})^{-1}]_{j,j} = \left(D_3 {D_3}^T + D_4 {D_4}^T - D_3 {D_3}^T\right)^{-1} = (D_4 {D_4}^T)^{-1}.$$

It can be re-written using the edge weight and error variances as follows:

$$\mathrm{Var}(X_j \mid X_S) = (D_4 {D_4}^T) = \sigma_j^2 + \sum_{k \in \mathrm{Pa}(j)\setminus S} \beta_{k\to j}^2 \sigma_k^2.$$

Sketch of the Proof for Backward Elimination I

The diagonal entry of the precision matrix is

$$\Omega_{jj} = \frac{1}{\sigma_j^2} + \sum_{\ell \in \mathsf{Ch}(j)} \frac{\beta_{j\ell}^2}{\sigma_\ell^2}.$$

The precision matrix can be expressed as: For any node $j \in V$ and $S = V \setminus j$,

$$\begin{aligned}(\Omega_{jj})^{-1} &= \Sigma_{jj} - \Sigma_{jS}\Sigma_{SS}^{-1}\Sigma_{Sj} \\ &= \mathbb{E}(\mathsf{Var}(X_j \mid X_S)) = \sigma_j^2 + \mathbb{E}(\mathsf{Var}(\mathbb{E}(X_j \mid X_S) \mid X_{\mathsf{Pa}(j)})).\end{aligned}$$

Therefore, for any terminal node j and non-terminal node in the ordering ℓ,

$$\begin{aligned}&\sigma_j^2 > \sigma_\ell^2 + \mathbb{E}(\mathsf{Var}(\mathbb{E}(X_\ell \mid X_S) \mid X_{\mathsf{Pa}(\ell)})) \\ \iff \Omega_{jj} < \Omega_{\ell\ell} &\iff \frac{1}{\sigma_j^2} < \frac{1}{\sigma_\ell^2} + \sum_{k \in \mathsf{Ch}(\ell)} \frac{\beta_{\ell k}^2}{\sigma_k^2}\end{aligned}$$

Relationship between (A) and (B)

$$X_1 = \epsilon_1, \quad X_2 = \beta_1 X_1 + \epsilon_2, \quad X_3 = \beta_2 X_2 + \epsilon_3, \quad \text{where } \epsilon_j \sim N(0, \sigma_j^2).$$

Assumption (A):

$$(i)\ \ \sigma_1^2 < \sigma_2^2 + \beta_1^2\sigma_1^2, \quad (ii)\ \ \sigma_2^2 < \sigma_3^2 + \beta_2^2\sigma_2^2, \quad (iii)\ \ \sigma_1^2 < \sigma_3^2 + \beta_2^2\sigma_2^2 + \beta_1^2\beta_2^2\sigma_1^2.$$

Assumption (B):

$$(i)\ \ \frac{\sigma_2^2}{\sigma_1^2} > (1 - \beta_1^2), \quad (ii)\ \ \frac{\sigma_3^2}{\sigma_2^2} > (1 - \beta_2^2), \quad (iii')\ \ \frac{\sigma_3^2}{\sigma_1^2} > 1 - \frac{\beta_1^2\sigma_3^2}{\sigma_2^2}.$$

Relationship between (A) and (B)

- Suppose that $(\sigma_1^2, \sigma_2^2, \sigma_3^2) = (2, 2, 1)$ and $\beta = (\beta_1, 1)$. Assumption (B) is violated if $\beta_1^2 \leq 1$

$$\frac{1}{2} \leq 1 - \frac{\beta_1^2}{2}$$

  while Assumption (A) holds.

- Suppose that $(\sigma_1^2, \sigma_2^2, \sigma_3^2) = (2, 1.5, 1)$ and $\beta = (1, \beta_2)$. Then Assumption (A) is violated if $\beta_2^2 \leq 2/7$

$$\sigma_1^2 = 2 < \sigma_3^2 + \beta_2^2 \sigma_2^2 + \beta_1^2 \beta_2^2 \sigma_1^2 = 1 + 3.5\beta_2^2$$

  while Assumption (B) holds.

Prior Identifiability Assumptions for Gaussian SEMs

Identifiability Assumption (Peter and Bülmann, 2014, Loh et al., 2014, Ghoshal and Honorio, 2017)

Let $P(X)$ be generated from a Gaussian linear SEM with directed acyclic graph G. If all error variances are the same, the model is identifiable.

Corollary

The DAG G is uniquely identifiable, if there exists $\pi \in \Pi_G$ satisfying either of the two following conditions: For any node $m \in V$, let $j = \pi_m$, $k \in \mathrm{De}(j)$, and $\ell \in \mathrm{An}(j)$,

(A) Forward Selection:

$$0 < \sum_{k' \in \mathrm{Pa}(k) \setminus 1:(j-1)} \beta^2_{k' \to k}.$$

(B) Backward Elimination:

$$0 < \sum_{k' \in \mathrm{Ch}(\ell)} \beta^2_{\ell k'}.$$

Algorithm for (Gaussian) Linear SEMs

Input: n i.i.d. samples from a (Gaussian) Linear SEM, $X^{1:n}$.

Output: Estimated causal graph, $\widehat{G} = (V, \widehat{E})$.

Step (1): Ordering Estimation using the OLS.

Step (2): Parents Estimation using independence tests.

Note that

$$\mathrm{Var}(X_j \mid X_S) = X_j^T (I - Proj_S) X_j$$

Assumptions

- There exists positive constants ρ_{min} and ρ_{max} such that the smallest and largest eigenvalue of covariance matrix are bounded

$$\rho_{min} \leq \Lambda_{\min}(\Sigma) \leq \Lambda_{\max}(\Sigma) \leq \rho_{\max}$$

- For any $m \in \{1, 2, ..., p\}$, let $j = \pi_m$, $k \in \mathrm{De}(j)$, $\ell \in \mathrm{An}(j)$, and $1 : j = \{\pi_1, \pi_2, .., \pi_m\}$. Then, there exist positive constants $M_{forw}, M_{back} > 0$ such that
  - Forward Selection:

$$\sigma_j^2 + M_{forw} < \sigma_k^2 + \mathbb{E}(\mathrm{Var}(\mathbb{E}(X_k \mid X_{\mathrm{Pa}(k)}) \mid X_{1:(j-1)})),$$

  - Backward Elimination:

$$\sigma_j^2 - M_{back} > \sigma_\ell^2 + \mathbb{E}(\mathrm{Var}(\mathbb{E}(X_\ell \mid X_{1:(j-1)}) \mid X_{\mathrm{Pa}(\ell)}))$$

Recovery of the Ordering

Theorem

Consider a linear SEM with a true set of ordering Π. Suppose that $\widehat{\pi}$ is the estimated ordering via our algorithm. Then, there exists positive constants C_1 such that

$$P(\widehat{\pi} \in \Pi) \geq 1 - \exp(-C_1 n / \log^2 n).$$

Sketch of the Proof I

In linear SEM settings, a conditional variance of X_j given X_S can be expressed as

$$\Sigma_{jj} - \Sigma_{j,S}\Sigma_{S,S}^{-1}\Sigma_{S,j}.$$

Then, we have,

$$\begin{aligned}
|\widehat{\mathrm{Var}}(X_j \mid X_S) - \mathrm{Var}(X_j \mid X_S)| &= \left|\left(\widehat{\Sigma}_{jj} - \widehat{\Sigma}_{j,S}\widehat{\Sigma}_{S,S}^{-1}\widehat{\Sigma}_{S,j}\right) - \left(\Sigma_{jj} - \Sigma_{j,S}\Sigma_{S,S}^{-1}\Sigma_{S,j}\right)\right| \\
&\leq \left|\left(\widehat{\Sigma}_{jj} - \Sigma_{jj}\right)\right| + \left|\left(\widehat{\Sigma}_{j,S}\widehat{\Sigma}_{S,S}^{-1}\widehat{\Sigma}_{S,j}\right) - \left(\Sigma_{j,S}\Sigma_{S,S}^{-1}\Sigma_{S,j}\right)\right|.
\end{aligned}$$

The first term is bounded directly from the Lemma 1 of Ravikumar et al., 2011 for given $j \in V$:

$$P\left(\max_{j,k\in V}\left|\left(\widehat{\Sigma}_{jk} - \Sigma_{jk}\right)\right| \geq \frac{1}{\sqrt{\log n}}\right) \leq 4 \cdot \exp\left\{\frac{-n}{C_1 \log n}\right\},$$

where $C_1 = 1600 \max_j(\Sigma_{jj})^2$.

Sketch of the Proof II

The second term is also bounded by the following three terms:

$$\left| \left(\widehat{\Sigma}_{j,S} \widehat{\Sigma}_{S,S}^{-1} \widehat{\Sigma}_{S,j} \right) - \left(\Sigma_{j,S} \Sigma_{S,S}^{-1} \Sigma_{S,j} \right) \right|$$

$$\leq \left| \widehat{\Sigma}_{j,S} (\widehat{\Sigma}_{S,S}^{-1} - \Sigma_{S,S}^{-1}) \widehat{\Sigma}_{S,j} \right| + \left| \widehat{\Sigma}_{j,S} \Sigma_{S,S}^{-1} (\widehat{\Sigma}_{S,j} - \Sigma_{S,j}) \right| + \left| (\widehat{\Sigma}_{j,S} - \Sigma_{j,S}) \Sigma_{S,S}^{-1} \Sigma_{S,j} \right|$$

$$\leq \|\widehat{\Sigma}_{j,S}\|_2 \Lambda_{\max}(\widehat{\Sigma}_{S,S}^{-1} - \Sigma_{S,S}^{-1}) + \Lambda_{\max}(\Sigma_{S,S}^{-1}) \|\widehat{\Sigma}_{j,S} - \Sigma_{j,S}\|_2 (\|\widehat{\Sigma}_{j,S}\|_2 + \|\Sigma_{S,j}\|_2)$$

$$\leq \|\widehat{\Sigma}_{j,S}\|_2 \Lambda_{\max}(\widehat{\Sigma}_{S,S}^{-1} - \Sigma_{S,S}^{-1}) + \Lambda_{\max}(\Sigma_{S,S}^{-1}) \|\widehat{\Sigma}_{j,S} - \Sigma_{j,S}\|_2 (\|\widehat{\Sigma}_{j,S}\|_2 + \|\Sigma_{S,j}\|_2),$$

This is bounded directly from the Lemma 29 of Loh et al., 2014 for given $j \in V$ and $S \subset V$: for sufficiently large n such that $2\Lambda_{\max}(\Sigma^{-1}) \leq \sqrt{\log n}$,

$$P\left(\Lambda_{\max}(\widehat{\Sigma}_{S,S} - \Sigma_{S,S}) \right) \geq \frac{1}{\sqrt{\log n}} \Big) \leq 2 \exp\left(-C_2 \frac{n}{\log n} \right).$$

and,

$$P\left(\Lambda_{\max}(\widehat{\Sigma}_{S,S}^{-1} - \Sigma_{S,S}^{-1}) \right) \geq \frac{\Lambda_{\max}(\Sigma^{-1})}{\sqrt{\log n}} \Big) \leq 2 \exp\left(-C_3 \frac{n}{\log n} \right).$$

Sketch of the Proof III

In addition, we have

$$\|\widehat{\Sigma}_{j,S} - \Sigma_{j,S}\|_2 \leq \Lambda_{\max}(\widehat{\Sigma}_{S',S'} - \Sigma_{S',S'})$$

where $S' = \{j\} \cup S$. Furthermore, we also have,

$$\|\widehat{\Sigma}_{j,S}\|_2 \leq \|\Sigma_{j,S}\|_2 + \Lambda_{\max}(\widehat{\Sigma}_{S',S'} - \Sigma_{S',S'})$$

Hence, for a sufficiently large n, there exists a positive constant $C_4 > 0$ such that

$$\begin{aligned}
&|\widehat{\mathrm{Var}}(X_j \mid X_S) - \mathrm{Var}(X_j \mid X_S)| \\
&\leq \frac{1}{\sqrt{\log n}} + \left(\|\Sigma_{j,S}\|_2 + \frac{1}{\sqrt{\log n}}\right) \frac{\Lambda_{\max}(\Sigma^{-1})}{\sqrt{\log n}} + \frac{\Lambda_{\max}(\Sigma^{-1})}{\sqrt{\log n}} \left(2\|\Sigma_{j,S}\|_2 + \frac{1}{\sqrt{\log n}}\right) \\
&\leq \frac{M_{forw}}{2}
\end{aligned}$$

with probability at least of $1 - \exp(-C_4 n/\log n)$.

Bounded Partial Correlation

Assumption

Suppose that a true ordering is provided, and it is $\pi^* = (1, 2, ..., p)$. For all $(k, j) \in E$ and $\mathrm{Pa}(j) \setminus \{k\} \subset S \subset \mathrm{Nd}(j) \setminus \{k\}$, there exists $M > 0$ and $\kappa(n) = O(\sqrt{1/\log(n)})$ such that

$$0 < \kappa(n) < \inf_{j,k} |\rho_{j,k,S}| < \sup_{j,k} |\rho_{j,k,S}| < M < 1.$$

- In finite sample settings, a partial correlation should be far from 0.

Recovery of the Edges

Theorem

Consider a Gaussian linear SEM with edge parameters (β_{jk}) and error variances $(\sigma_j^2)_{j \in V}$. Suppose that true ordering is provided and $\widehat{E}$ is estimated from the Step (2) of our algorithm. Then, there exists positive constants C_2 and C_3 such that

$$P(\widehat{E} = E) \geq 1 - C_2 \cdot n \cdot \exp\left(-C_3 n \cdot \kappa(n)^2\right).$$

- Therefore, our algorithm can consistently recover the graph.

Sketch of the Proof I

The sample partial correlation $\rho_{j,k,S}$ can be calculated via linear regression: for some $s \in S$,

$$\rho_{j,k,S} = \frac{\rho_{j,k,S\setminus s} - \rho_{j,s,S\setminus s}\rho_{k,s,S\setminus s}}{\sqrt{(1-\rho^2_{j,s,S\setminus s})(1-\rho^2_{k,s,S\setminus s})}}.$$

With theses partial correlations, we apply Fisher's z-transform for the conditional independence tests:

$$Z_{j,k,S} = \frac{1}{2}\log\left(\frac{1+\widehat{\rho}_{j,k,S}}{1-\widehat{\rho}_{j,k,S}}\right).$$

Sketch of the Proof II

By Lemma3 of Kalisch 2007, for any $\gamma > 0$,

$$\sup_{j,k,S} P(|Z_{j,k,S} - z_{j,k,S}| > \gamma)$$
$$\leq O(n-d)\left(\exp\left((n-p-4)\log\left(\frac{4-(\gamma/L)^2}{4+(\gamma/L)^2}\right)\right) + \exp(-C_2(n-p))\right),$$

for some constant $0 < C_2 < \infty$ and $L = 1/(1-(1+M)^2/4)$.

Let $E_{j,k,S}$ is an error event that consists of type I and II errors where $E_{j,k,S} = E^I_{j,k,S} \cup E^{II}_{j,k,S}$. Then, the each type of error is as follows:

type I Error$E^I_{j,k,S}$: $\sqrt{n-|S|-3}\,|Z_{j,k,S}| > \Phi^{-1}(1-\alpha/2)$ when $z_{j,k,S} = 0$,

type I Error$E^{II}_{j,k,S}$: $\sqrt{n-|S|-3}\,|Z_{j,k,S}| \leq \Phi^{-1}(1-\alpha/2)$ when $z_{j,k,S} \neq 0$.

Sketch of the Proof III

Let significance level $\alpha_n = 2(1 - \Phi(\sqrt{n} \cdot \kappa(n)/2))$. Then, we have,

$$\begin{aligned}\sup_{j,k,S} P(E^I_{j,k,S}) &= \sup_{j,k,S} P\left(|Z_{j,k,S} - z_{j,k,S}| > \sqrt{\frac{n}{n - |S| - 3}}\frac{\kappa(n)}{2}\right) \\ &\leq O(n-p)\exp\left(-C_3(n-p)\kappa(n)^2\right),\end{aligned}$$

for some positive constant $C_3 > 0$.

In addition, we have,

$$\begin{aligned}\sup_{j,k,S} P(E^{II}_{j,k,S}) &= \sup_{j,k,S} P\left(|Z_{j,k,S} - z_{j,k,S}| > \frac{\sqrt{1 - n/(n - |S| - 3)}\kappa(n)}{2}\right) \\ &\leq O(n-p)\exp(-C_4(n-p)\kappa(n)^2),\end{aligned}$$

for some positive constant $C_4 > 0$.

Summary

- We provide new identifiability assumption for structural equation models.
- We also provide a consistent and computationally feasible algorithm.

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

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
```

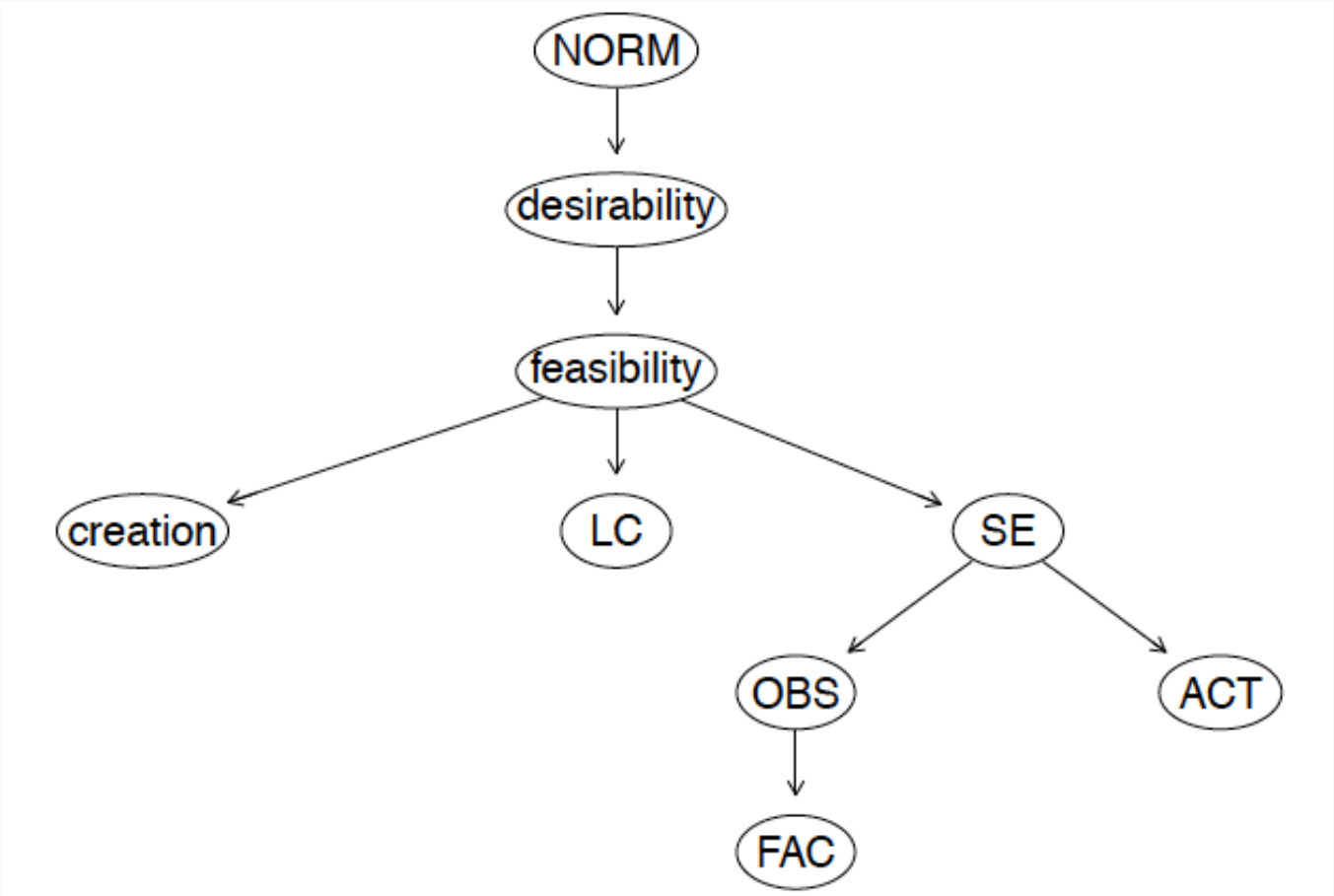
The same is true for decreasing OBS, but not as much; the reason is that OBS is farther away from creation so the effect of the conditioning is smaller.

```
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!

That's It, Thanks!