COMP9418: Advanced Topics in Statistical Machine Learning

Learning Bayesian Networks Structure with Maximum Likelihood

Instructor: Gustavo Batista

University of New South Wales

Introduction

- So far, we have assumed we know the structure of a Bayesian network
  - We were mostly concerned with estimating its parameters
  - The main approach for this estimation has been the search for ML estimates
  - That is, the ones that maximize the probability of observing a given dataset
- Now, we assume the network structure is unknown
  - We want to learn the structure from a given dataset
- We adopt the same approach for parameter estimation
  - That is, we search for network structures that maximize the probability of observing a given dataset
  - We start with this approach and show it leads to a general class of scoring functions for network structures
- We focus on learning structures from complete data
  - Dealing with incomplete data is similar but much more demanding computationally

Introduction

- Consider the Bayesian network on the right
  - Remind that in the previous lecture, we defined the log-likelihood of a structure G as

$$LL(G;\mathcal{D}) \stackrel{\text{def}}{=} \log L(\theta^{ml};\mathcal{D})$$

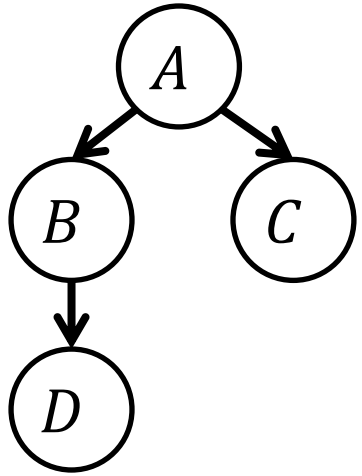
  - For the dataset $\mathcal{D}$, we can compute $LL(G;\mathcal{D})$ as

$$LL(G;\mathcal{D}) = \log \prod_{i=1}^{5} P_{\theta^{ml}}(\boldsymbol{d}_i) = \sum_{i=1}^{5} \log P_{\theta^{ml}}(\boldsymbol{d}_i)$$

$\mathcal{D}$	A	B	C	D	$P_{\theta^{ml}}(\boldsymbol{d}_i)$		$\log P_{\theta^{ml}}(\boldsymbol{d}_i)$
1	a	b	$\bar{c}$	d	$^3/_4 \times ^3/_4 \times ^4/_5 \times ^1/_4$	.1125	-3.15
2	a	b	$\bar{c}$	$\bar{d}$	$^3/_4 \times ^3/_4 \times ^4/_5 \times ^3/_4$	.3375	-1.57
3	a	$\bar{b}$	c	d	$^1/_4 \times ^1/_4 \times ^4/_5 \times 1$	.05	-4.32
4	$\bar{a}$	b	c	$\bar{d}$	$1 \times 1 \times ^1/_5 \times ^3/_4$	.15	-2.74
5	a	b	$\bar{c}$	$\bar{d}$	$^3/_4 \times ^3/_4 \times ^4/_5 \times ^3/_4$	.3375	-1.57
						$LL(G;\mathcal{D}) =$	-13.35

A	B	$\theta^{ml}_{b\|a}$
a	b	$3/4$
a	$\bar{b}$	$1/4$
$\bar{a}$	b	1
$\bar{a}$	$\bar{b}$	0

A	C	$\theta^{ml}_{c\|a}$
a	c	$1/4$
a	$\bar{c}$	$3/4$
$\bar{a}$	c	1
$\bar{a}$	$\bar{c}$	0



A	θ^{ml}_{a}
a	$4/5$
$\bar{a}$	$1/5$

B	D	$\theta^{ml}_{b\|d}$
b	d	$1/4$
b	$\bar{d}$	$3/4$
$\bar{b}$	d	1
$\bar{b}$	$\bar{d}$	0

$\mathcal{D}$	A	B	C	D
1	a	b	$\bar{c}$	d
2	a	b	$\bar{c}$	$\bar{d}$
3	a	$\bar{b}$	c	d
4	$\bar{a}$	b	c	$\bar{d}$
5	a	b	$\bar{c}$	$\bar{d}$

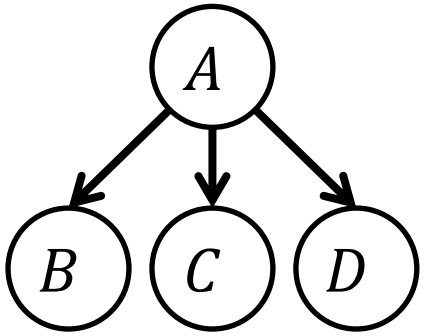
Introduction

- Now, consider this alternate network structure G^* and ML estimates
  - Let us compute the log-likelihood of this network for the same dataset

$\mathcal{D}$	A	B	C	D	$P_{\theta^{ml}}(\boldsymbol{d}_i)$		$\log P_{\theta^{ml}}(\boldsymbol{d}_i)$
1	a	b	$\bar{c}$	d	$3/4 \times 3/4 \times 4/5 \times 1/2$	.225	-2.15
2	a	b	$\bar{c}$	$\bar{d}$	$3/4 \times 3/4 \times 4/5 \times 1/2$	.225	-2.15
3	a	$\bar{b}$	c	d	$1/4 \times 1/4 \times 4/5 \times 1/2$	.025	-5.32
4	$\bar{a}$	b	c	$\bar{d}$	$1 \times 1 \times 1/5 \times 1$	.2	-2.32
5	a	b	$\bar{c}$	$\bar{d}$	$3/4 \times 3/4 \times 4/5 \times 1/2$	.225	-2.15
						$LL(G^*;\mathcal{D}) =$	-14.09

- As $LL(G^*;\mathcal{D}) < LL(G;\mathcal{D})$, we prefer G to G^*
  - Our goal is to search for a maximum likelihood structure

A	B	$\theta^{ml}_{b\|a}$
a	b	$3/4$
a	$\bar{b}$	$1/4$
$\bar{a}$	b	1
$\bar{a}$	$\bar{b}$	0



A	C	$\theta^{ml}_{c\|a}$
a	c	$1/4$
a	$\bar{c}$	$3/4$
$\bar{a}$	c	1
$\bar{a}$	$\bar{c}$	0

A	θ^{ml}_{a}
a	$4/5$
$\bar{a}$	$1/5$

$\mathcal{D}$	A	B	C	D
1	a	b	$\bar{c}$	d
2	a	b	$\bar{c}$	$\bar{d}$
3	a	$\bar{b}$	c	d
4	$\bar{a}$	b	c	$\bar{d}$
5	a	b	$\bar{c}$	$\bar{d}$

A	D	$\theta^{ml}_{d\|a}$
a	d	$1/2$
a	$\bar{d}$	$1/2$
$\bar{a}$	d	0
$\bar{a}$	$\bar{d}$	1

Learning Tree Structures

- We start exploring an algorithm based on a scoring measure for tree structures
  - Therefore each node will have at most one parent
  - The scoring function is expressed in terms of mutual information
- Mutual information between variables X and U
  - It is a measure of dependence between these variables in a distribution
  - Our scoring measure is based on mutual information in the empirical distribution
- Given a tree structure G with edges $U \rightarrow X$
  - The scoring function is given by
  - The score is the addition of MI between every variable and its single parent in the tree structure

$$MI_{\mathcal{D}}(X,U) \stackrel{\text{def}}{=} \sum_{x,u} P_{\mathcal{D}}(x,u) \log \frac{P_{\mathcal{D}}(x,u)}{P_{\mathcal{D}}(x)P_{\mathcal{D}}(u)}$$

$$tScore(G;\mathcal{D}) \stackrel{\text{def}}{=} \sum_{U \rightarrow X} MI_{\mathcal{D}}(X,U)$$

Learning Tree Structures

- We can show that trees having a maximal likelihood are those trees that maximize $tScore$
  - That is, if G is a tree structure, and $\mathcal{D}$ is a complete dataset, then

$$argmax_G\ tScore(G;\mathcal{D}) = argmax_G\ LL(G;\mathcal{D})$$

- Therefore, we can find a maximum likelihood tree using an algorithm for computing maximum spanning trees
  - This is known as the Chow-Liu algorithm
  - We will discuss this algorithm shortly
  - But before that, lets us understand the relationship between $tScore$ and maximum log-likelihood

$tScore$ and Maximum Log-likelihood

- Let us remind the definitions of
  - Entropy
  - Conditional entropy, and
  - Mutual information
- Also, remember from the previous lecture that the log-likelihood decomposes into components
  - Each component is a conditional entropy
- Expanding MI and substituting the definitions of entropy and conditional entropy leads to
  - $MI_{\mathcal{D}}(X, \boldsymbol{U}) = H_{\mathcal{D}}(X) - H_{\mathcal{D}}(X|\boldsymbol{U})$

$$H_{\mathcal{D}}(X) \stackrel{\text{def}}{=} -\sum_{x} P_{\mathcal{D}}(x) \log P_{\mathcal{D}}(x)$$

$$H_{\mathcal{D}}(X|\boldsymbol{U}) \stackrel{\text{def}}{=} -\sum_{x,\boldsymbol{u}} P_{\mathcal{D}}(x, \boldsymbol{u}) \log P_{\mathcal{D}}(x|\boldsymbol{u})$$

$$MI_{\mathcal{D}}(X, \boldsymbol{U}) \stackrel{\text{def}}{=} \sum_{x,\boldsymbol{u}} P_{\mathcal{D}}(x, \boldsymbol{u}) \log \frac{P_{\mathcal{D}}(x, \boldsymbol{u})}{P_{\mathcal{D}}(x) P_{\mathcal{D}}(\boldsymbol{u})}$$

$$LL(G; \mathcal{D}) = -N \sum_{X\boldsymbol{U}} H_{\mathcal{D}}(X|\boldsymbol{U})$$

$tScore$ and Log ML

- Therefore,

$$LL(G;\mathcal{D}) = -N \sum_{XU} H_{\mathcal{D}}(X|U)$$

$$= -N \sum_{XU} (H_{\mathcal{D}}(X) - MI_{\mathcal{D}}(X,U))$$

$$= -N \sum_{XU} H_{\mathcal{D}}(X) + N \sum_{XU} MI_{\mathcal{D}}(X,U)$$

- Note that neither N nor the term $-N \sum_{XU} H_{\mathcal{D}}(X)$ depend on the tree structure G. Hence,

$$argmax_G\, LL(G;\mathcal{D}) = argmax_G \sum_{XU} MI_{\mathcal{D}}(X,U) = argmax_G\, tScore(G;\mathcal{D})$$

$$H_{\mathcal{D}}(X) \stackrel{\text{def}}{=} -\sum_{x} P_{\mathcal{D}}(x) \log P_{\mathcal{D}}(x)$$

$$H_{\mathcal{D}}(X|\boldsymbol{U}) \stackrel{\text{def}}{=} -\sum_{x,\boldsymbol{u}} P_{\mathcal{D}}(x,\boldsymbol{u}) \log P_{\mathcal{D}}(x|\boldsymbol{u})$$

$$MI_{\mathcal{D}}(X,\boldsymbol{U}) \stackrel{\text{def}}{=} \sum_{x,\boldsymbol{u}} P_{\mathcal{D}}(x,\boldsymbol{u}) \log \frac{P_{\mathcal{D}}(x,\boldsymbol{u})}{P_{\mathcal{D}}(x) P_{\mathcal{D}}(\boldsymbol{u})}$$

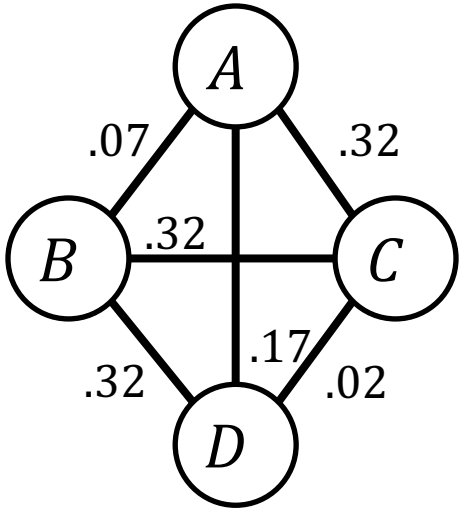
$$LL(G;\mathcal{D}) = -N \sum_{X\boldsymbol{U}} H_{\mathcal{D}}(X|\boldsymbol{U})$$

$$MI_{\mathcal{D}}(X,\boldsymbol{U}) = H_{\mathcal{D}}(X) - H_{\mathcal{D}}(X|\boldsymbol{U})$$

Learning Tree Structures

- Let us return to the algorithm for learning ML tree structures using $tScore$
  - We illustrate this algorithm for variables A, B, C and D and the dataset on the right
- The first step is constructing a complete, undirected graph over all variables
  - The cost of each edge is the mutual information between the two nodes connected by the edge
  - Let us detail the computation for edge $A - B$

$\mathcal{D}$	A	B	C	D
1	a	b	$\bar{c}$	d
2	a	b	$\bar{c}$	$\bar{d}$
3	a	$\bar{b}$	c	d
4	$\bar{a}$	b	c	$\bar{d}$
5	a	b	$\bar{c}$	$\bar{d}$



Learning Tree Structures

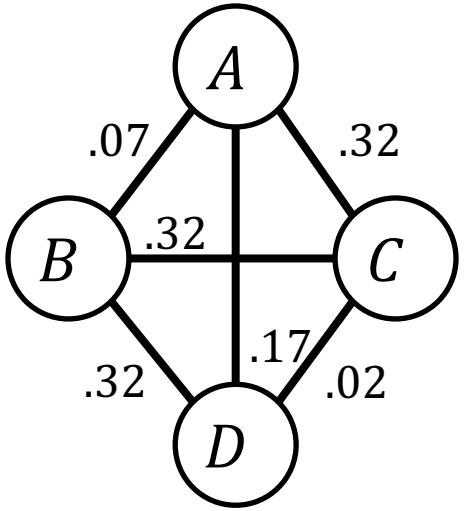
$$MI_{\mathcal{D}}(X,U) \stackrel{\text{def}}{=} \sum_{x,u} P_{\mathcal{D}}(x,u) \log \frac{P_{\mathcal{D}}(x,u)}{P_{\mathcal{D}}(x)P_D(u)}$$

$$= P_{\mathcal{D}}(a,b) \log \frac{P_{\mathcal{D}}(a,b)}{P_{\mathcal{D}}(a)P_D(b)} + P_{\mathcal{D}}(a,\bar{b}) \log \frac{P_{\mathcal{D}}(a,\bar{b})}{P_{\mathcal{D}}(a)P_D(\bar{b})}$$

$$= P_{\mathcal{D}}(\bar{a},b) \log \frac{P_{\mathcal{D}}(\bar{a},b)}{P_{\mathcal{D}}(\bar{a})P_D(b)} + P_{\mathcal{D}}(\bar{a},\bar{b}) \log \frac{P_{\mathcal{D}}(\bar{a},\bar{b})}{P_{\mathcal{D}}(\bar{a})P_D(\bar{b})}$$

$\mathcal{D}$	A	B	C	D
1	a	b	$\bar{c}$	d
2	a	b	$\bar{c}$	$\bar{d}$
3	a	$\bar{b}$	c	d
4	$\bar{a}$	b	c	$\bar{d}$
5	a	b	$\bar{c}$	$\bar{d}$

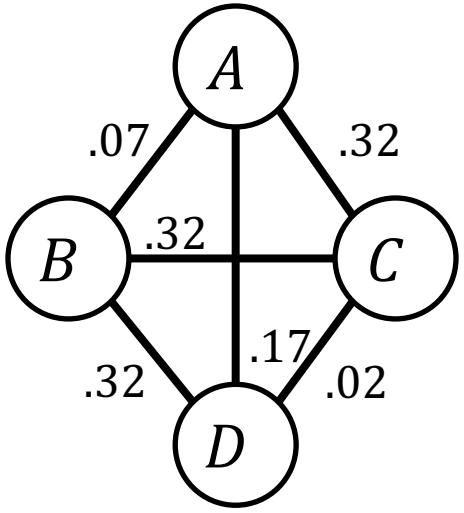
A	B	$P(A,B)$	$P(A)$	$P(B)$	$R(A,B)$	$\log R(A,B)$	$P \log R(A,B)$
a	b	$3/5$	$4/5$	$4/5$	.94	$-.09$	$-.054$
a	$\bar{b}$	$1/5$	$4/5$	$1/5$	1.25	.32	.064
$\bar{a}$	b	$1/5$	$1/5$	$4/5$	1.25	.32	.064
$\bar{a}$	$\bar{b}$	$0/5$	$1/5$	$1/5$	0	$-\infty$	0
						$MI_{\mathcal{D}}(A,B) =$	.074



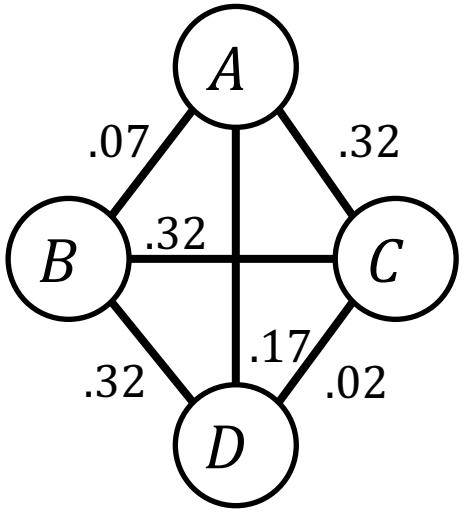
Learning Tree Structures

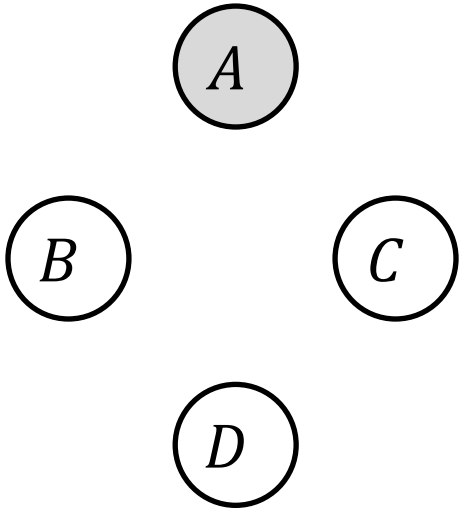
- The first step is constructing a complete, undirected graph over all variables
  - The cost of each edge is the mutual information between the two nodes connected by the edge
  - We compute the spanning tree with maximal cost
    - The cost of the tree is just the sum of costs associated with its edges
    - The algorithm for maximum spanning tree is a trivial adaptation of algorithms for minimum spanning tree such as Prim and Kruskal
- This method generates an undirected spanning tree
  - It coincides with several directed trees
  - We can choose any of these trees by selecting a node as root and directing edges away from the root
- The resulting trees are guaranteed to have a maximal likelihood among all tree structures

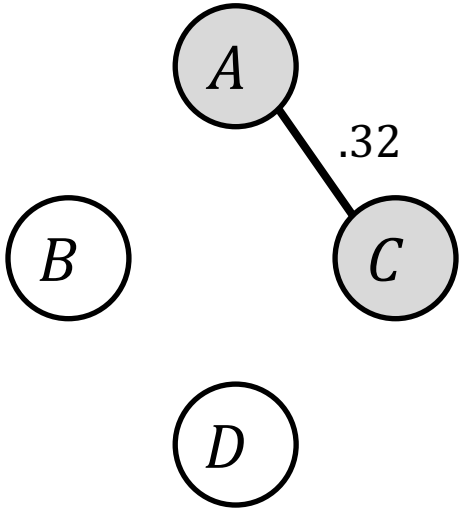
$\mathcal{D}$	A	B	C	D
1	a	b	$\bar{c}$	d
2	a	b	$\bar{c}$	$\bar{d}$
3	a	$\bar{b}$	c	d
4	$\bar{a}$	b	c	$\bar{d}$
5	a	b	$\bar{c}$	$\bar{d}$

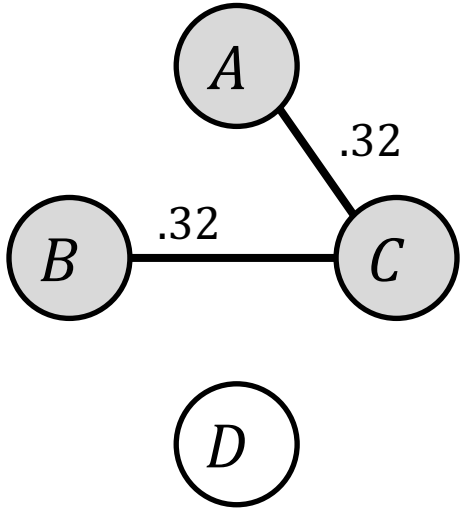


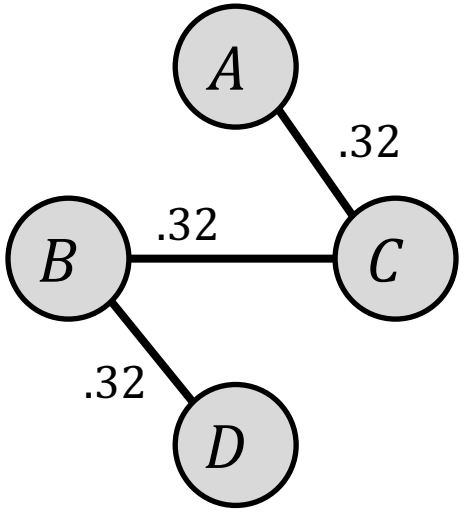
Learning Tree Structures: Prim



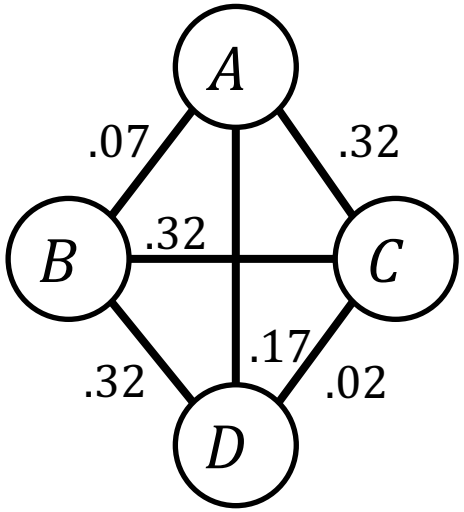


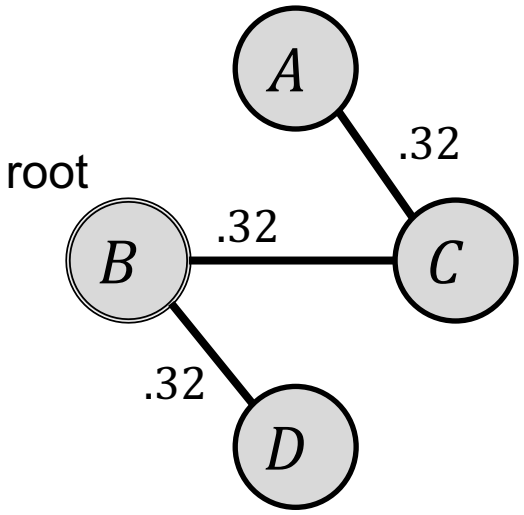


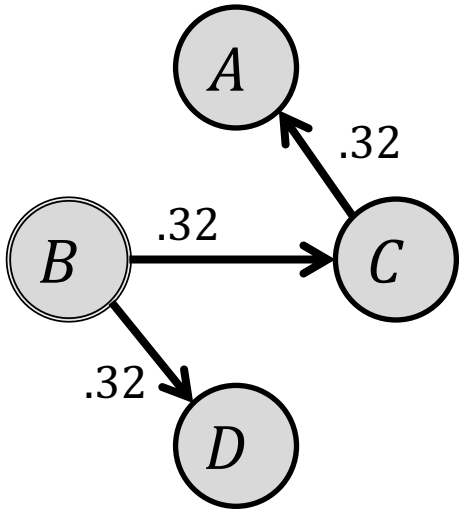




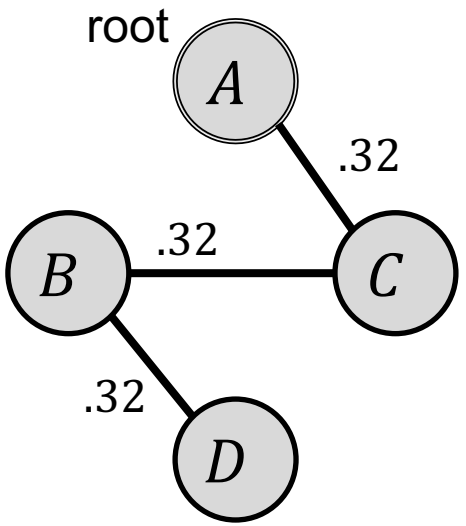
Learning Tree Structures: Prim

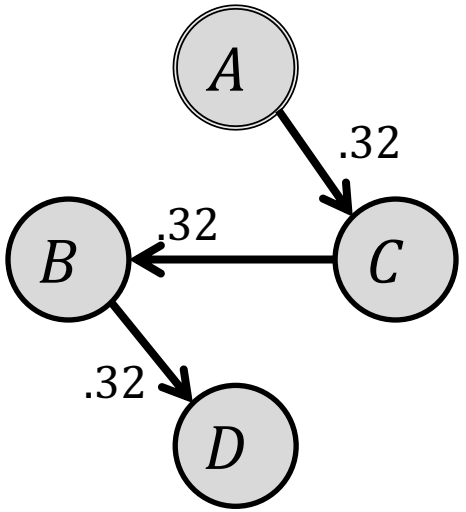






Log-likelihood: -12.1





Log-likelihood: -12.1

Learning Tree Structures

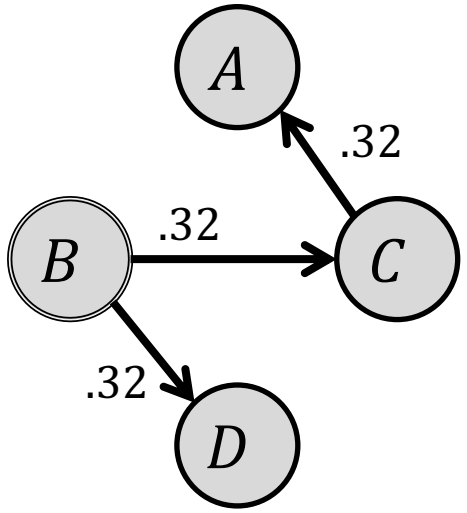
- Given a tree, we can compute the log-likelihood by
  - Computing the probability of each case in the dataset, as we did in slide 7
  - Or use the following equation that shows the log-likelihood corresponds to a sum of terms, one for each family in the network

$$LL(G;\mathcal{D}) = -N \sum_{XU} H_{\mathcal{D}}(X|U)$$

$\mathcal{D}$	A	B	C	D
1	a	b	$\bar{c}$	d
2	a	b	$\bar{c}$	$\bar{d}$
3	a	$\bar{b}$	c	d
4	$\bar{a}$	b	c	$\bar{d}$
5	a	b	$\bar{c}$	$\bar{d}$

- For this network, we have

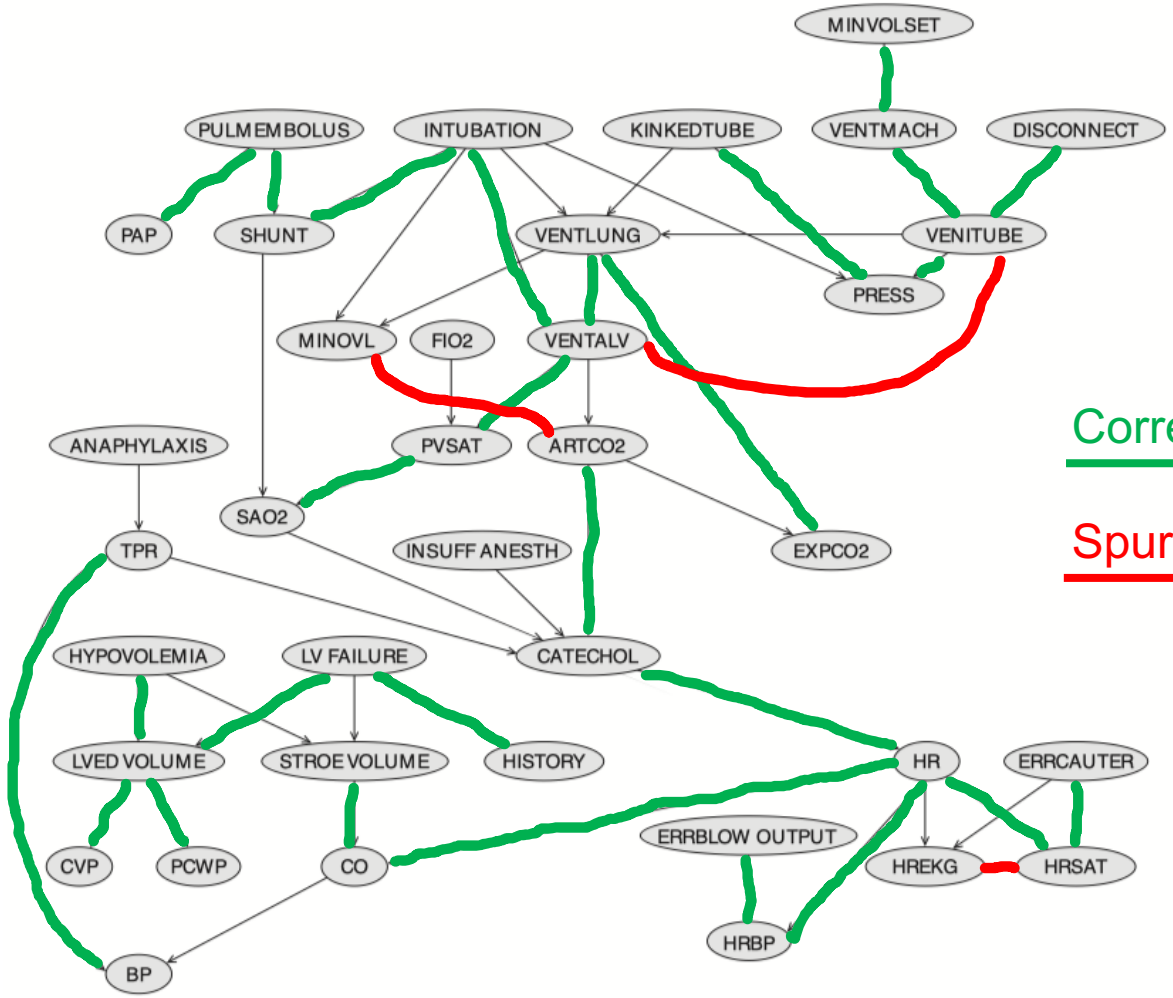
$$\begin{aligned} & LL(G;\mathcal{D}) \\ & \quad = -N(H_{\mathcal{D}}(A|C) + H_{\mathcal{D}}(B) + H_{\mathcal{D}}(C|B) + H_{\mathcal{D}}(D|B)) \\ & \quad = -5(.400 + .722 + .649 + .649) \\ & \quad = -12.1 \end{aligned}$$



- Notice the terms correspond to families of the tree structure
  - AC, B, CB, and DB

Learning Tree Structures: Example

- Tree learned from ICU-Alarm network data
- Not every edge in tree is in the original network
- Inferred edges are undirected



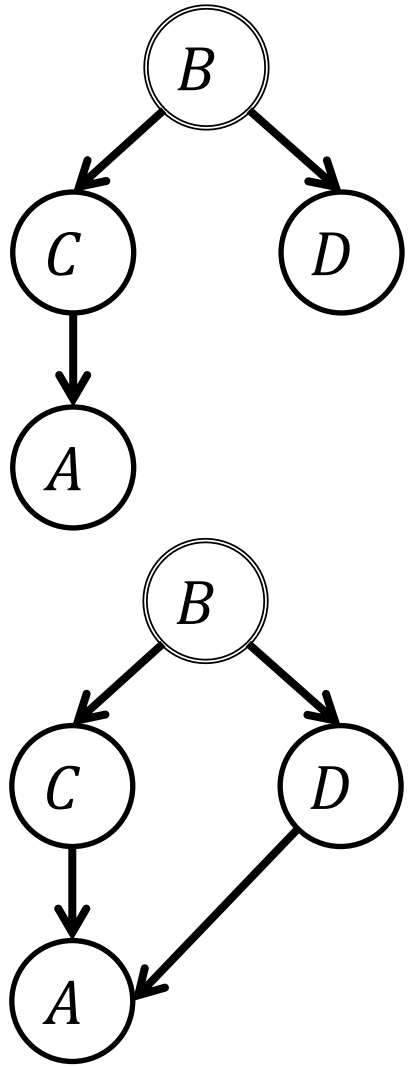
Learning DAG Structures

- Suppose now our goal is to find a maximum likelihood structure
  - Without restricting to tree structures

- For example, consider the tree structure obtained in the previous slide plus the edge $D \rightarrow A$
  - The log-likelihood of this DAG is

$$LL(G;\mathcal{D}) = -N(H_{\mathcal{D}}(A|C,D) + H_{\mathcal{D}}(B) + H_{\mathcal{D}}(C|B) + H_{\mathcal{D}}(D|B))$$
$$= -5(0 + .722 + .649 + .649)$$
$$= -10.1$$

  - Which is larger than the log-likelihood of the tree



Learning DAG Structures

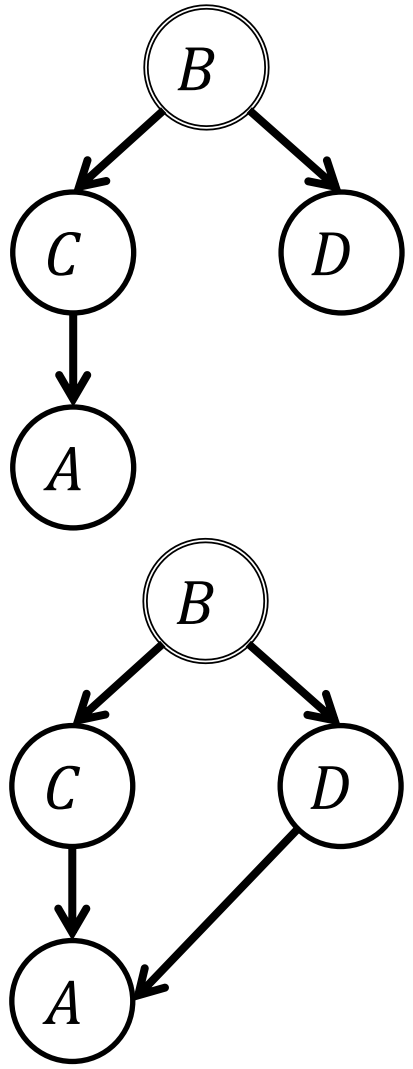
- Notice the only difference between the two likelihoods is the entropy for variable A
  - This is the only variable with different families in the two structures
  - The family of A is AC in the tree and ACD in the DAG

$$H_{\mathcal{D}}(A|C,D) < H_{\mathcal{D}}(A|C)$$

  - Hence

$$-H_{\mathcal{D}}(A|C,D) > -H_{\mathcal{D}}(A|C)$$

  - Which is why the DAG has larger log-likelihood than the tree
- However, this result is not completely accidental
  - We can show that if $\boldsymbol{U} \subseteq \boldsymbol{U}^*$, then $H_{\mathcal{D}}(X|\boldsymbol{U}) \geq H_{\mathcal{D}}(X|\boldsymbol{U}^*)$
- Adding more parents to a variable never increases the entropy term
  - Therefore, never decreases the log-likelihood of the resulting structure

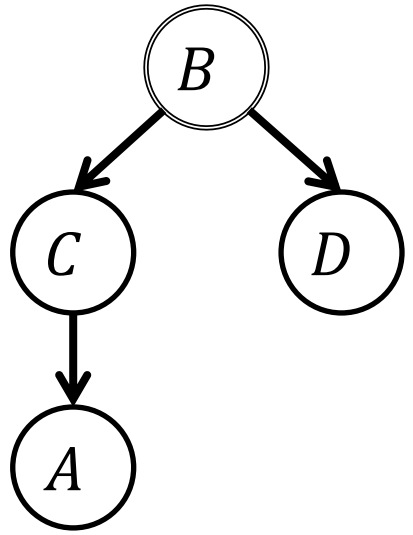


Learning DAG Structures and Overfitting

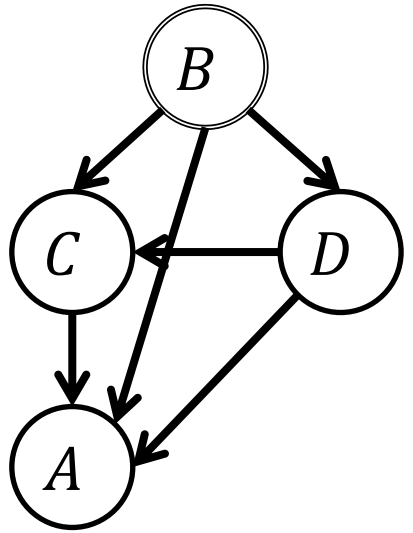
- Therefore, if a DAG G^* is the result of adding edges to a DAG G, then

$$LL(G^*; \mathcal{D}) \geq LL(G; \mathcal{D})$$

- If we simply search for a network structure with maximal likelihood
  - We end up choosing a complete network structure
  - That is, a DAG to which no more edges can be added without introducing cycles

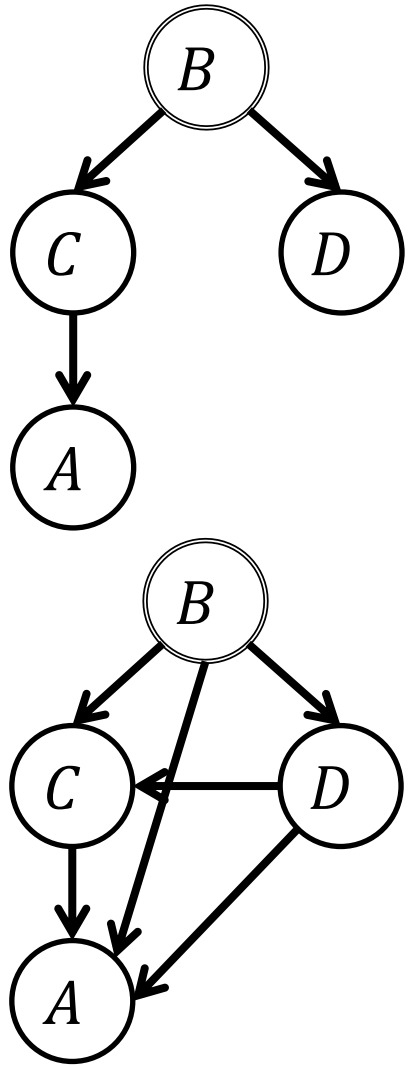


- Complete DAGs are undesirable for several reasons
  - They make no assertions of conditional independence
  - Their topology does not reveal any properties of the distribution they induce
  - A complete DAG over n variables has treewidth of $n - 1$
  - Complete DAGs suffer of the problem of *overfitting*: the use of a model that has too many parameters compared to the available data



Overfitting

- In summary, the problem of overfitting occurs when
  - We focus on learning a model that fits the data well
  - Without constraining enough the number of free model parameters
- The result is that we end up adopting models that are more complex than necessary
  - Such models tend to have poor generalization performance
  - That is, they perform poorly on cases that are not part of the dataset
- There is no agreed upon solution for overfitting
  - However, the available solutions are based on the principle of *Occam's Razor*
  - Which states we should prefer simpler models, other things being equal



Model Complexity: DAG Dimension

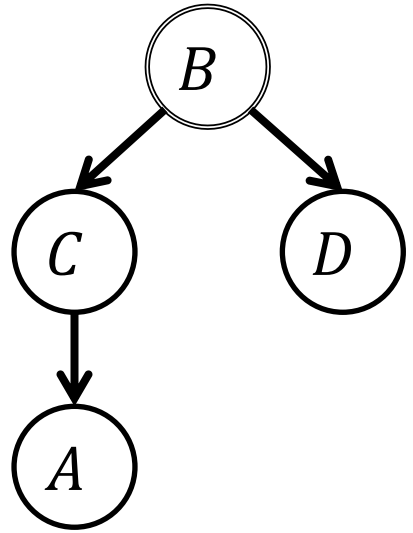
- To realize Occam's razor principle, we need
  - A measure of model complexity
  - A method for balancing model complexity and data fit
- A common choice for model complexity is the number of independent parameters in the model
  - This gives us the following measure of complexity
- Let G be a DAG over variables $X_1, \ldots, X_n$ with corresponding parents $\boldsymbol{U}_1, \ldots, \boldsymbol{U}_n$ and let $Y^{\#}$ denote the number of instantiations for variables Y
  - The *dimension of* G is defined as
  - Therefore, the dimension of a DAG is equal to the number of independent parameters in its CPTs

$$\|G\| \stackrel{\text{def}}{=} \sum_i \|X_i \boldsymbol{U}_i\|$$

$$\|X_i \boldsymbol{U}_i\| \stackrel{\text{def}}{=} \left(X_i^{\#} - 1\right) \boldsymbol{U}_i^{\#}$$

Model Complexity: DAG Dimension

- For example, the dimension of this DAG considering all binary variables is
  - $\|B\| = 1, \|CB\| = 2, \|DB\| = 2, \|AC\| = 2$. Total = 7

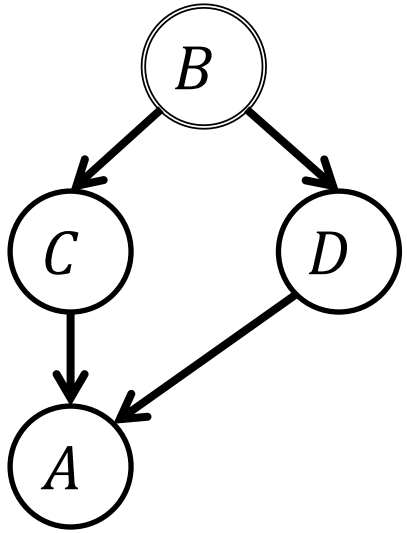


$$\|G\| \stackrel{\text{def}}{=} \sum_{i}^{N} \|X_i \boldsymbol{U}_i\|$$

$$\|X_i \boldsymbol{U}_i\| \stackrel{\text{def}}{=} \left(X_i^{\#} - 1\right) \boldsymbol{U}_i^{\#}$$

Model Complexity: DAG Dimension

- For example, the dimension of this DAG considering all binary variables is
  - $\|B\| = 1, \|CB\| = 2, \|DB\| = 2, \|AC\| = 2$. Total = 7
  - $\|B\| = 1, \|CB\| = 2, \|DB\| = 2, \|ACD\| = 4$. Total = 9

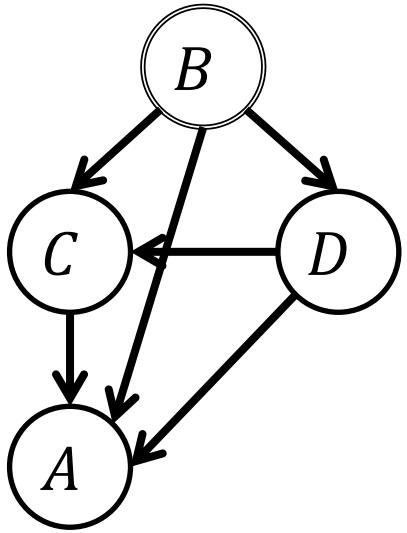


$$\|G\| \stackrel{\text{def}}{=} \sum_{i}^{N} \|X_i \boldsymbol{U}_i\|$$

$$\|X_i \boldsymbol{U}_i\| \stackrel{\text{def}}{=} \left(X_i^{\#} - 1\right) \boldsymbol{U}_i^{\#}$$

Model Complexity: DAG Dimension

- For example, the dimension of this DAG considering all binary variables is
  - $\|B\| = 1, \|CB\| = 2, \|DB\| = 2, \|AC\| = 2$. Total = 7
  - $\|B\| = 1, \|CB\| = 2, \|DB\| = 2, \|ACD\| = 4$. Total = 9
  - $\|B\| = 1, \|CBD\| = 4, \|DB\| = 2, \|ABCD\| = 8$. Total = 15

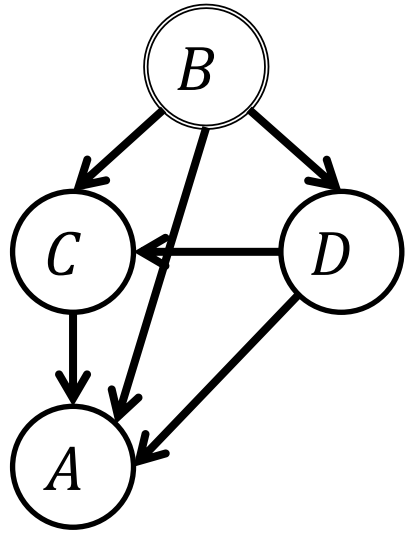


$$\|G\| \stackrel{\text{def}}{=} \sum_{i}^{N} \|X_i \boldsymbol{U}_i\|$$

$$\|X_i \boldsymbol{U}_i\| \stackrel{\text{def}}{=} \left(X_i^{\#} - 1\right) \boldsymbol{U}_i^{\#}$$

Model Complexity: DAG Dimension

- For example, the dimension of this DAG considering all binary variables is
  - $\|B\| = 1, \|CB\| = 2, \|DB\| = 2, \|AC\| = 2$. Total = 7
  - $\|B\| = 1, \|CB\| = 2, \|DB\| = 2, \|ACD\| = 4$. Total = 9
  - $\|B\| = 1, \|CBD\| = 4, \|DB\| = 2, \|ABCD\| = 8$. Total = 15



- Using this notion of model complexity, we can define the following class of scoring measures

$$Score(\mathrm{G};\mathcal{D}) \stackrel{\text{def}}{=} LL(G;\mathcal{D}) - \psi(N) \cdot \|G\|$$

  - The first component is the log-likelihood of the graph G
  - The second one $\psi(N) \cdot \|G\|$ is a penalty term that favours simple models
  - The penalty term has a weight, $\psi(N) \geq 0$, that is a function of the dataset size N

$$\|G\| \stackrel{\text{def}}{=} \sum_{i}^{N} \|X_i \boldsymbol{U}_i\|$$

$$\|X_i \boldsymbol{U}_i\| \stackrel{\text{def}}{=} \left(X_i^{\#} - 1\right)\boldsymbol{U}_i^{\#}$$

Scoring Measures

- When the penalty weight $\psi(N)$ is a constant independent of N
  - We obtain a score in which the model's complexity is a secondary issue
  - The log-likelihood function $LL(G;\mathcal{D})$ grows linearly in the dataset size N
  - Therefore the log-likelihood will quickly dominate the penalty term
  - The model complexity will be used to distinguish between models that have relatively equal log-likelihood terms
  - This scoring measure is known as *Akaike information criterion* (AIC)
- A more common choice is $\psi(N) = {\log_2 N}/{2}$, which leads to a more influential term
  - However, this term grows logarithmically in N, while the log-likelihood term grows linearly
  - The influence of model complexity decreases as N grows, allowing the log-likelihood eventually dominate
  - This penalty weight gives rise to the *minimum description length* (MDL) score

Minimum Description Length (MDL)

- Minimum Description Length (MDL) score

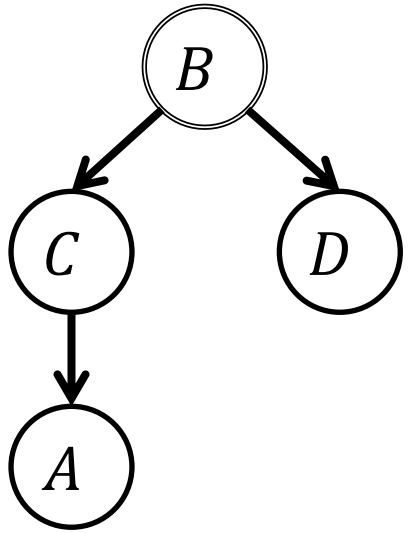
$$MDL(G;\mathcal{D}) \stackrel{\text{def}}{=} LL(G;\mathcal{D}) - \left(\frac{\log_2 N}{2}\right)\|G\|$$

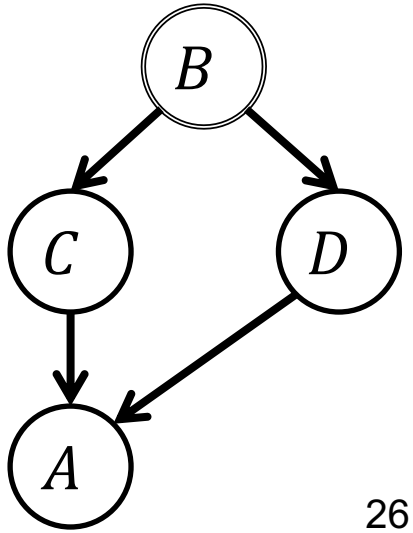
- For example, the network on the top has the following MDL score

$$= -12.1 - \left(\frac{\log_2 5}{2}\right)7$$
$$= -12.1 - 8.1$$
$$= -20.2$$

- For example, the network on the bottom has the following MDL score

$$= -10.1 - \left(\frac{\log_2 5}{2}\right)9$$
$$= -10.1 - 10.4$$
$$= -20.5$$





Minimum Description Length (MDL)

- Minimum Description Length (MDL) score

$$MDL(G;\mathcal{D}) \stackrel{\text{def}}{=} LL(G;\mathcal{D}) - \left(\frac{\log_2 N}{2}\right)\|G\|$$

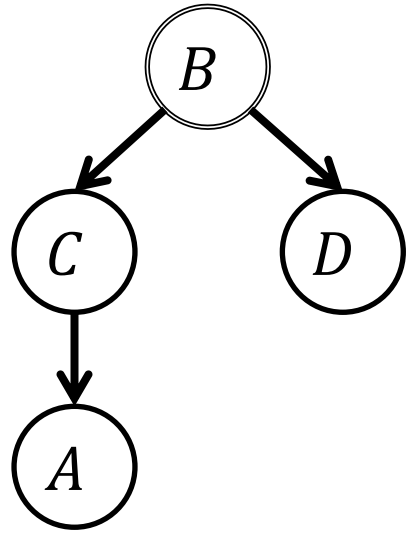
- For example, the network on the top has the following MDL score

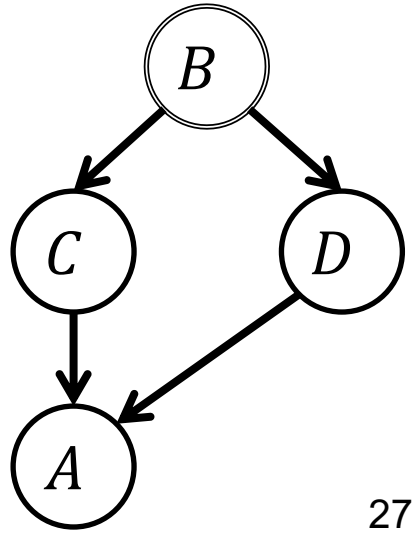
$$= -20.2$$

- For example, the network on the bottom has the following MDL score

$$= -20.5$$

- Therefore, MDL score prefers the top network even though it has smaller log-likelihood
- The MDL score is also known as *Bayesian information criterion* (BIC)
  - It is sometimes expressed as the negative of the score above
  - The goal becomes to minimize the score instead of maximizing it





Searching for Network Structure

- Searching for a network structure can be quite expensive
  - Due to the very large number of structures we need to consider
  - Greedy algorithms tend to be more practical when learning structures
  - Systematic search algorithms can be practical under some conditions
- Both classes of algorithms rely for their efficiency on a property of scoring functions
  - Most score functions are *decomposable* or *modular*
  - They allow to decompose the score into an aggregate of local scores
  - One for each network family

Searching for Network Structure

- For instance, the class of scores

$$Score(G;\mathcal{D}) \stackrel{\text{def}}{=} LL(G;\mathcal{D}) - \psi(N) \cdot \|G\|$$

  - can be decomposed as

$$Score(G;\mathcal{D}) = \sum_{X\boldsymbol{U}} Score(X,\boldsymbol{U};G)$$

  - where

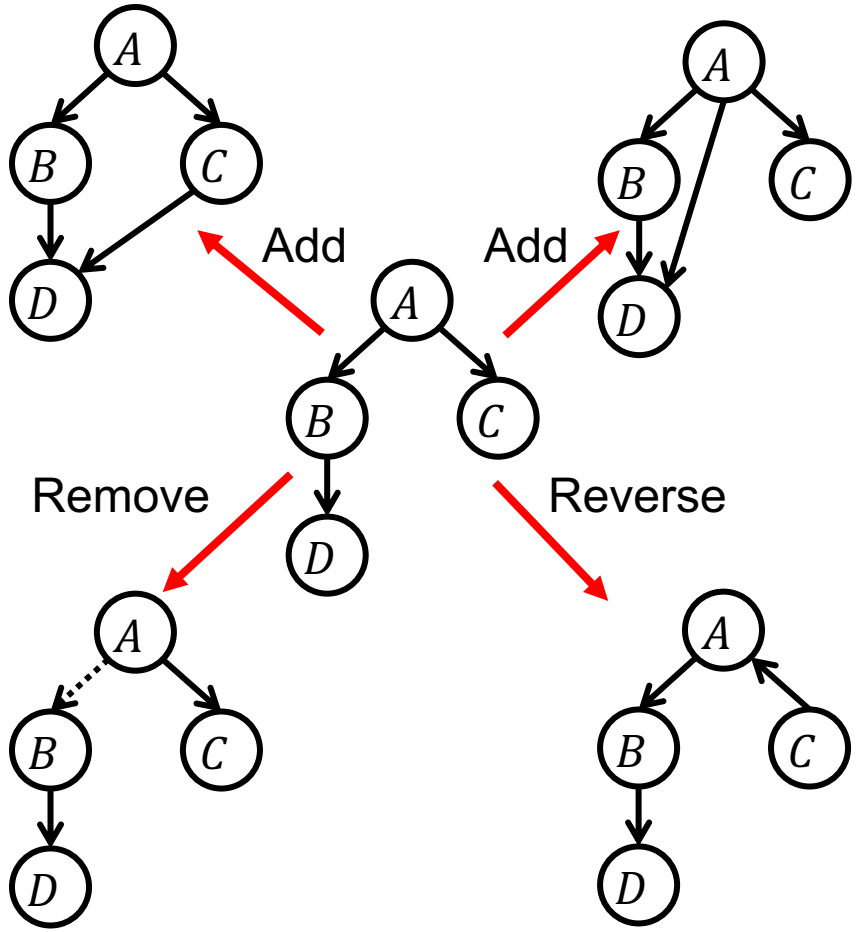
$$Score(X,\boldsymbol{U};\mathcal{D}) \stackrel{\text{def}}{=} -N \cdot H_{\mathcal{D}}\,(X|\boldsymbol{U}) - \psi(N) \cdot \|X\boldsymbol{U}\|$$

  - Note how the score is a sum of local scores, each for some family $X\boldsymbol{U}$

- The contribution of each family is split into two parts
  - One resulting from a decomposition of the log-likelihood component of the score
  - The other resulting from a decomposition of the penalty component
  - This decomposition enables several efficient implementations of heuristic and optimal search algorithms

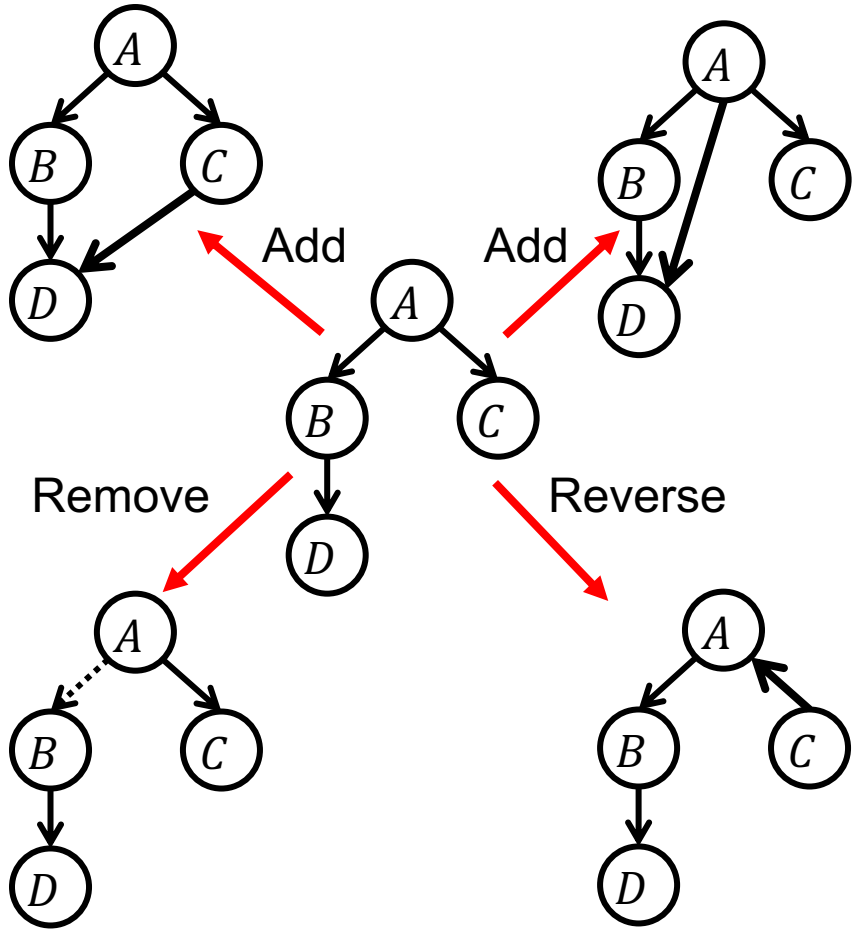
Local Search

- We can search for network structure by starting with some initial structure
  - Then modify it locally to increase its score
  - The initial structure can be chosen randomly, based on some prior knowledge, or can be a tree structure as discussed previously
- The local modifications to the structure are constrained to
  - Adding an edge
  - Removing an edge
  - Reversing an edge while ensuring the structure remains a DAG
- These local changes to the network structure also change the score
  - Possibly increasing or decreasing it
  - However, the goal is to commit to the change that increases the score the most
  - If none of the local changes can increase the score, the algorithm terminates and returns the current structure



Local Search

- Some observations about this local search algorithm
  - It is not guaranteed to return an optimal network structure
  - The only guarantee provided by the algorithm is that the returned structure is locally optimal
  - That is, no local change can improve its score
- This suboptimal behaviour can be improved by techniques such as *random restarts*
  - We repeat the local search multiple times, each time starting with a different network
  - Return the network with the best score across all repetitions

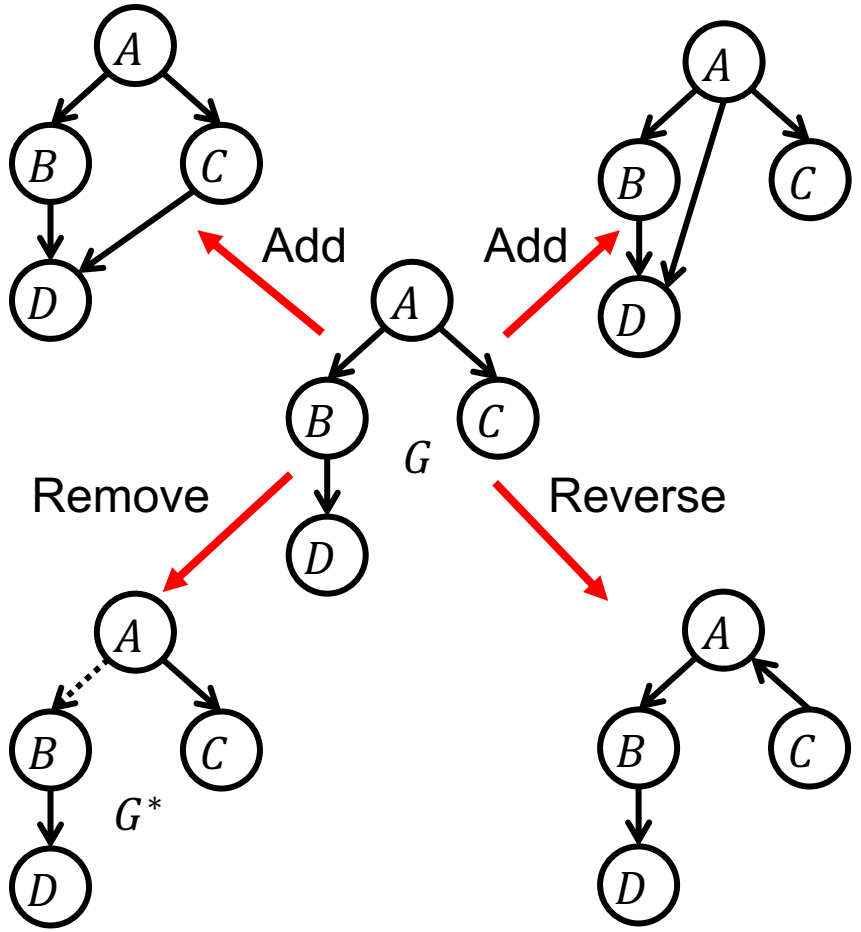


Local Search

- Another observation relates to score updating
  - Consider the networks G (centre) and G^* (after removing edge $A \rightarrow B$)
  - Since this change affects only the family of node B, we have

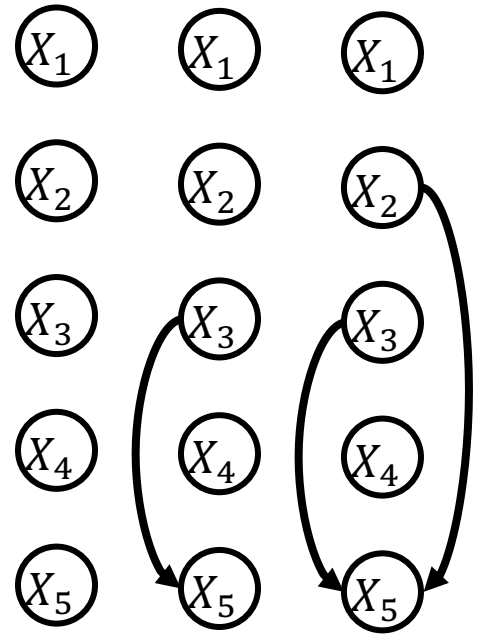
$$Score(G^*; \mathcal{D}) = Score(G; \mathcal{D}) - Score(B, A; \mathcal{D}) + Score(B; \mathcal{D})$$

  - That is, we discount the contribution of B's old family and add the contribution of B's new family
- More generally, adding or removing one edge changes only one family, while reversing an edge changes only two families
  - The score can always be updated locally as a result of the local network change



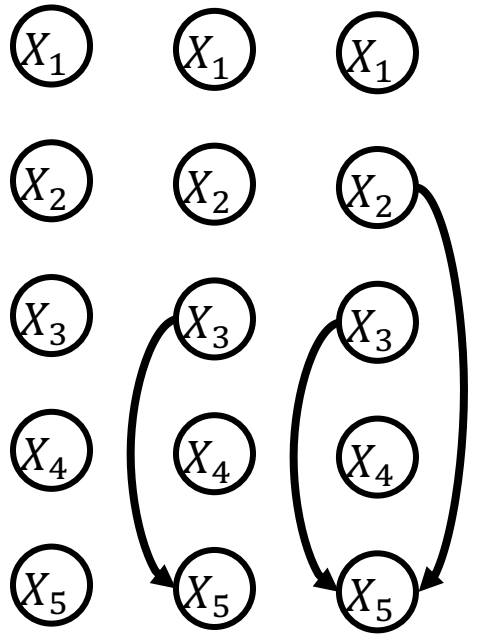
Constraining the Search Space

- A technique for reducing the search space size is to assume a total ordering on variables
  - Search only among network structures that are consistent with the chosen order
  - For instance, we use variable order $X_1, \ldots, X_n$
  - The search process tries to find for each variable X_i a set of parents $\boldsymbol{U}_i \subseteq X_1, \ldots, X_{i-1}$
- This technique also allows us to decompose the search into n independent problems
  - Each concerned with finding a set of parents for some network variable
  - That is, the search problem reduces to considering each variable X_i independently
  - And then finding a set of parents $\boldsymbol{U}_i \subseteq X_1, \ldots, X_{i-1}$ that maximize $Score(X_i, \boldsymbol{U}_i; \mathcal{D})$
- We discuss greedy and optimal methods for maximizing these local scores



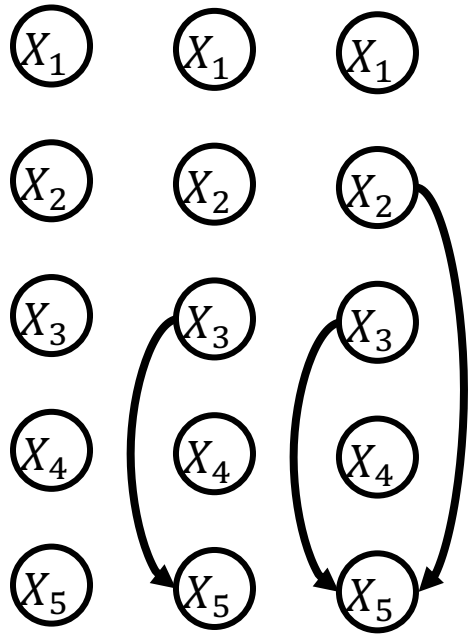
Greedy Search

- A well-known heuristic algorithm for optimizing a family score is known as *K3*
  - It starts with an empty set of parents
  - Successively add variables to the set one at a time
  - Until such additions no longer increase the score
- For example, we want to find a set of parents for X_5 from $X_1, \ldots X_4$
  - We start with $\boldsymbol{U}_5 = \emptyset$, and try to find X_i that maximizes
    $$Score(X_5, X_i; \mathcal{D}) \geq Score(X_5; \mathcal{D})$$
  - Suppose X_3 is such a variable, then $\boldsymbol{U}_5 = \{\mathrm{X}_3\}$ and we search for another variable X_i that maximizes
    $$Score(X_5, X_3, X_i; \mathcal{D}) \geq Score(X_5, X_3; \mathcal{D})$$
  - Suppose X_2 happens to be such a variable, then $\boldsymbol{U}_5 = \{\mathrm{X}_2, \mathrm{X}_3\}$
  - If no other variable can improve the score, then K3 terminates with $\boldsymbol{U}_5 = \{\mathrm{X}_2, \mathrm{X}_3\}$ as the parent set for X_5



Greedy Search

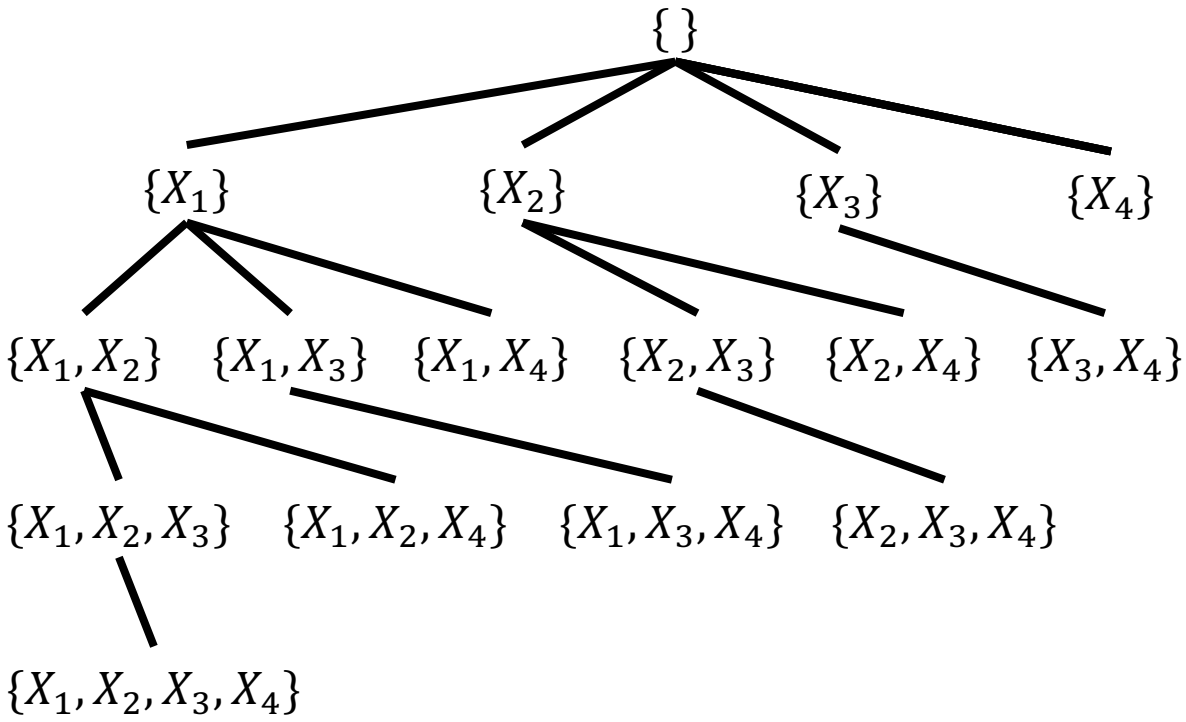
- K3 is not guaranteed to identify the optimal set of parents
  - That is, the one that maximizes $Score(X_i, \boldsymbol{U}_i; \mathcal{D})$
- Therefore, it is common to use the output of this algorithm as a starting point for other algorithms
  - Such as the local search algorithm discussed previously
  - Or the optimal search algorithm we discuss next



Optimal Search

- We discuss an optimal search algorithm for network structures
  - Based on branch-and-bound depth-first search
  - Like K3, it assumes a total order of variables $X_1, \ldots, X_n$
  - The search is restricted among structures consistent with this order
  - This allow us to decompose the search into n independent search problems

- For instance, consider the search for parents of X_5
  - The first level of the tree represents an empty set of parents $\boldsymbol{U}_5 = \{\}$
  - Each additional level corresponds by adding a single variable to each parent set while avoiding duplicate sets

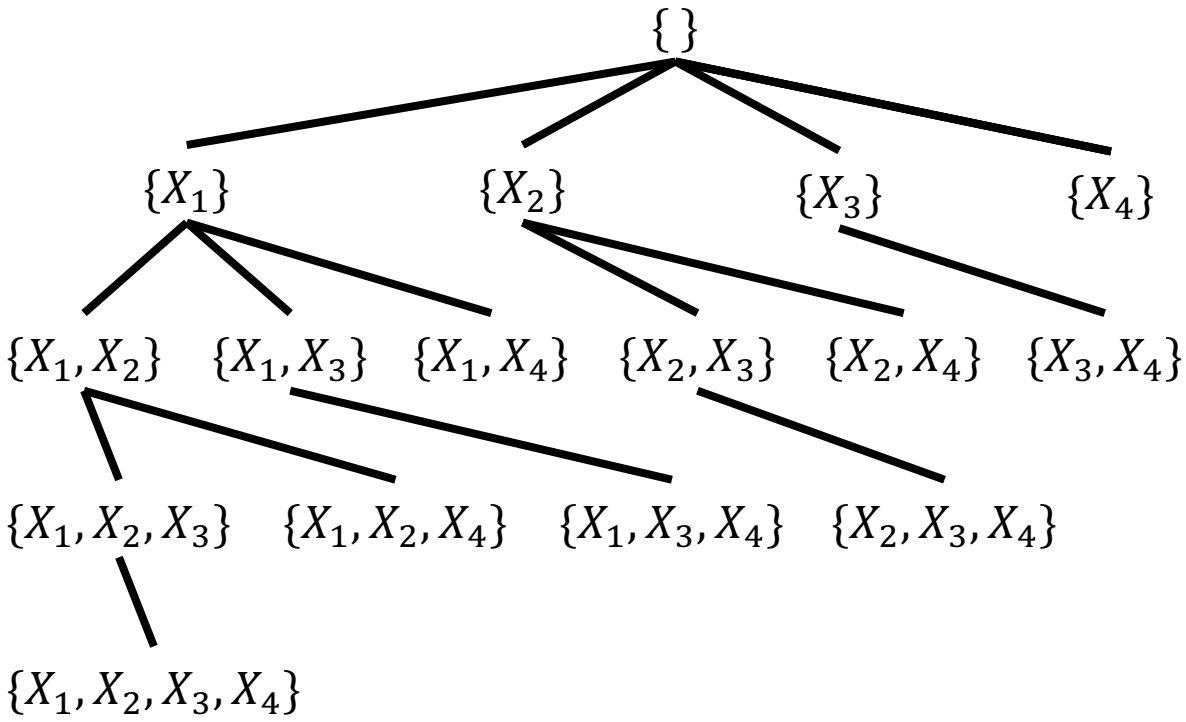


- The search tree has 2^{i-1} nodes, corresponding to the number of subsets we can choose from $X_1, \ldots, X_{i-1}$

Optimal Search: Pruning

- We can search this tree using depth-first search
  - Maintaining the score s of the best parent visited so far
  - When visiting node $\boldsymbol{U}_i$, we evaluate $Score(X_i, \boldsymbol{U}_i; \mathcal{D})$ and check if it is better than s
  - Depth-first guarantees every parent set is visited, leading us to identify the optimal parent set
  - However, such optimality comes at the expense of exponential complexity

- The complexity can be improved on average
  - If we use an upper bound on $Score(X_i, \boldsymbol{U}_i^*; \mathcal{D})$, where $\boldsymbol{U}_i \subseteq \boldsymbol{U}_i^*$
  - If the upper bound at node $\boldsymbol{U}_i$ is not better than the best score s, then we prune $\boldsymbol{U}_i$ and all nodes bellow it
  - The extend of pruning depends on the quality of the upper bound used

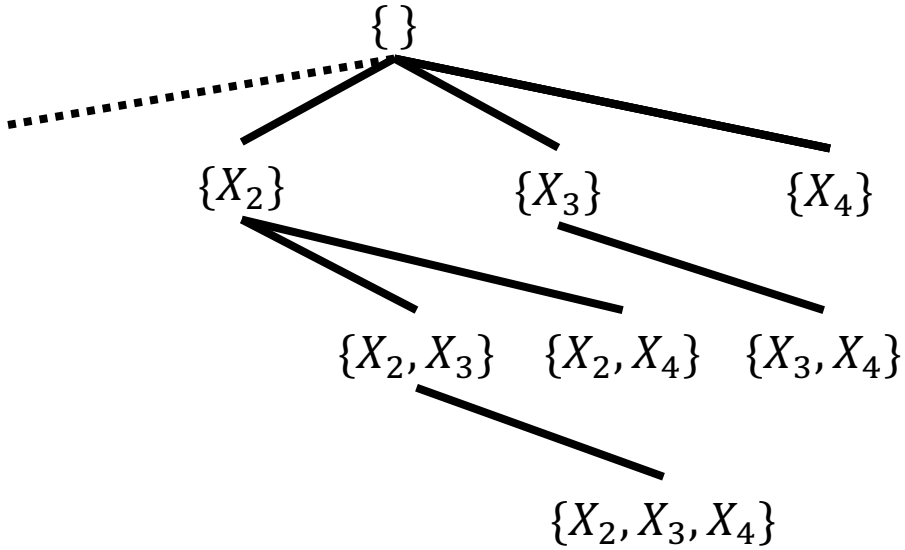


Optimal Search: Pruning

- For the MDL score, we can use the following upper bound
  - Let $\boldsymbol{U}_i$ be the parent set and let $\boldsymbol{U}_i^+$ be the largest parent set appearing bellow $\boldsymbol{U}_i$ in the search tree. If $\boldsymbol{U}_i^*$ is a parent set in the tree rooted at $\boldsymbol{U}_i$, then

$$MDL(X_i, \boldsymbol{U}_i^*; \mathcal{D}) \leq -N \cdot H_{\mathcal{D}}(X_i|\boldsymbol{U}_i^+) - \psi(N) \cdot \|X_i \boldsymbol{U}_i\|$$

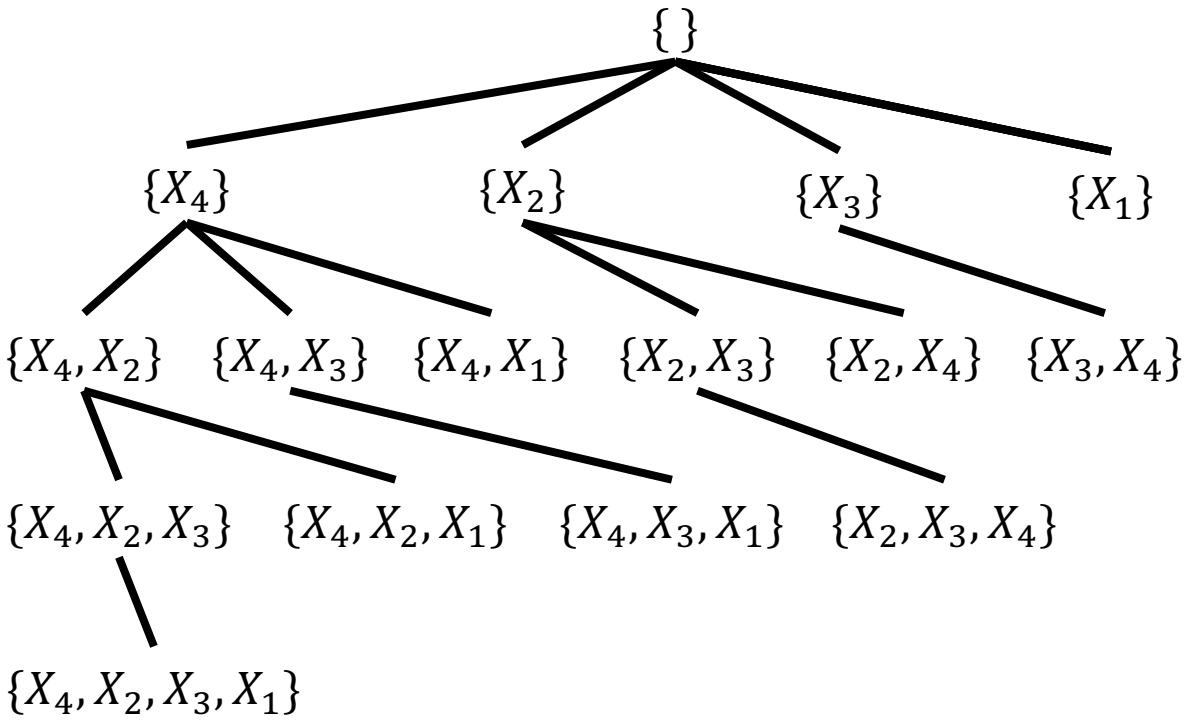
- This bound needs to be computed at each node $\boldsymbol{U}_i$
  - For example, at search node $\boldsymbol{U}_5 = \{X_2\}$, we get $\boldsymbol{U}_5^+ = \{X_2, X_3, X_4\}$
  - Moreover, $\boldsymbol{U}_5^*$ ranges over parent sets $\{X_2\}, \{X_2, X_3\}, \{X_2, X_4\}$ and $\{X_2, X_3, X_4\}$



Optimal Search: Variable Order

- Consider this search tree with different variable orders
  - Now variable X_4 appears more frequently on the first branch
  - Suppose variable X_4 tends to reduce the entropy of X_5 more than does X_1
  - Then we expect this search tree to visit parent sets with higher scores first
  - This may lead to more aggressive pruning
- Therefore, we prefer a tree that is expanded according to variable order $X_{k_1}, \ldots, X_{k_{i-1}}$, where

$$H(X_i|X_{k_1}) \leq \cdots \leq H(X_i|X_{k_{i-1}})$$



Structure Learning and Incomplete Data

- We have restricted our discussion on the search for network structures to complete datasets
  - The main reason is computational
  - The likelihood of a network structure does not admit a closed form when the dataset is incomplete
  - Also, it does not decompose into components

- Hence, algorithms for learning structures with incomplete data involve two searches
  - An outer search in the space of network structures
  - An inner search in the space of network parameters

Conclusion

- In this lecture, we studied approaches to learn the network structure from data
  - We focused on approaches that search for a maximum likelihood structure
- The first approach was restricted to tree structures
  - Therefore, each nodes had at most one parent
  - Our approach could find an optimal structure using maximum spanning trees
- The second approach extended structures to DAGs
  - Adding edges never decreases the log-likelihood of the structure
  - Therefore, unrestricted search to minimize conditional entropy leads to complex structures and overfitting
  - We reviewed a class of scoring functions that ally data fitness and model complexity
- When searching for a DAG, we studied two main strategies
  - Local search based on hill climbing
  - Constrained search methods that assume a variable order to reduce the search space
- Task
  - Read chapter 17