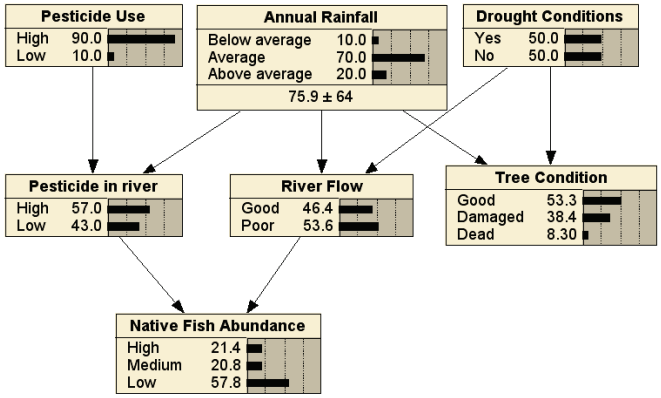
ICS Department of Information & Computing Sciences

Universiteit Utrecht

ALGORITHMS FOR DECISION SUPPORT

Bayesian networks



Guest Lecturer: Silja Renooij

(Thanks to Thor Whalen for kindly contributing to these slides)

Probabilistic Independence

$L =$ outcome of throwing dice I
$R =$ outcome of throwing dice II

$P(L) = \{P(L = 1), P(L = 2), \ldots, P(L = 6)\}$
$P(R) = \ldots$

$$P(L = 2,\ R = 3) = \ P(L = 2) \cdot P(R = 3)$$

Events '$L = 2$' and '$R = 3$' are independent.

In fact, this hold regardless of the specific outcomes
$\Longrightarrow$ variables L and R are independent.

Conditional (In)dependence

$L =$ outcome of throwing dice I
$R =$ outcome of throwing dice II
$C =$ colour of dice
$P(C) = \{P(C = \text{red}), P(C = \text{white})\}$

$$P(L = 2,\ R = 3 \mid C = \text{red}) =$$
$$= \quad P(L = 2 \mid C = \text{red}) \cdot P(R = 3 \mid C = \text{red})$$

Events '$L = 2$' and '$R = 3$' are independent given that '$C = \text{red}$' is known for sure.

In fact, variables L and R are independent given C.

This also holds given e.g. a variable F that represents the fairness of the dice!

Conditional (In)dependence

$L =$ outcome of throwing dice I
$R =$ outcome of throwing dice II
$S = L + R$

$P(S) = \{P(S = 2), P(S = 3), \ldots, P(S = 12)\}$

$$P(L = 2,\ R = 3 \mid S = 6) =$$

$$= \cancel{P(L = 2 \mid S = 6) \cdot P(R = 3 \mid S = 6)}$$

$$= P(L = 2 \mid S = 6,\ R = 3) \cdot P(R = 3 \mid S = 6)$$

'$L = 2$' and '$R = 3$' are not independent given '$S = 6$'.
$\Longrightarrow$ variables L and R are not independent given S.

Chain rule & independence

Any joint distribution over a set of stochastic variables $\boldsymbol{X} = \{X_1, \ldots X_n\}$ can be factorised (chain rule):

$$P(\boldsymbol{X}) = \prod_{i=1}^{n} P(X_i \mid \bigcap_{k=1}^{i-1} X_k)$$

e.g. $P(X_1, \ldots, X_4) =$

$$P(X_4 \mid X_3, X_2, X_1) \cdot P(X_3 \mid X_2, X_1) \cdot P(X_2 \mid X_1) \cdot P(X_1)$$

(Conditional) independence is now important, since it

- reduces size of conditioning sets (space efficiency)
- simplifies computation of probabilities (time efficiency)

Independence & space/time complexity

Consider a joint distribution over the outcomes of 10 dice:

$$
\begin{aligned}
P(D_1, \ ..., \ D_{10}) &= \\
&= P(D_1 \mid D_2, \ ..., \ D_{10}) \cdot ... \cdot P(D_9 \mid D_{10}) \cdot P(D_{10}) \\
&= P(D_1) \cdot ... \cdot P(D_{10})
\end{aligned}
$$

A complete specification of the distribution requires:

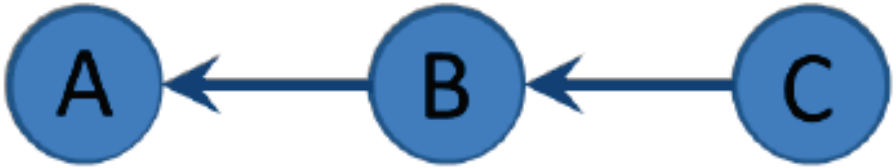
- no independence: $6^{10} \sim 60$ million probabilities
- independence: $6 \cdot 10 = 60$ probabilities

Computing e.g. $P(D_1 = 6, \ D_3 = 4)$ requires

- no independence: summing ~ 1.7 million probabilities
- independence: 1 multiplication

Efficient representation of independence

One way is to use a directed acyclic graph (DAG):



A is independent of C given B

$$\begin{aligned} \Longrightarrow P(A,\ B,\ C) &= P(A \mid B,\ C) \cdot P(B \mid C) \cdot P(C) \\ &= P(A \mid B) \cdot P(B \mid C) \cdot P(C) \end{aligned}$$

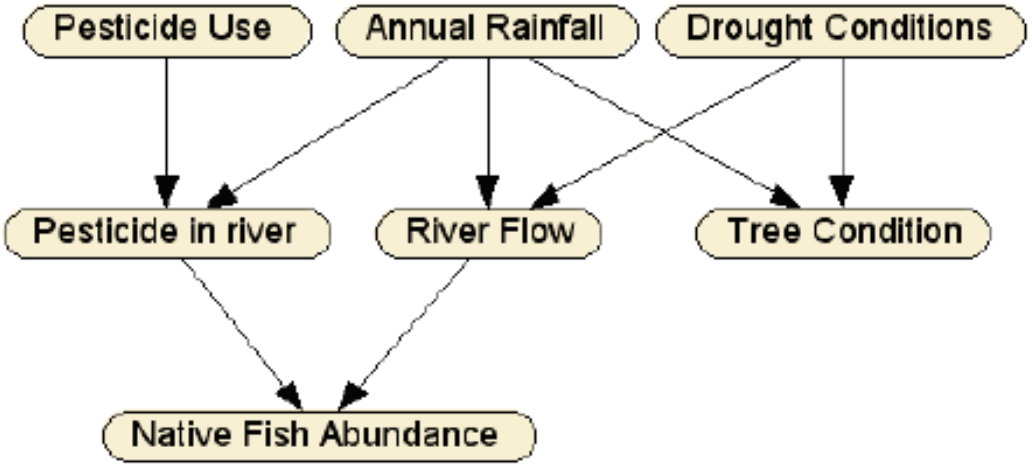
$\Longrightarrow$ efficient representation of joint distribution over a set of stochastic variables $\boldsymbol{X} = \{X_1, \ldots X_n\}$:

$$P(\boldsymbol{X}) = \prod_{i=1}^{n} P(X_i \mid par_G(X_i))$$

$\uparrow$ parents of X_i in the graph

Bayesian network (BN)

- model $\mathcal{B}$ of discrete joint probability distribution $P(\boldsymbol{X})$
- qualitative part: DAG $G = (\boldsymbol{V} = \boldsymbol{X}, \boldsymbol{A})$ of independence relation
- quantitative part: conditional distributions $P(X_i \mid par_G(X_i))$



P(Pesticides)	
High	Low
90	10

P(Annual Rainfall)		
BelowAvg	Avg	AboveAvg
10	70	20

		P(NativeFishAbundance | Pesticides, RiverFlow)		
Pesticides	RiverFlow	High	Medium	Low
High	Good	20	40	40
High	Poor	1	10	89
Low	Good	80	15	5
Low	Poor	5	15	80

Bayesian network queries

Consider a BN defined over variables $\boldsymbol{X}$. Let $\boldsymbol{e}$ and $\boldsymbol{h}$ denote value assignments to disjoint $\boldsymbol{E} \subset \boldsymbol{X}$ and $\boldsymbol{H} \subset \boldsymbol{X}$.

Typical queries posed to a BN are:

- $\arg\max_h P(\mathbf{H} = h \mid \mathbf{E} = \mathbf{e}) = \arg\max_h P(\mathbf{H} = h, \mathbf{E} = \mathbf{e})$
  - Most probable explanation (MPE) if $\boldsymbol{H} \cup \boldsymbol{E} = \boldsymbol{X}$
  - Maximum a-posteriori probability assignment (MAP) if $\boldsymbol{H} \cup \boldsymbol{E} \subset \boldsymbol{X}$
- $P(H = h \mid \mathbf{E} = \mathbf{e}) = \dfrac{P(H = h, \mathbf{E} = \mathbf{e})}{P(\mathbf{E} = \mathbf{e})} \propto P(H = h, \mathbf{E} = \mathbf{e})$
  - Inference (typically $\boldsymbol{H}$ equals a single X_i)

Complexity of queries (decision versions)

all NP-hard

- MPE: NP-complete
- MAP: NP^{PP}-complete (NP with PP-oracle; NP $\subseteq$ PP)
- Inference: PP-complete

Complexity is due to optimisation (MPE/MAP) and marginalisation ('summing out'):

let $\boldsymbol{X} = \boldsymbol{H} \cup \boldsymbol{I} \cup \boldsymbol{E}$, with $\boldsymbol{I}$ and/or $\boldsymbol{E}$ possible empty, then

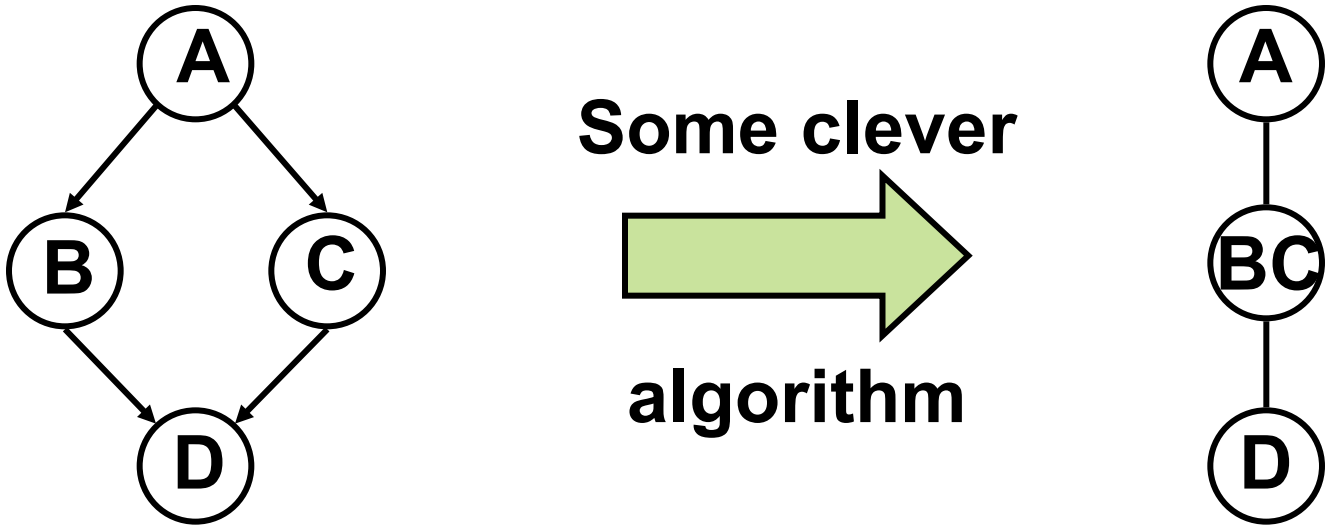
$$P(\boldsymbol{H} = \boldsymbol{h}, \boldsymbol{E} = \boldsymbol{e}) = \sum_{\boldsymbol{I}=\boldsymbol{i}} P(\boldsymbol{H} = \boldsymbol{h}, \boldsymbol{I} = \boldsymbol{i}, \boldsymbol{E} = \boldsymbol{e})$$

(See Kwisthout & DeCampos (2015), Lecture notes: Computational Complexity of Bayesian Networks)

Inference algorithms

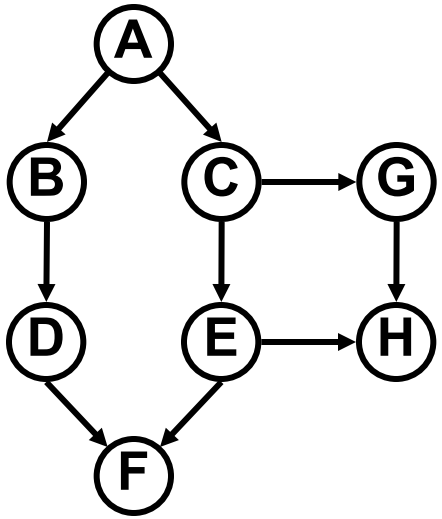
- Exact inference
  - Variable elimination (VE)
  - Message passing (Pearl)
  - Junction tree propagation (aka join tree/Hugin prop.)
- Approximate inference
  - Loopy belief propagation
  - Stochastic sampling (various Monte Carlo methods)
  - (!) in general, approximation (within a guaranteed margin of error) does not reduce complexity of inference

Idea behind the Junction tree algorithm



Many problems that are *hard* on arbitrary graphs are *easy* on tree-like structures.

Or more specifically….



Bayesian Network

- one-dim. stochastic variables
- conditional probabilities

Secondary Structure: Junction Tree

- multi-dim. stochastic variables
- cluster ‘potentials’

Let's take a couple of steps back…

Suppose we are interested in $P(X_1)$, then compute:

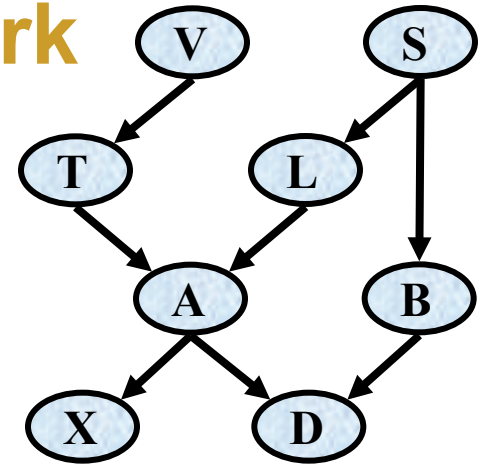
$$P(X_1) = \sum_{X_2} \sum_{X_3} \dots \sum_{X_n} P(\boldsymbol{X}) =$$

$$= \sum_{X_2} \sum_{X_3} \dots \sum_{X_n} \prod_i P(X_i \,|par(X_i))$$

Less naïve:

- Variable elimination (VE), i.e. iteratively:
  - Move all irrelevant terms outside of innermost sum
  - Perform innermost sum, getting a new term
  - Insert the new term into the product

VE example in "Asia" network



We are interested in $P(D)$

- Need to sum out (eliminate):

  V, S, X, T, L, A, B

Initial factors:

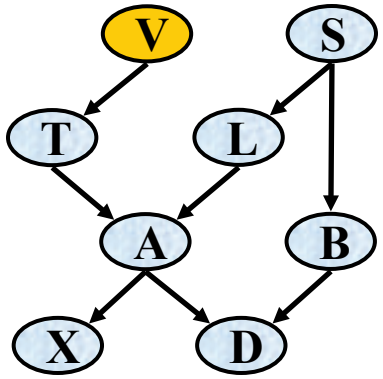
$$P(V)\,P(S)\,P(T\,|\,V)\,P(L\,|\,S)\,P(B\,|\,S)\,P(A\,|\,T,L)\,P(X\,|\,A)\,P(D\,|\,A,B)$$

Brute force:

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

But let's try something more elegant…

VE example continued



Eliminate variables in order:

$$V \rightarrow S \rightarrow X \rightarrow T \rightarrow L \rightarrow A \rightarrow B$$

Combine all initial factors using V:

$$\boxed{P(V)}\, P(S)\, \boxed{P(T \mid V)}\, P(L \mid S)\, P(B \mid S)\, P(A \mid T, L)\, P(X \mid A)\, P(D \mid A, B)$$

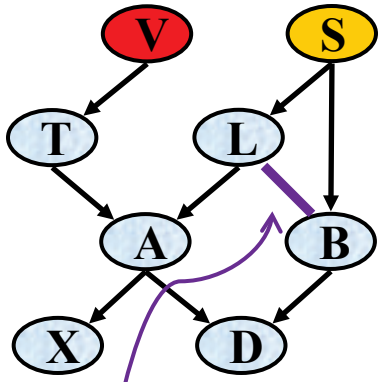
$$f_V(T) = \sum_v P(v) P(T \mid v)$$

[Note: although $f_V(T) = P(T)$, in general the result of elimination is not necessarily a probability term]

$$\Rightarrow f_V(T) P(S) P(L \mid S) P(B \mid S) P(A \mid T, L) P(X \mid A) P(D \mid A, B)$$

$f_V(T)$ more or less 'joins' T and V

VE example cntnd



Eliminate variables in order:

$$V \rightarrow S \rightarrow X \rightarrow T \rightarrow L \rightarrow A \rightarrow B$$

Combine initial factors for this iteration:

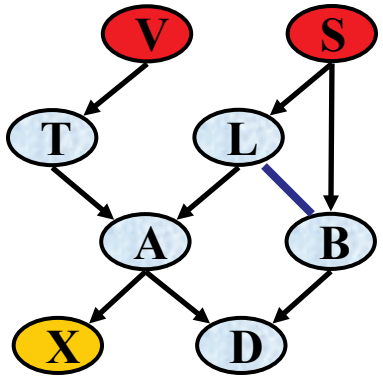
$$f_V(T)\,P(S)\,P(L \mid S)\,P(B \mid S)\,P(A \mid T, L)\,P(X \mid A)\,P(D \mid A, B)$$

$$f_S(B, L) = \sum_s P(s)P(B \mid s)P(L \mid s)$$

[Note: result of elimination may be a function of several variables; L and B thus become 'connected']

$$\Rightarrow f_V(T)\,f_S(B, L)\,P(A \mid T, L)\,P(X \mid A)\,P(D \mid A, B)$$

VE example cntnd



Eliminate variables in order:

$$V \rightarrow S \rightarrow X \rightarrow T \rightarrow L \rightarrow A \rightarrow B$$

Combine factors for this iteration:

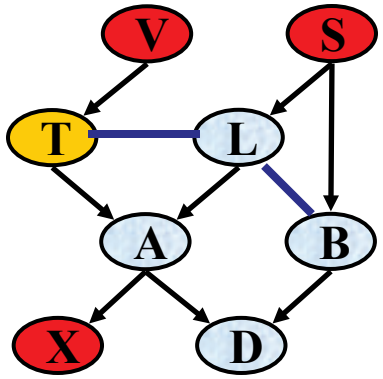
$$f_V(T) f_S(B,L) P(A \mid T,L) P(X \mid A) P(D \mid A,B)$$

$$f_X(A) = \sum_x P(x \mid A)$$

[Note: $f_X(a) = 1$ for all values a of A]

$$\Rightarrow f_V(T) f_S(B,L) f_X(A) P(A \mid T,L) P(D \mid A,B)$$

VE example cntnd



Eliminate variables in order:

$$V \rightarrow S \rightarrow X \rightarrow T \rightarrow L \rightarrow A \rightarrow B$$

Combine factors for this iteration:

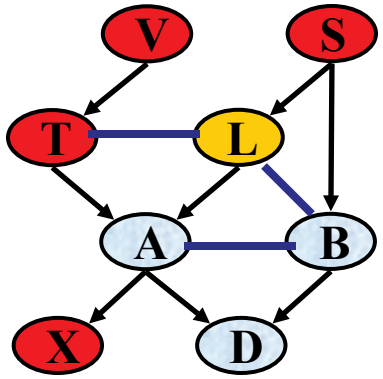
$$f_V(T) f_S(B,L) f_X(A) P(A \mid T,L) P(D \mid A,B)$$

$$f_T(A,L) = \sum_t f_V(t) P(A \mid t,L)$$

[Note: factors f can include other f's; this factor 'joins' T and L]

$$\Rightarrow f_S(B,L) f_X(A) f_T(A,L) P(D \mid A,B)$$

VE example cntnd



Eliminate variables in order:

$$V \rightarrow S \rightarrow X \rightarrow T \rightarrow L \rightarrow A \rightarrow B$$

Combine factors for this iteration:

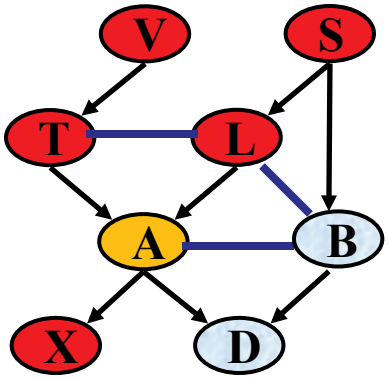
$$f_S(B,L) f_X(A) f_T(A,L) P(D \mid A,B)$$

$$f_L(A,B) = \sum_l f_S(B,l) f_T(A,l)$$

[Note: 'joins' A and B]

$$\Rightarrow f_L(A,B) f_X(A) P(D \mid A,B)$$

VE example cntnd



Eliminate variables in order:

$$V \rightarrow S \rightarrow X \rightarrow T \rightarrow L \rightarrow A \rightarrow B$$

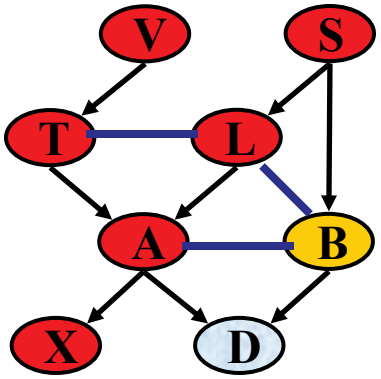
Combine factors for this iteration:

$$f_L(A,B)\, f_X(A)\, P(D \mid A,B)$$

$$f_A(B,D) = \sum_a f_L(a,B)\, f_X(a)\, P(D \mid a,B)$$

$$\Rightarrow f_A(B,D)$$

VE example cntnd



Eliminate variables in order:

$$V \rightarrow S \rightarrow X \rightarrow T \rightarrow L \rightarrow A \rightarrow B$$

Combine factors for this iteration:

$$f_A(B,D)$$

$$f_B(D) = \sum_b f_A(b,D)$$

$$\Rightarrow f_B(D)$$

VE intermediate factors

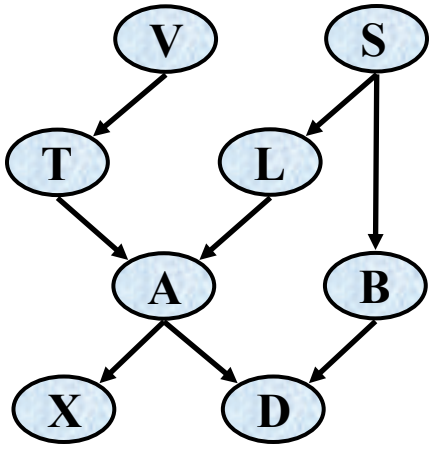
In our previous example:

$$V \rightarrow S \rightarrow X \rightarrow T \rightarrow L \rightarrow A \rightarrow B$$

$$f_V(T)$$
$$f_S(B,L)$$
$$f_X(A)$$
$$f_T(A,L)$$
$$f_L(A,B)$$
$$f_A(B,D)$$
$$f_B(D)$$

With a different ordering:

$$A \rightarrow B \rightarrow X \rightarrow T \rightarrow V \rightarrow S \rightarrow L$$



$$g_A(L,T,D,B,X)$$
$$g_B(L,T,A,X,S)$$
$$g_X(L,T,D,S)$$
$$g_T(L,T,S,V)$$
$$g_V(L,D,S)$$
$$g_S(L,D)$$
$$g_L(D)$$

Complexity is exponential in the size of these factors!

Notes about VE

- Actual computation is done in the elimination steps
- Computation depends on the order of elimination
- For each query we need to compute everything again!
  - Many redundant calculations

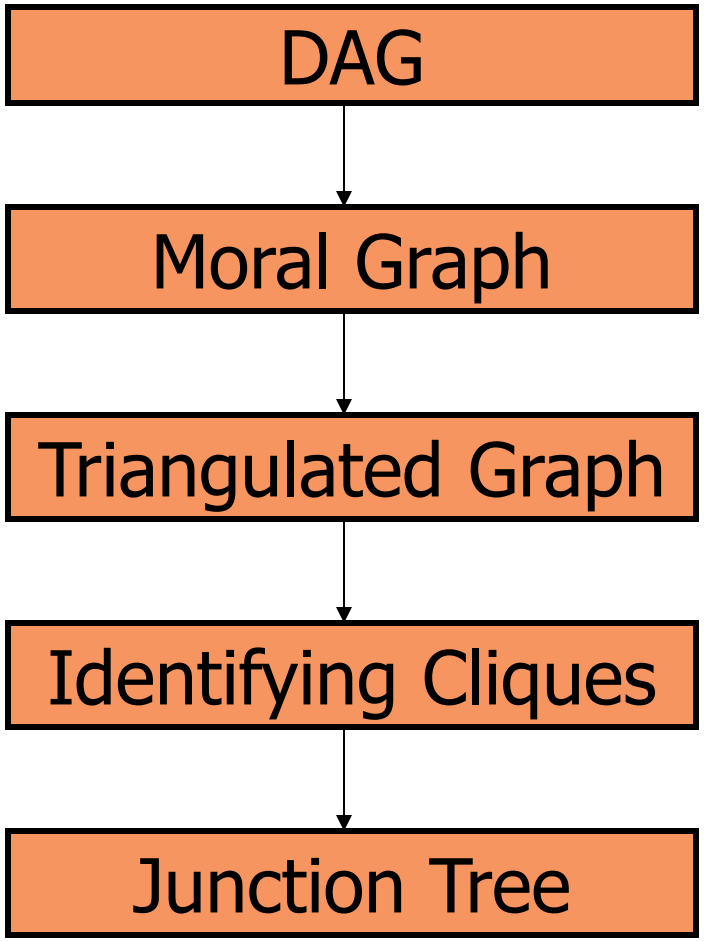
Junction Trees

- Redundant calculations VE can be avoided by ‘generalising’ to the junction tree (JT) algorithm

  (introduced by Lauritzen & Spiegelhalter, 1988)

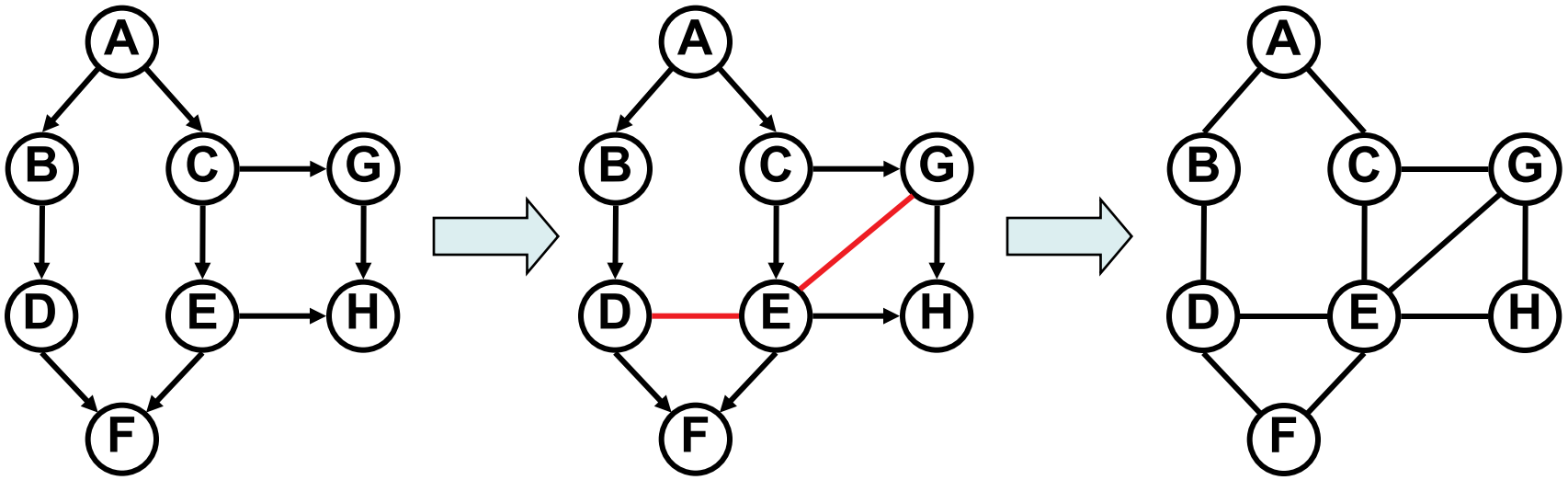
- The JT algorithm compiles a class of elimination orders into a data structure that supports the computation of all possible queries.

Building a Junction Tree



Step 1: Moralization

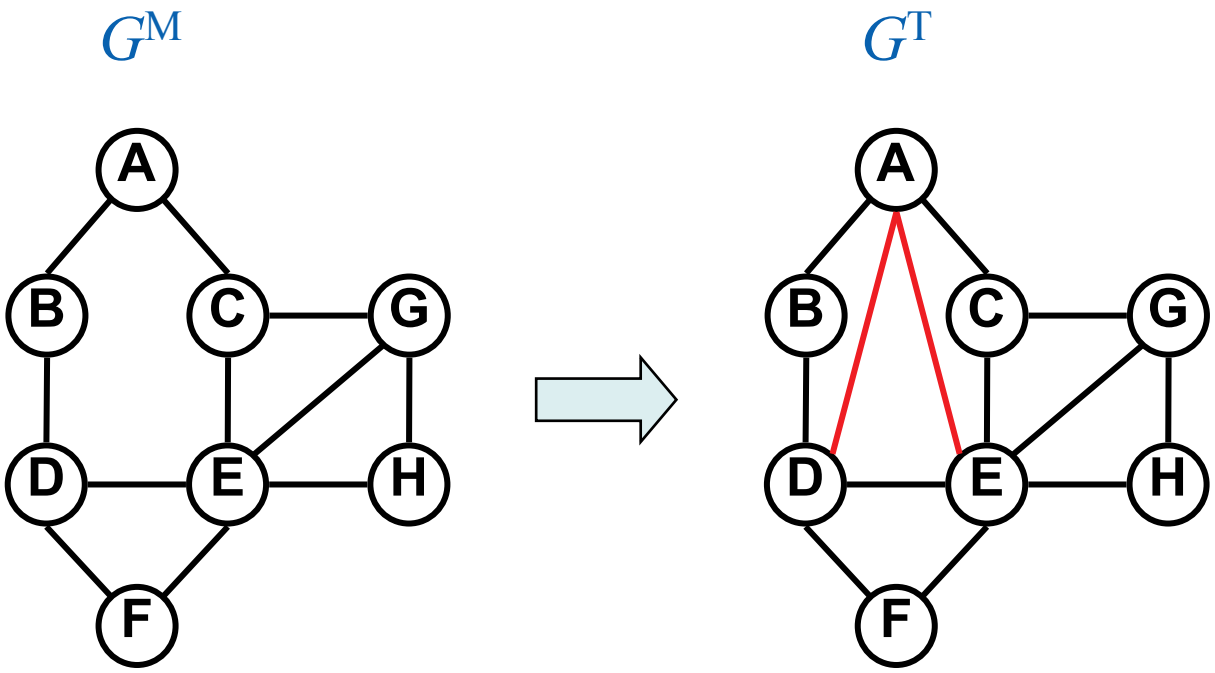
$G = (V, A)$



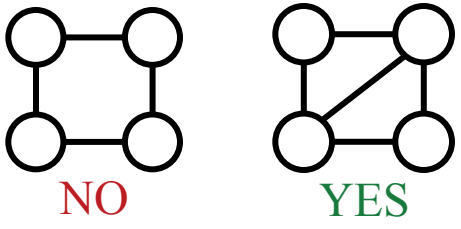
G^{M}

1. For all $Z \in V$:
   - For all $X, Y \in par(Z)$ add an edge X—Y.
2. Undirect all edges.

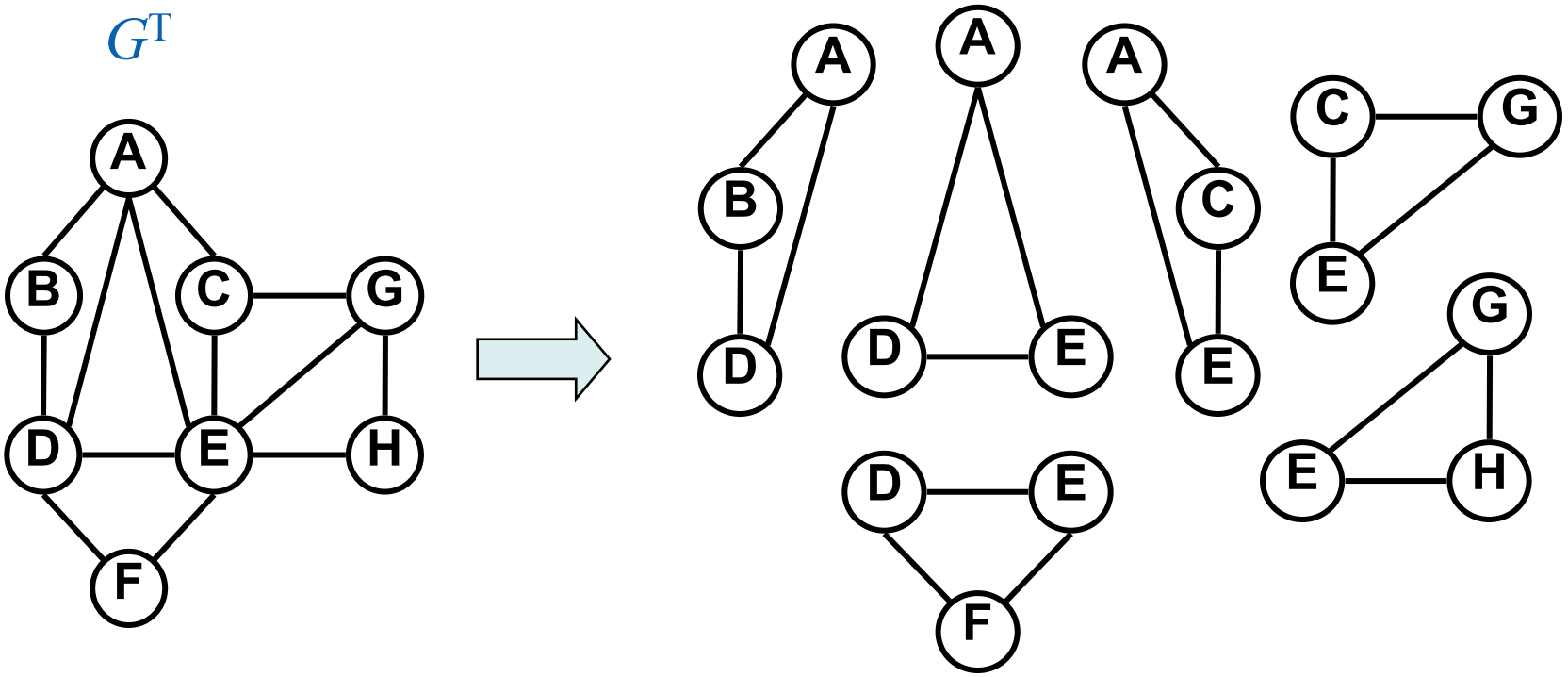
Step 2: Triangulation



Add edges to G^M such that there is no cycle with length ≥ 4 that does not contain a chord.

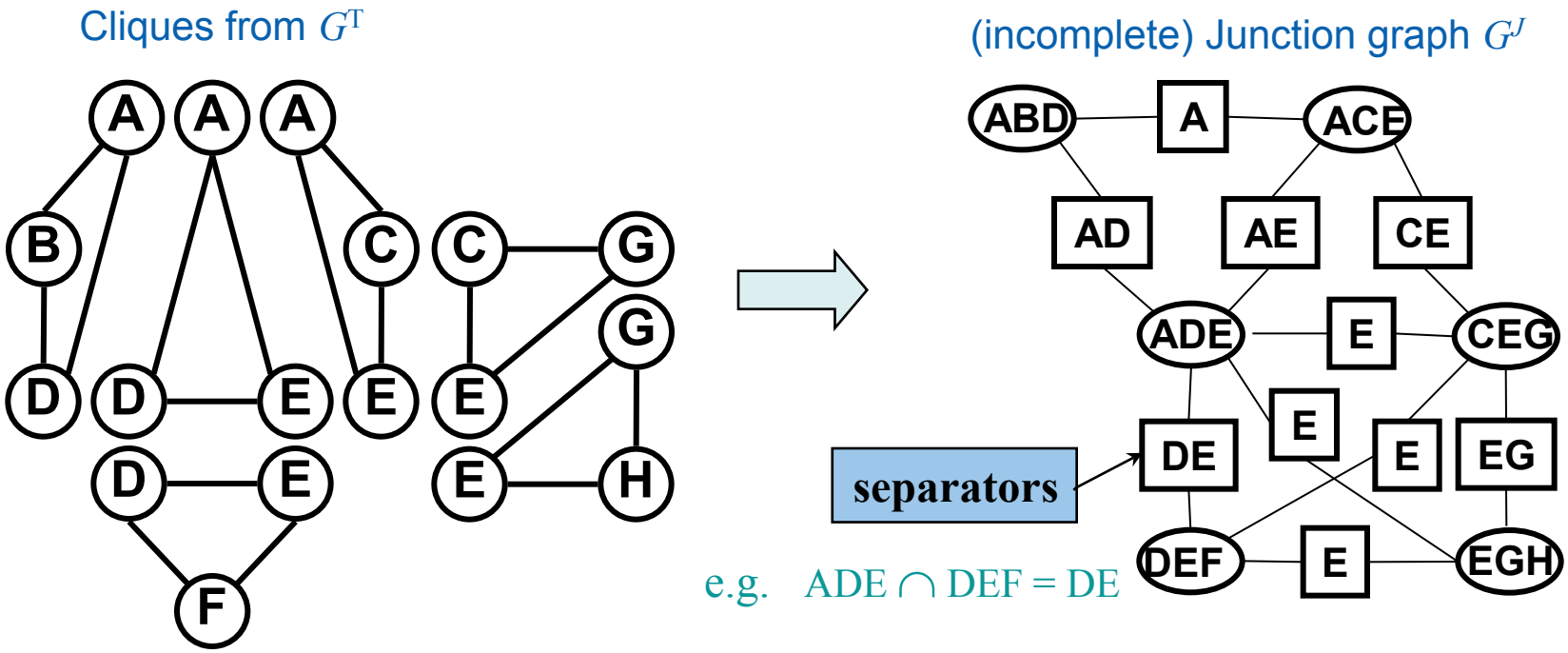


Step 3: Identifying Cliques



All maximal cliques
(complete subgraphs) of G^T

Step 4-I: Junction Graph



- A junction graph for an undirected graph G is an undirected, labeled graph.
- The nodes are the cliques in G.
- If two cliques intersect, they are joined in the junction graph by an edge labeled with their intersection.

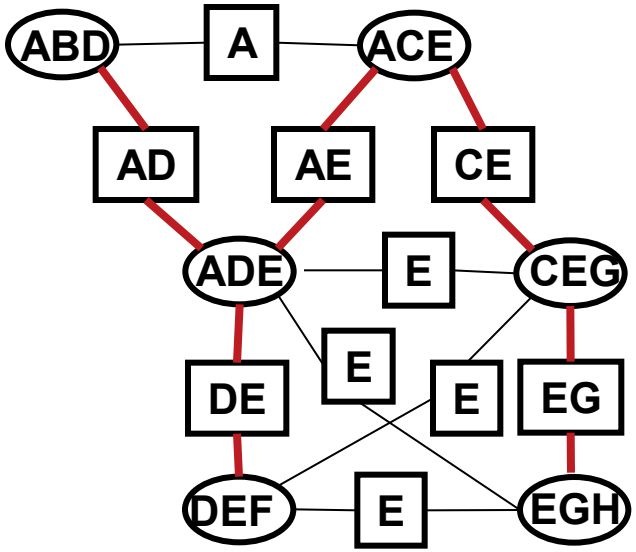
Step 4-II: Junction Tree

A junction tree is a sub-graph of the junction graph that

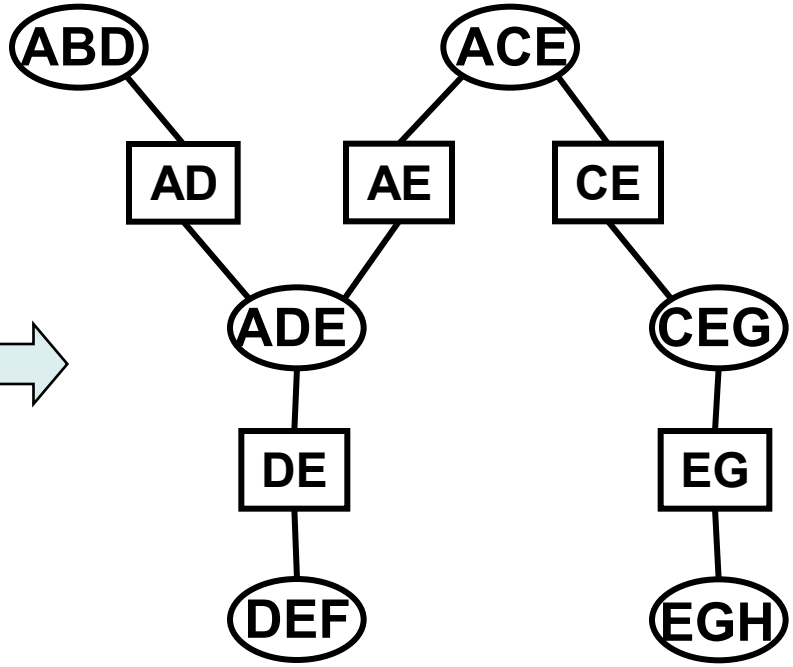
- Is a tree
- Contains all the cliques (spanning tree)
- Satisfies the ***running intersection*** property:

for each pair of nodes X, Y, all nodes on the path between X and Y contain $X \cap Y$

Junction graph G^J (incomplete)



Junction tree G^{JT}

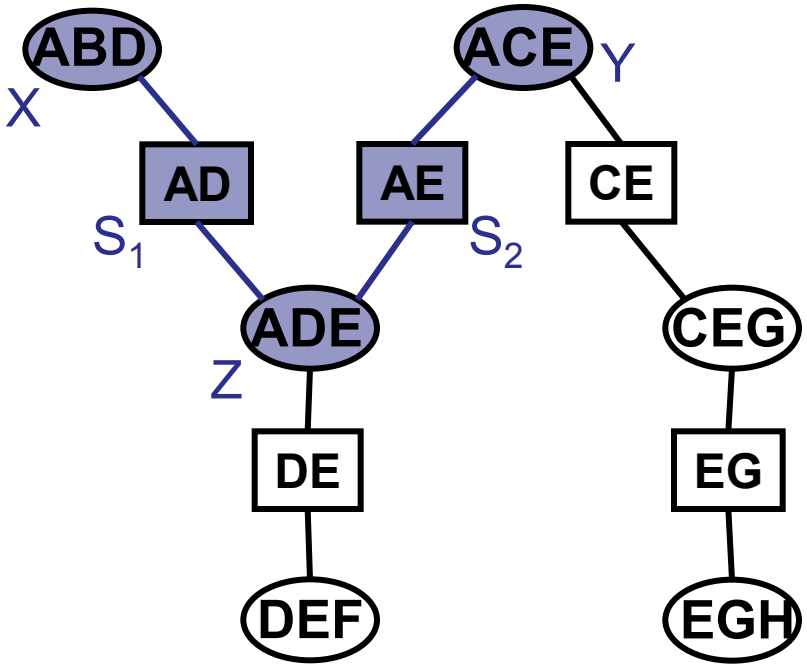


Running intersection?

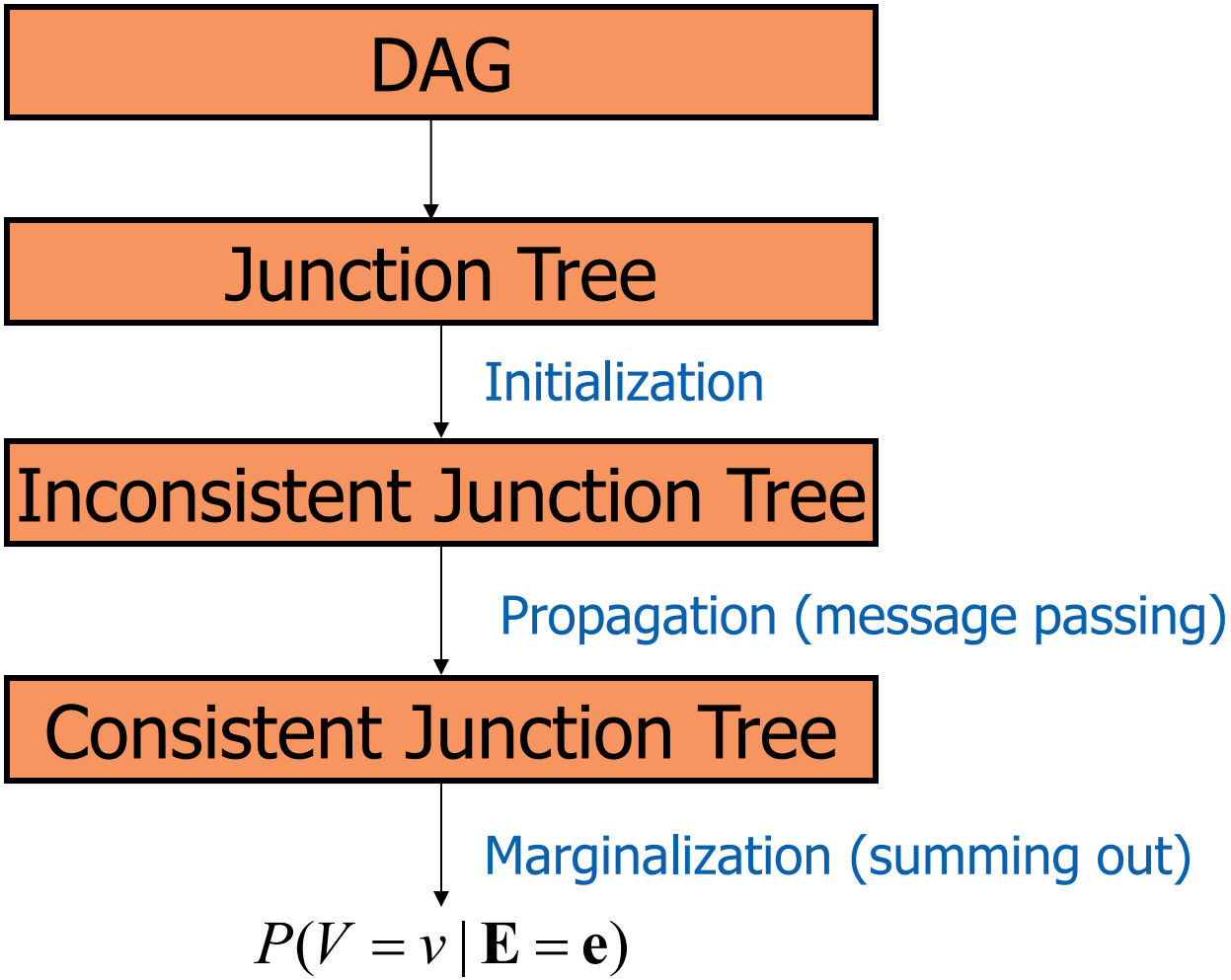
All cliques Z and separators S along the path between any two nodes X and Y contain the intersection $X \cap Y$.

Ex: $X=\{A,B,D\}$, $Y=\{A,C,E\} \Rightarrow X \cap Y=\{A\}$

$C=\{A,D,E\} \supseteq \{A\}$, $S_1=\{A,D\} \supseteq \{A\}$, $S_2=\{A,E\} \supseteq \{A\}$



Using a Junction Tree for inference



Step 1: Initialization

- For each (conditional) distribution from the BN, create a node potential:

$$P(X_i \mid par(X_i)) \Rightarrow \phi_i\,(X_i, par(X_i))$$

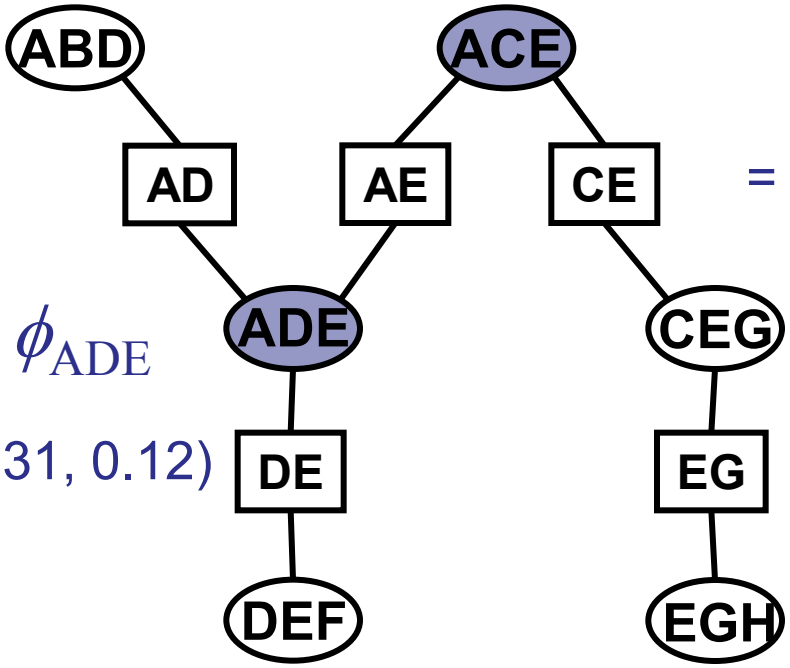
- Assign each node potential to a single clique C, for which

$$(\{X_i\} \cup par(X_i)) \subseteq (\text{variables in } C)$$

- The clique potential Φ_C for C is the product of its assigned node potentials

Marginalisation and Inconsistency

- Potentials are not necessarily joint probability distributions, i.e.: $\phi_X \neq P(\boldsymbol{X})$
- Potentials in the junction tree can be inconsistent, i.e. computing a marginal $P(X_i)$ from different cliques can give different results:



$$P(A) = \sum_{ce} \phi_{\mathrm{ACE}}$$

= (0.12, 0.33, 0.11, 0.03)

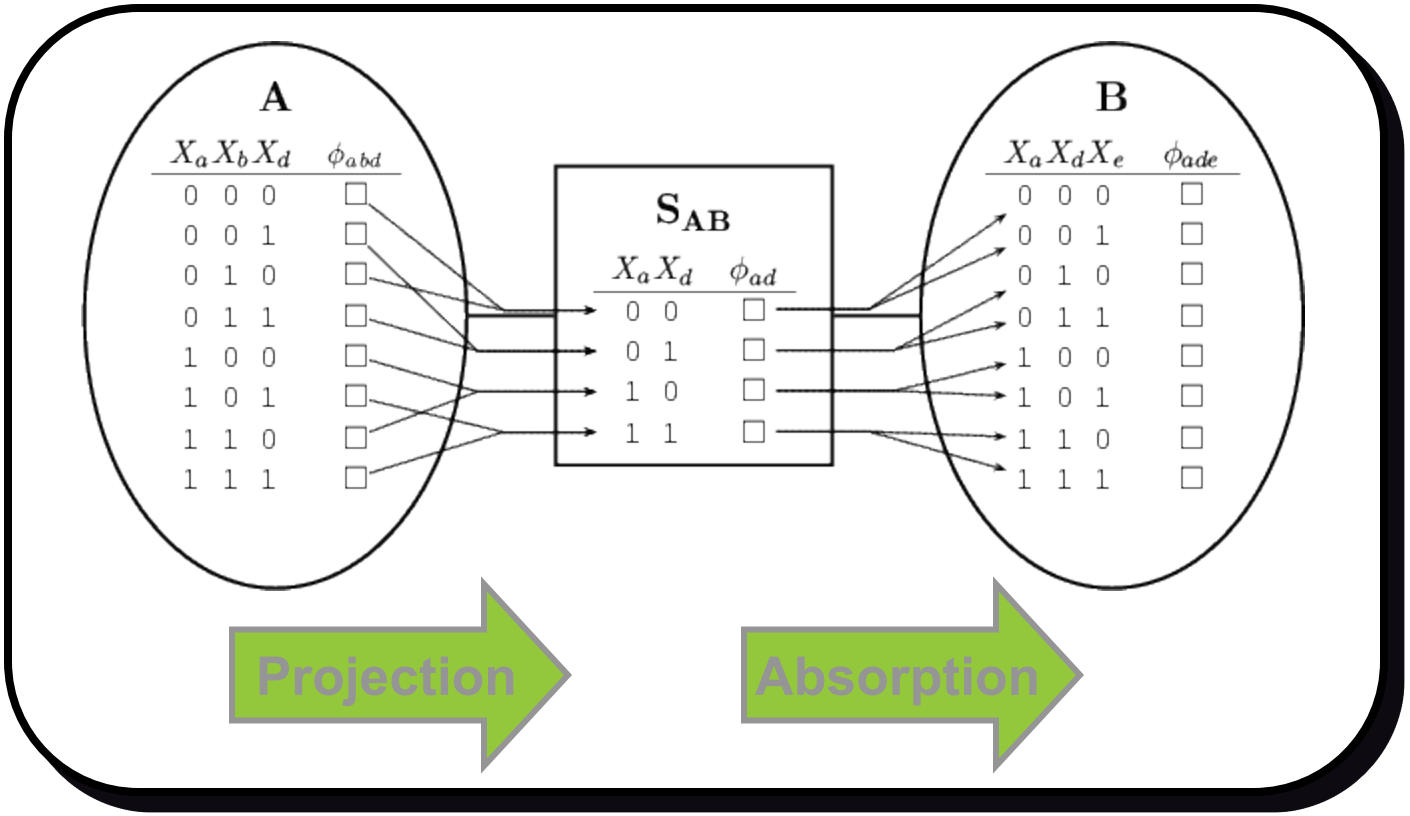
$$P(A) = \sum_{de} \phi_{\mathrm{ADE}}$$

= (0.02, 0.43, 0.31, 0.12)

Propagating potentials: idea

Message Passing from clique A to clique B

1. Project the potential of A into separator S_{AB}
2. Absorb the potential of separator S_{AB} into B

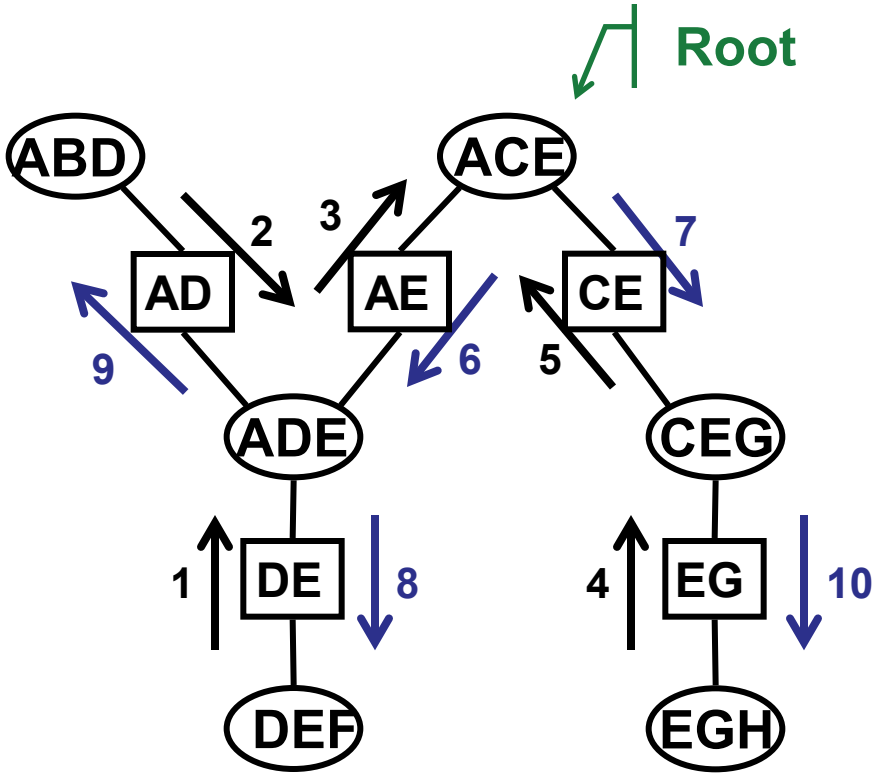


Global propagation: idea

1. Choose a root

2. COLLECT-EVIDENCE
(messages 1-5: leafs to root.
NB corresponds with a
perfect elimination order!)

3. DISTRIBUTE-EVIDENCE
(messages 6-10: root to leafs)



After global propagation, potentials *are* consistent and marginalisation gives correct results.

Message passing

Message passing in the junction tree resembles Pearl's λ-π -message passing algorithm for singly connected graphs.

Do you want to know how and why that works?

Ask those doing the Probabilistic Reasoning course!

Back to complexity

Computing probabilities from a BN with graph G, with n nodes and tree-width w, requires $O(n \exp(w))$ time.

- tree-width of G = minimum width over all possible junction trees of G
- width of a junction tree = size of the largest clique, minus 1

→ inference and MPE can be solved in polynomial time on networks of bounded tree-width!

(Only MAP remains NP-complete even on graphs with $w \leq 2$)

Summary & More

We've seen that:

- Bayesian networks efficiently represent a joint probability distribution.
- The junction tree propagation algorithm elegantly combines elimination orders from VE and message passing alike Pearl.

We haven't discussed how to:

- triangulate a graph
- construct a Junction Tree from a junction graph
- exactly compute probabilities from it

Curious? A bit more can be found in the bonus slides…

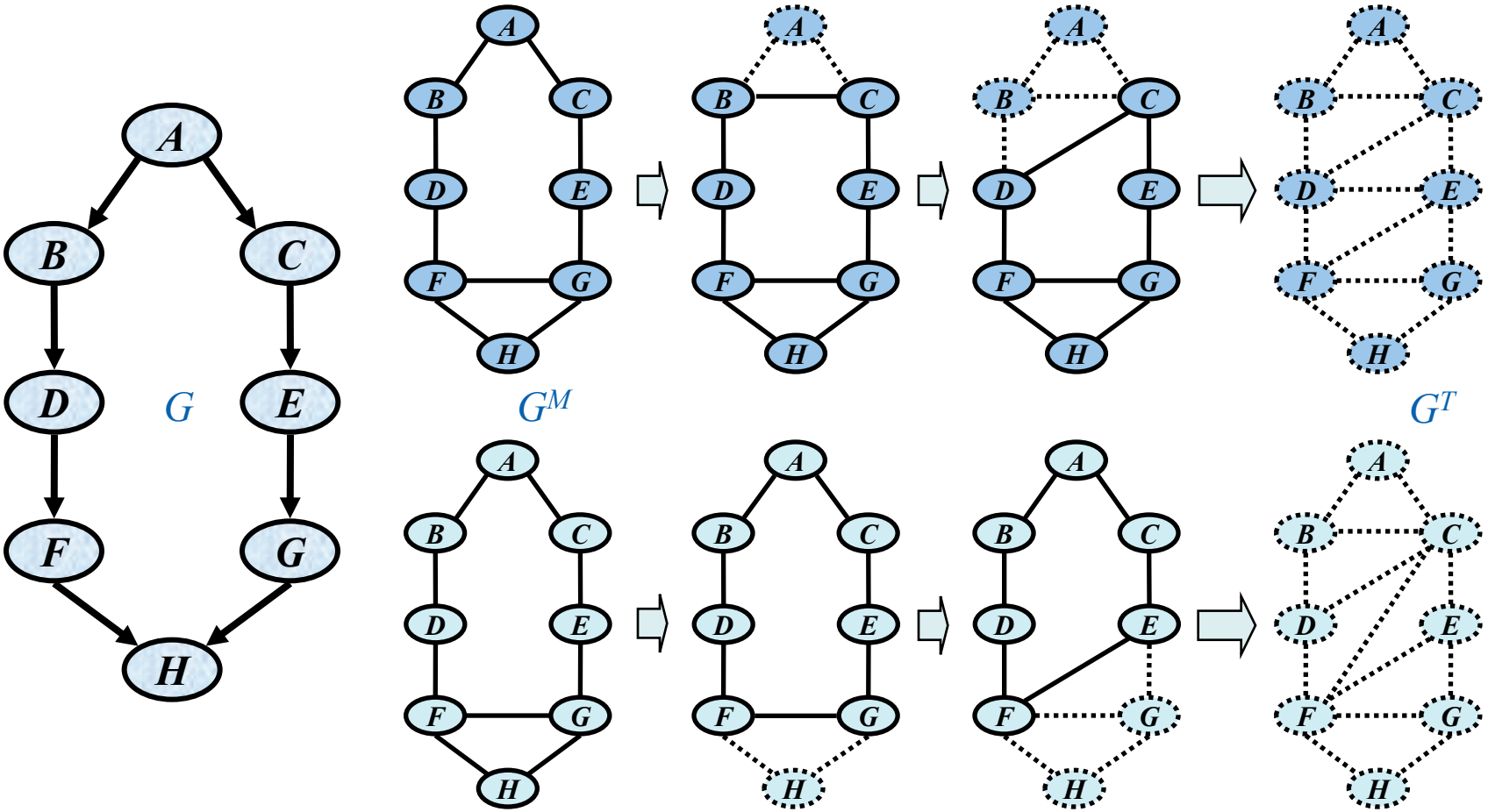
Finally:

- Junction tree algorithms are also useful for other purposes!
- There's so much more to BNs…!

Bonus slides

Triangulation

Each elimination ordering triangulates the graph, not necessarily in the same way:



Triangulation with Min-Fill

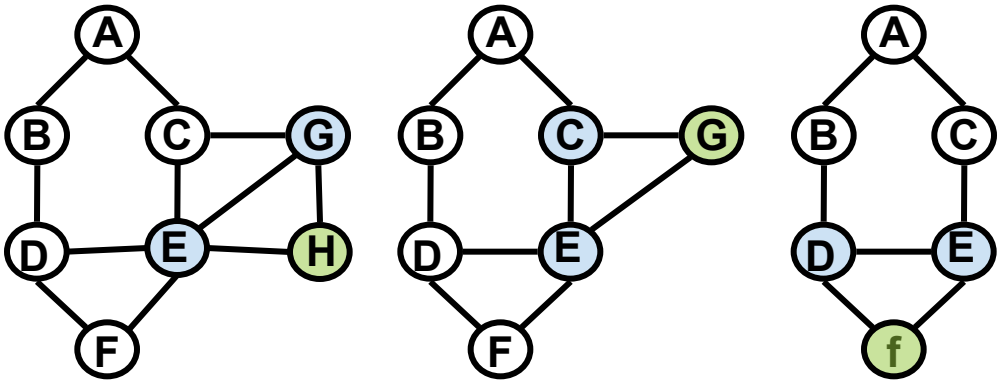
Intuitively, triangulations with as few fill-ins as possible are preferred

- Leaves us with small cliques (small potentials)

A common heuristic ('Min-fill'):

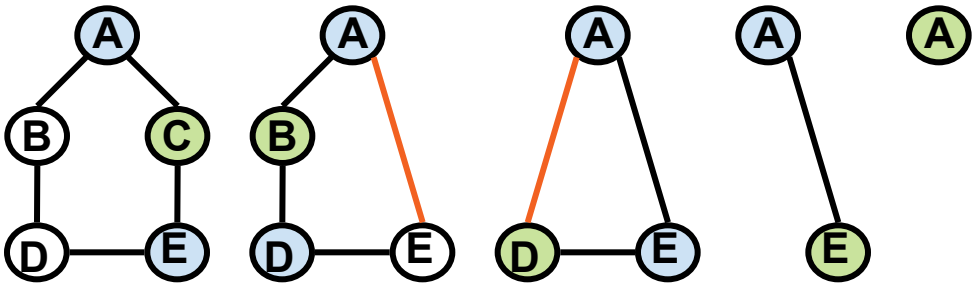
- Repeat until no nodes remain:
  - Find the node whose elimination would require the least number of fill-ins (may be zero).
  - Eliminate that node, and note the need for a fill-in edge between any two non-adjacent neighbors.
- Add the fill-in edges to the original graph.

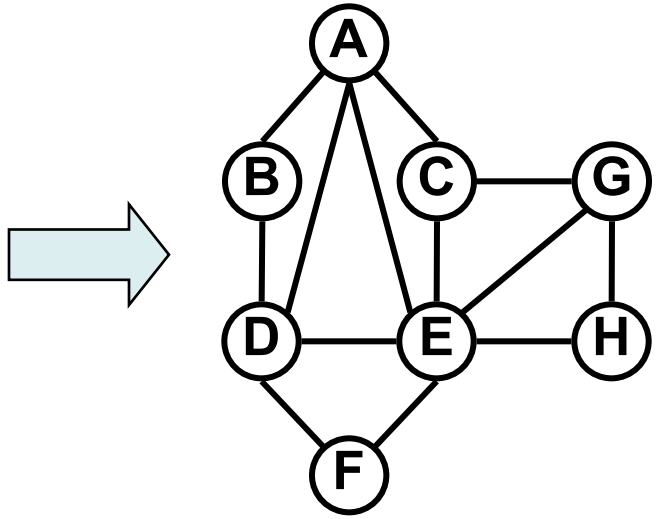
Triangulation example



G^M

Eliminate the vertex that requires least number of edges to be added.





G^T

	vertex removed	induced clique	added edges
1	H	EGH	-
2	G	CEG	-
3	F	DEF	-
4	C	ACE	A--E

	vertex removed	induced clique	added edges
5	B	ABD	A--D
6	D	ADE	-
7	E	AE	-
8	A	A	-

A few useful theorems

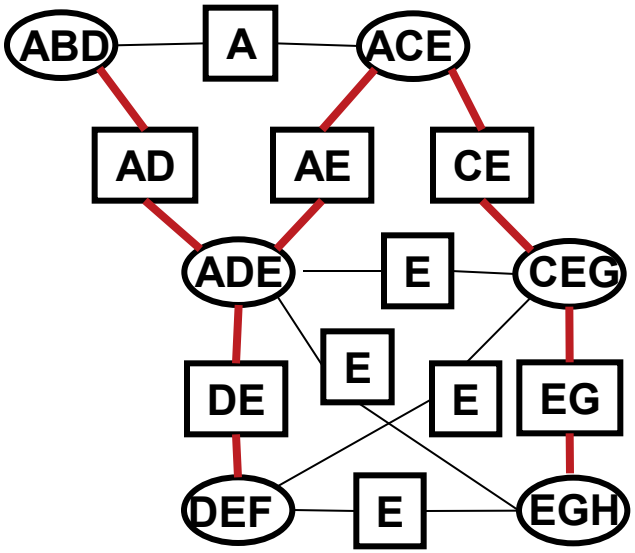
- An undirected graph is triangulated
  if and only if
  its junction graph has a junction tree

- A sub-tree of the junction graph of a triangulated graph is a junction tree
  if and only if
  it is a spanning of maximal weight (MST).

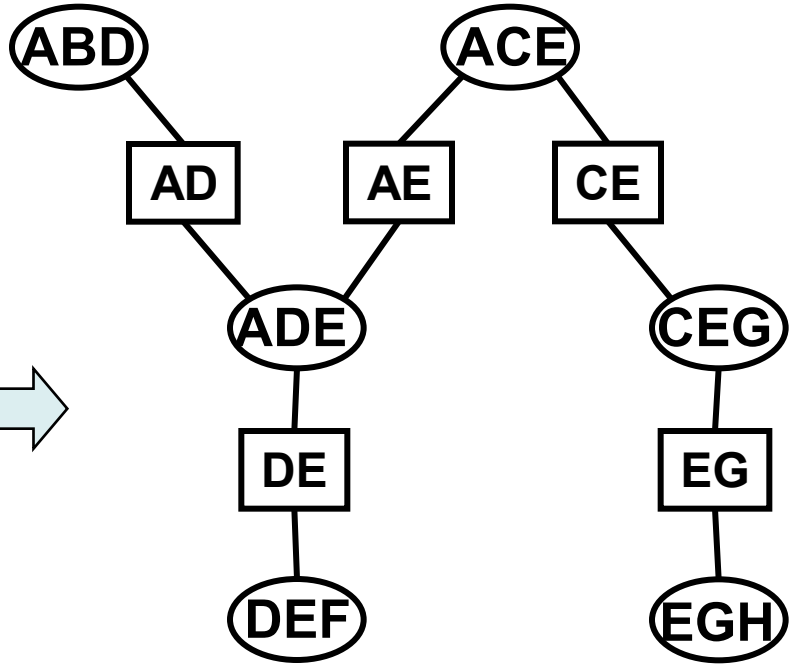
Finding a Minimal Spanning Tree

Kruskal's algorithm: choose successively a link of maximal weight unless it creates a cycle.

Junction graph G^J (incomplete)

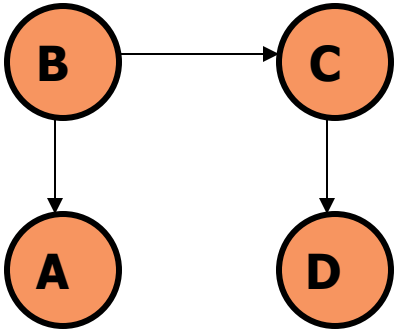


Junction tree G^{JT}

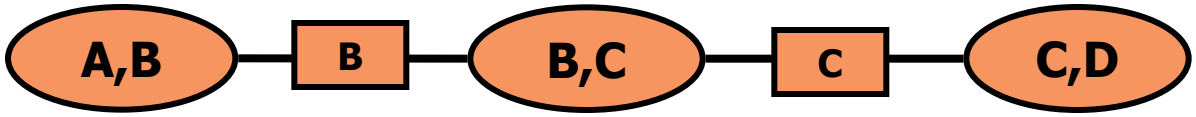


Small propagation example

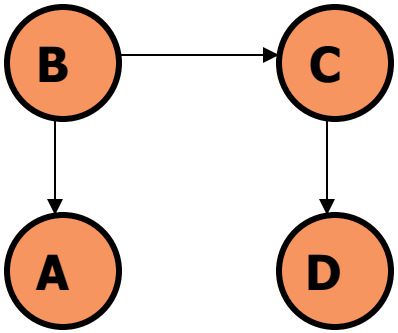
Example BN:

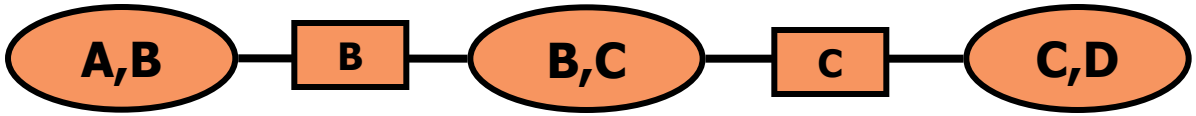


Phase 1: create a Junction Tree:



Small propagation example





Phase 2-step 1: initialization

Variable	Associated Cluster	Clique Potentials
A	A,B	$\phi_{A,B} = P(B)P(A \mid B)$
B	A,B	$\phi_{A,B} = P(B)P(A \mid B)$
C	B,C	$\phi_{B,C} = P(C \mid B)$
D	C,D	$\phi_{C,D} = P(D \mid C)$

Small propagation example

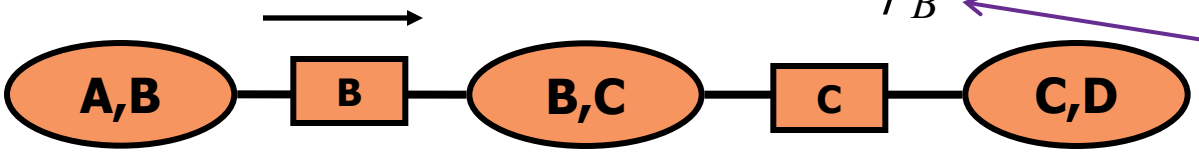
Phase 2: Collect evidence

- Choose arbitrary clique, e.g. {B,C}, where all potential functions will be collected.
- Recursively call neighbouring cliques for messages:
- 1. Call {A,B}:
  - 1. Projection onto separator B:

$$\phi_B = \sum_A \phi_{A,B} = \sum_A P(B)P(A \mid B) = P(B)$$

  - 2. Absorption into {B,C}:

$$\phi_{B,C} \leftarrow \phi_{B,C} \frac{\phi_B}{\phi_B^{old}} = P(C \mid B)P(B) = P(B,C)$$



No old value in first pass (1)

Small propagation example

Phase 2: Collect evidence (cntd)

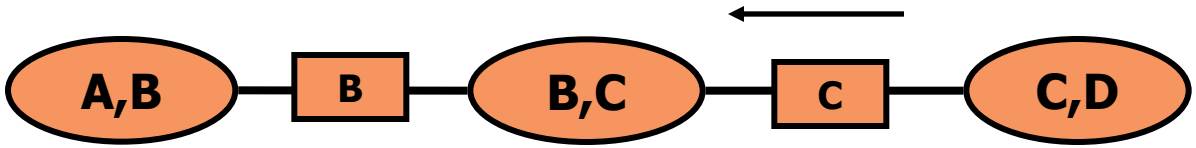
- 2. Call {C,D}:
  - 1. Projection onto separator C:

$$\phi_C = \sum_D \phi_{C,D} = \sum_D P(D \mid C) = 1$$

  - 2. Absorption into {B,C}:

$$\phi_{B,C} \leftarrow \phi_{B,C} \frac{\phi_C}{\phi_C^{old}} = P(B,C)$$

Result from absorption in first call



Small propagation example

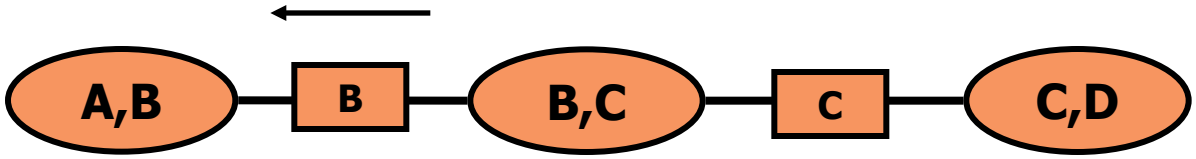
Phase 2: Distribute evidence

- Pass messages recursively to neighboring nodes
- Pass message from {B,C} to {A,B}:
  - 1. Projection onto separator B:

$$\phi_B = \sum_C \phi_{B,C} = \sum_C P(B,C) = P(B)$$

  - 2. Absorption into {A,B}:

$$\phi_{A,B} \leftarrow \phi_{A,B} \frac{\phi_B}{\phi_B^{old}} = P(A,B) \frac{P(B)}{P(B)}$$



From phase 1

Small propagation example

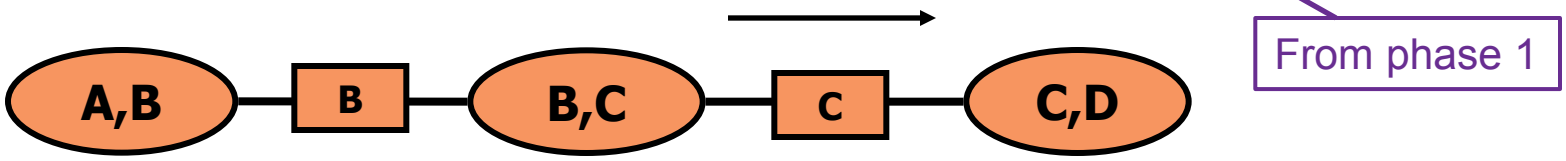
Phase 2: Distribute evidence (cntd)

- Pass message from {B,C} to {C,D}:
  - 1. Projection onto separator C:

$$\phi_C = \sum_B \phi_{B,C} = \sum_B P(B,C) = P(C)$$

  - 2. Absorption into {C,D}:

$$\phi_{C,D} \leftarrow \phi_{C,D} \frac{\phi_C}{\phi_C^{old}} = P(D \mid C) \frac{P(C)}{1} = P(C,D)$$



Now the junction tree is consistent and marginalisation in any clique is okay.