COMP9418: Advanced Topics in Statistical Machine Learning

Jointree Algorithm

Instructor: Gustavo Batista

University of New South Wales

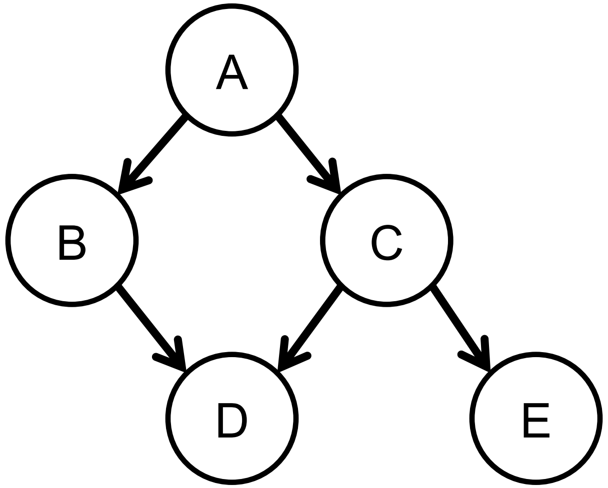
Introduction

- In this lecture, we will study a variation of VE known as *jointree algorithm*
  - Also known as *clique-tree* and *tree-clustering* algorithm
  - Jointree can be understood in terms of factor elimination
  - It improves VE complexity by answering multiple queries
  - It forms the basis of a class of approximate algorithms we will discuss later in this course

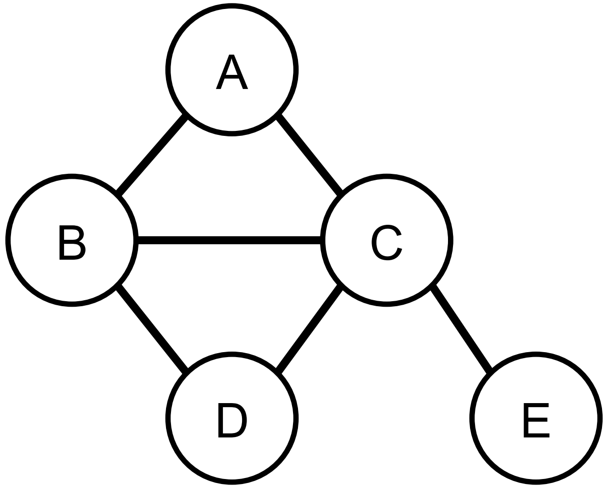
- We will start describing the idea of inference by factor elimination
  - In the sequence, we formalise the jointree algorithm using these ideas

Introduction

- Given a network we want to compute posterior marginals for each of its *n* variables
  - VE can compute a single marginal in $O(n \exp(w))$, where w is the width of the elimination order
  - We can run VE n times, leading to a total complexity of $O(n^2 \exp(w))$
  - The n^2 term can be problematic even when the treewidth is small



- Jointree can avoid this complexity leading to a $O(n \exp(w))$ time and space complexity
  - Bayesian networks, it will compute the posterior marginals for all network families (a variable and its parents)
  - For Markov networks, it will provide posterior marginals for all cliques (clique-tree algorithm)



Factor Elimination

- We want to compute prior marginals over some variable Q
  - VE eliminates every other variable
  - Factor elimination will eliminate all factors except for one that contain Q

- The elimination of factor f_i from a set of factors $\boldsymbol{S}$ is a two–step process
  - Eliminate all variables $\boldsymbol{V}$ that appear *only* in factor f_i
  - Multiply the result $\sum_{\boldsymbol{V}} f_i$ by some other factor f_j in the set $\boldsymbol{S}$

Factor Elimination Algorithm: FE1

Input: Network N, a variable Q in the network

Output: prior marginal $P(Q)$

$\boldsymbol{S} \leftarrow$ factors of network N

$f_r \leftarrow$ a factor in $\boldsymbol{S}$ that contains variable Q

while S has more than one factor **do**

  remove a factor $f_i \neq f_r$ from set $\boldsymbol{S}$

  $\boldsymbol{V} \leftarrow$ variables that appear in factor f_i but not in $\boldsymbol{S}$

  $f_j \leftarrow f_j \sum_{\boldsymbol{V}} f_i$ for some factor $f_j \in \boldsymbol{S}$

return $\text{project}(f_r, Q)$

- This algorithm makes use of the factor operation $\text{project}(f, \boldsymbol{Q})$, which simply sums out all variables not in $\boldsymbol{Q}$:

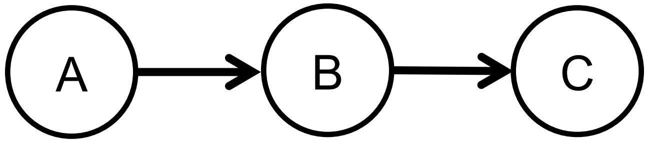
$$\text{project}(f, \boldsymbol{Q}) \stackrel{\text{def}}{=} \sum_{vars(f)-\boldsymbol{Q}} f$$

Factor Elimination: Correctness

- Factor elimination is simply a variation of VE
  - While VE eliminates one variable at a time, FE eliminates a set of variables $\boldsymbol{V}$ at once
  - As these variables appear only in f_i, we replace f_i by a new factor $\sum_{\boldsymbol{V}} f_i$
  - We take an extra step and multiply the new factor by other factor f_j
- At each iteration, the number of factors in $\boldsymbol{S}$ decreases by 1
  - After enough iterations, $\boldsymbol{S}$ will contain a single factor that contains Q
  - Eliminating all other variables but Q provides the answer to the query
- Two open choices
  - Which factor f_i to eliminate next
  - Which factor f_j to multiply by

Any set of choices will provide a valid answer
But some choices will be computationally better

Factor Elimination: Example



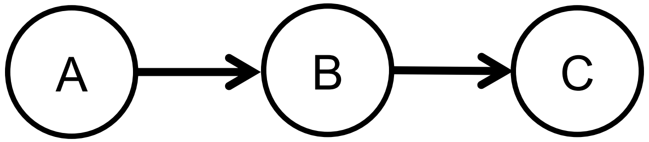
Suppose we want to answer $P(C)$ for this Bayesian network

A	$f_A = \Theta(A)$
a	.6
$\bar{a}$	.4

A	B	$f_B = \Theta_{B\|A}$
a	b	.9
a	$\bar{b}$	.1
$\bar{a}$	b	.2
$\bar{a}$	$\bar{b}$	.8

B	C	$f_C = \Theta_{C\|B}$
b	c	.3
b	$\bar{c}$	.7
$\bar{b}$	c	.5
$\bar{b}$	$\bar{c}$	.5

Factor Elimination: Example



We start eliminating f_A.
As A is in f_A and f_B, $\boldsymbol{V} = \emptyset$

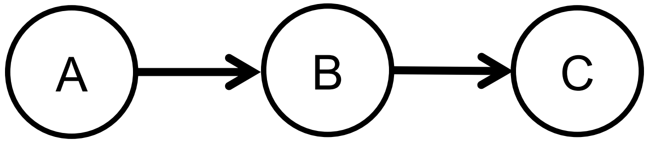
A	$f_A = \Theta(A)$
a	.6
$\bar{a}$	.4

$\times$

A	B	$f_B = \Theta_{B\|A}$
a	b	.9
a	$\bar{b}$	.1
$\bar{a}$	b	.2
$\bar{a}$	$\bar{b}$	.8

B	C	$f_C = \Theta_{C\|B}$
b	c	.3
b	$\bar{c}$	.7
$\bar{b}$	c	.5
$\bar{b}$	$\bar{c}$	.5

Factor Elimination: Example



We start eliminating f_A.
As A is in f_A and f_B, $\boldsymbol{V} = \emptyset$

A	$f_A = \Theta(A)$
a	.6
$\bar{a}$	.4

$\times$

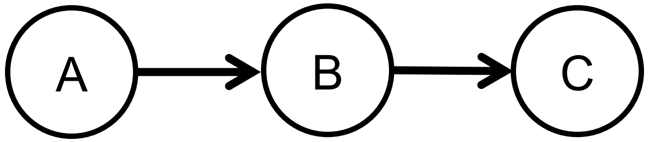
A	B	$f_B = \Theta_{B\|A}$
a	b	.9
a	$\bar{b}$	.1
$\bar{a}$	b	.2
$\bar{a}$	$\bar{b}$	.8

B	C	$f_C = \Theta_{C\|B}$
b	c	.3
b	$\bar{c}$	.7
$\bar{b}$	c	.5
$\bar{b}$	$\bar{c}$	.5

A	B	$f_A f_B$
a	b	.54
a	$\bar{b}$	.06
$\bar{a}$	b	.08
$\bar{a}$	$\bar{b}$	.32

B	C	f_C
b	c	.3
b	$\bar{c}$	.7
$\bar{b}$	c	.5
$\bar{b}$	$\bar{c}$	.5

Factor Elimination: Example



Now, we eliminate $f_A f_B$.
$\boldsymbol{V} = \{A\}$

A	B	$f_A f_B$
a	b	.54
a	$\bar{b}$	.06
$\bar{a}$	b	.08
$\bar{a}$	$\bar{b}$	.32

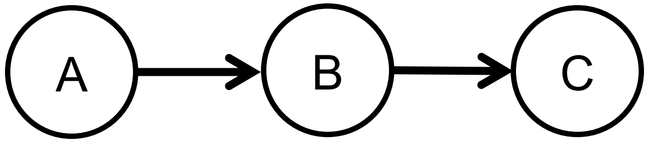
B	C	f_C
b	c	.3
b	$\bar{c}$	.7
$\bar{b}$	c	.5
$\bar{b}$	$\bar{c}$	.5

B	$\sum_A f_A f_B$
b	.62
$\bar{b}$	.38

X

B	C	f_C
b	c	.3
b	$\bar{c}$	.7
$\bar{b}$	c	.5
$\bar{b}$	$\bar{c}$	.5

Factor Elimination: Example



We multiply $\sum_A f_A f_B$ into f_C

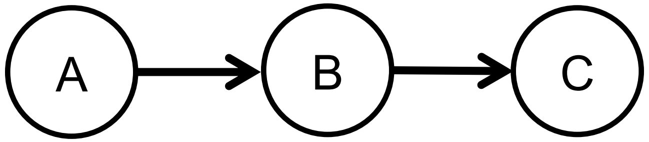
B	$\sum_A f_A f_B$
b	.62
$\bar{b}$	.38

×

B	C	f_C
b	c	.3
b	$\bar{c}$	.7
$\bar{b}$	c	.5
$\bar{b}$	$\bar{c}$	.5

B	C	$f_C \sum_A f_A f_B$
b	c	.186
b	$\bar{c}$	.434
$\bar{b}$	c	.190
$\bar{b}$	$\bar{c}$	.190

Factor Elimination: Example



Finally, we eliminate all other variables to obtain the answer

B	$\sum_A f_A f_B$
b	.62
$\bar{b}$	.38

×

B	C	f_C
b	c	.3
b	$\bar{c}$	.7
$\bar{b}$	c	.5
$\bar{b}$	$\bar{c}$	.5

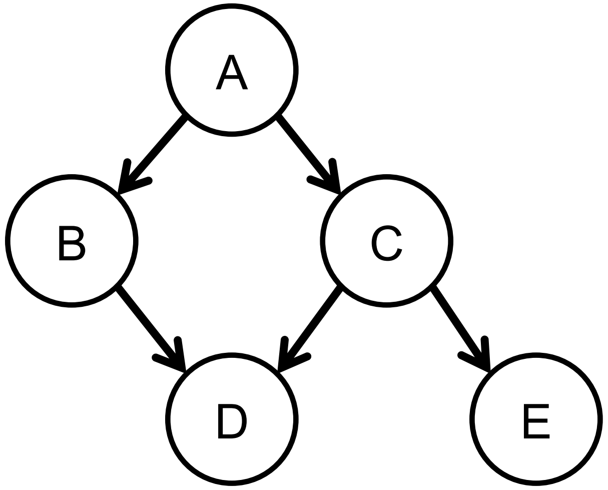
B	C	$f_C \sum_A f_A f_B$
b	c	.186
b	$\bar{c}$	.434
$\bar{b}$	c	.190
$\bar{b}$	$\bar{c}$	.190

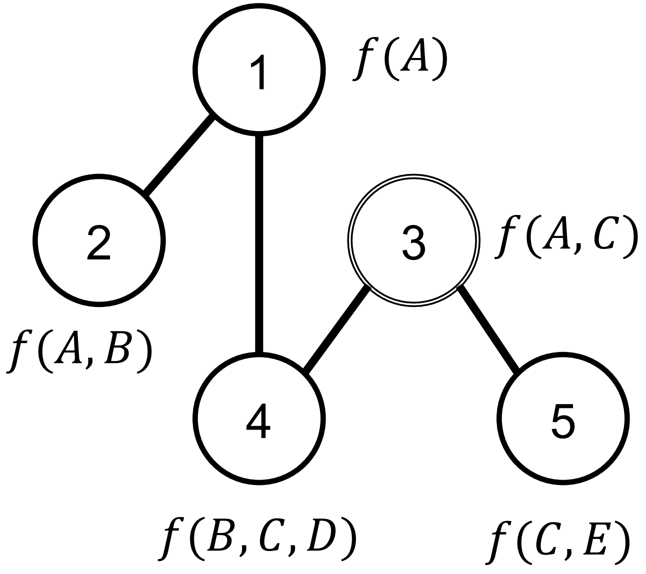
$\longrightarrow$

C	$\sum_B f_C \sum_A f_A f_B$
c	.376
$\bar{c}$	.624

Elimination Trees

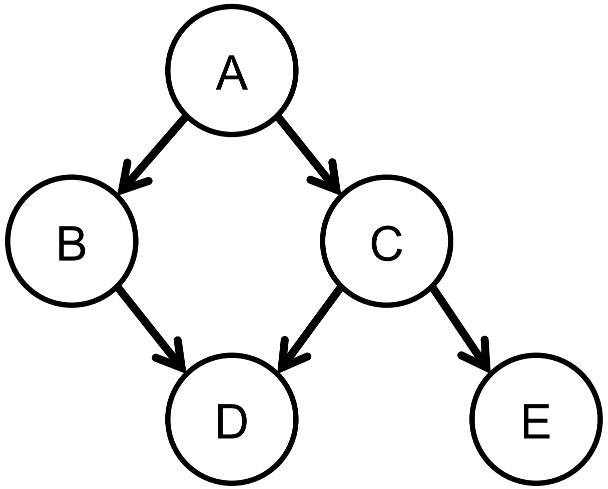
- In variable elimination, the elimination order was used to specify an elimination strategy
  - The amount of work performed by VE was determined by the quality (width) of the order
- In factor elimination, we use trees to specify an elimination strategy
  - Each organization of factors into a tree structure represents a particular strategy
  - The quality of such trees (also called *width*) can be used to quantify the amount of work performed
- The figure shows one such tree
  - We call it *elimination tree*

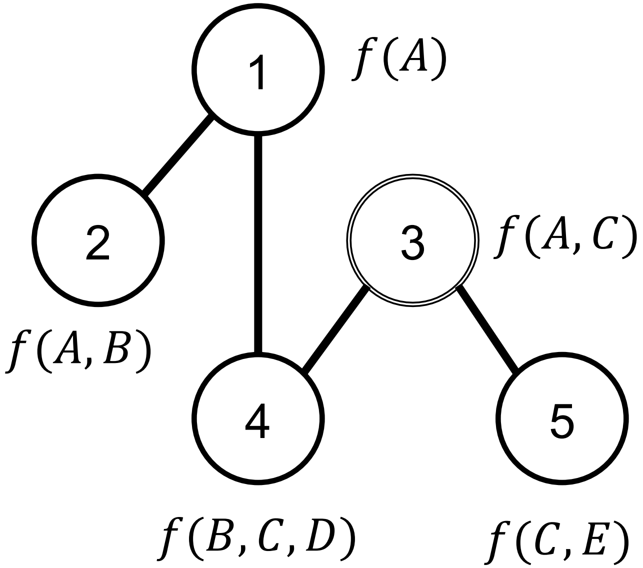




Elimination Trees

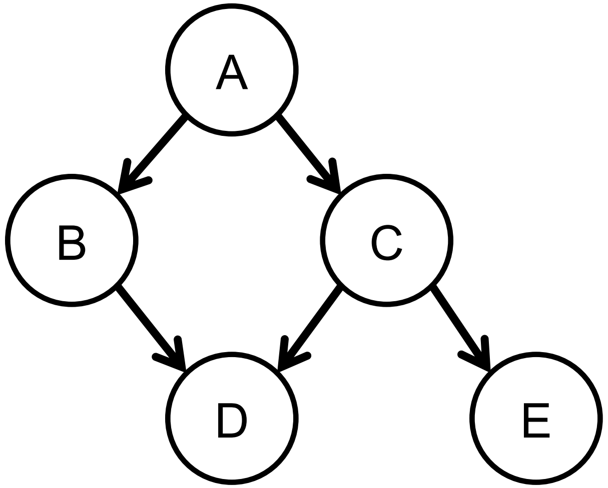
- An elimination tree for a set of factors $\boldsymbol{S}$ is a pair (T, ϕ) where T is a tree
  - Each factor in $\boldsymbol{S}$ is assigned to exactly one node in T
  - We use ϕ_i to denote the product of factors assigned to node i in T
  - We also use $\text{vars}(i)$ to denote the variables in factor ϕ_i
- In this figure, the elimination tree (T, ϕ) has five nodes which are in one-to-one correspondence with the given factors
  - A node in an elimination tree may have multiple factors assigned to it or no factors at all
  - For many examples, there will be a one-to-one correspondence between factors and nodes in an elimination tree

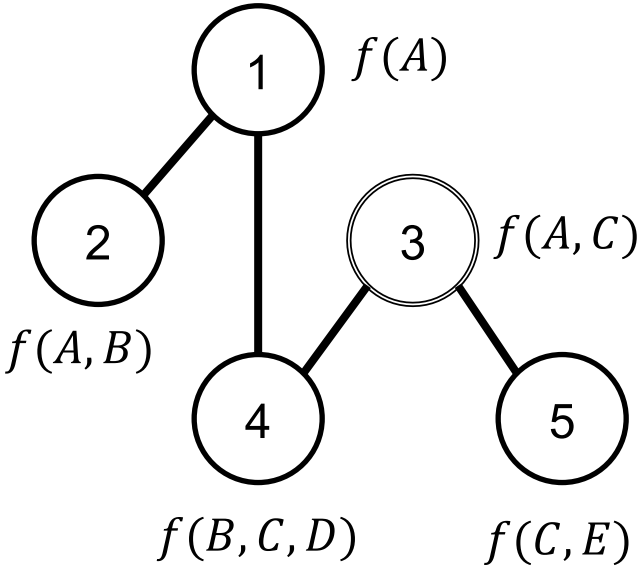




Elimination Trees

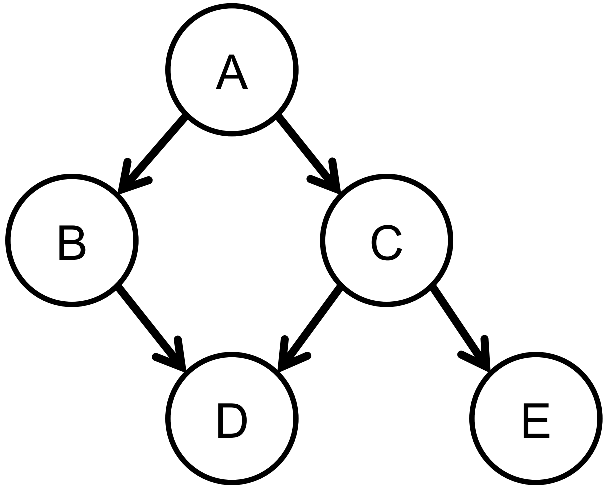
- When computing the marginal over variables $\boldsymbol{Q}$, we need to choose a special node r
  - This node r, called a *root*, must be chosen such that $\boldsymbol{Q} \subseteq vars(r)$
  - For example, if $\boldsymbol{Q} = \{C\}$, we can choose nodes 3, 4 and 5 can act as roots
  - A root node is not strictly needed but will simplify the discussion

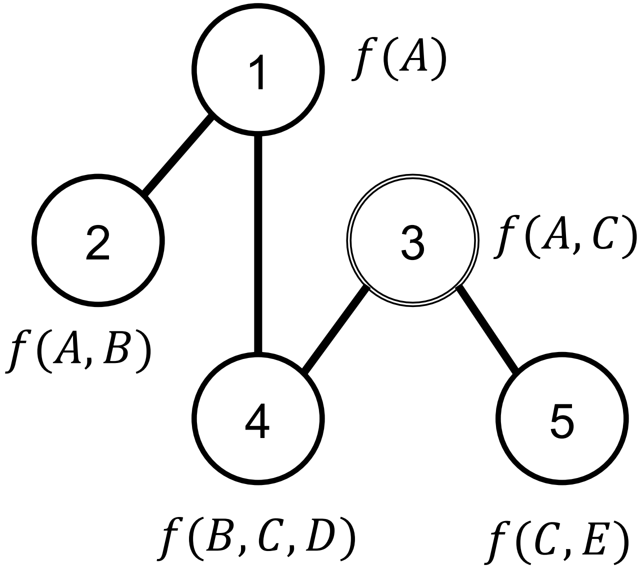




Elimination Trees

- Given an elimination tree and a corresponding root r, our elimination strategy is
  - Eliminate factor ϕ_i only if it has a single neighbor j and $i \neq r$
  - Sum out variables $\boldsymbol{V}$ that appear in ϕ_i but not in the rest of the tree
  - Multiply the result $\sum_{\boldsymbol{V}} \phi_i$ into the factor ϕ_j associated with its single neighbor j





Factor Elimination Algorithm: FE2

Input: Network N, a set of variables $\boldsymbol{Q}$ in the network, an elimination tree (T, ϕ), a root node r

Output: prior marginal $P(\boldsymbol{Q})$

while tree T has more than one node **do**

  remove a node $i \neq r$ having a single neighbour j from T

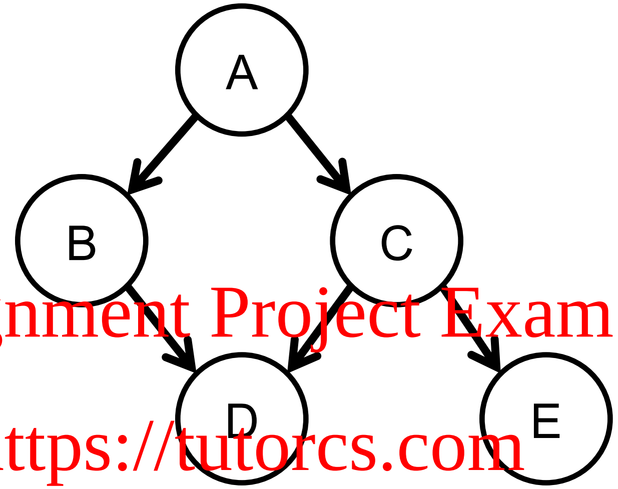
  $\boldsymbol{V} \leftarrow$ variables appearing in ϕ_i but not in remaining tree T

  $\phi_j \leftarrow \phi_j \sum_{\boldsymbol{V}} \phi_i$

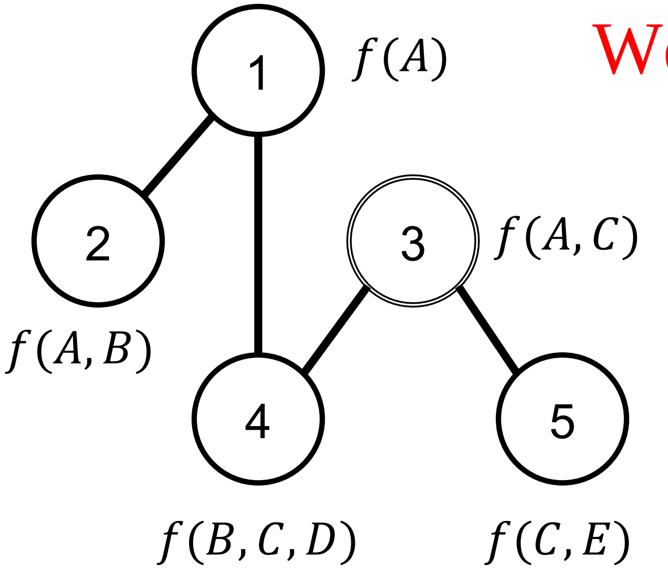
return project$(\phi_r, \boldsymbol{Q})$

- We still need to make a choice of which node to remove since we may have more than one node *i* in the tree that satisfies the stated properties
- However, the choice made at this step does not affect the amount of work done by the algorithm

Elimination Trees: Example

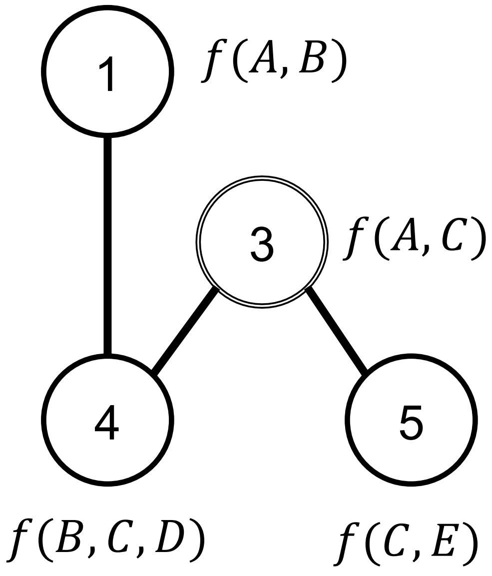


Let us suppose we want to compute the marginal over variable C. We set $r = 3$

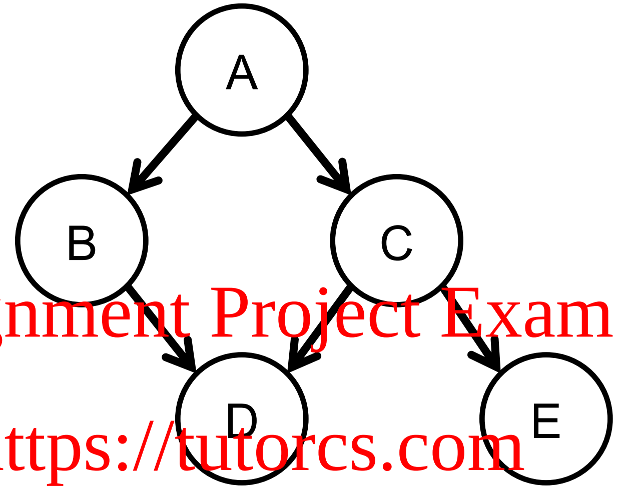


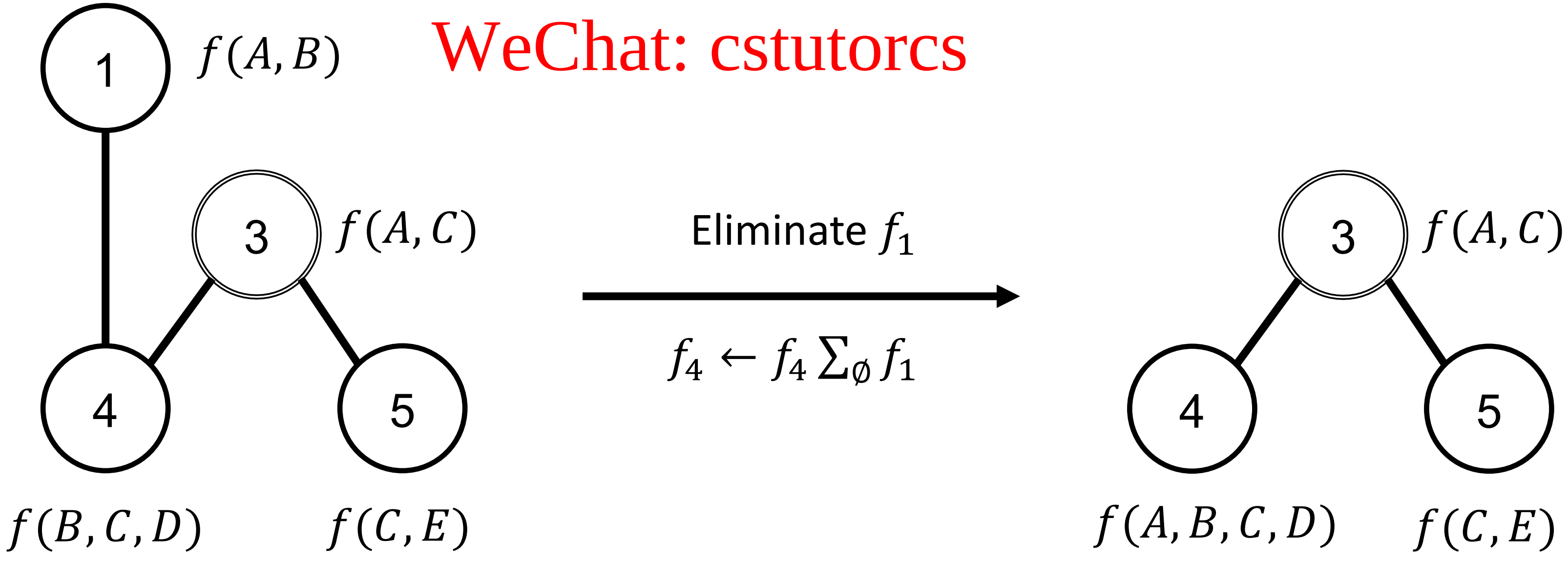
Eliminate f_2

$f_1 \leftarrow f_1 \sum_{\emptyset} f_2$

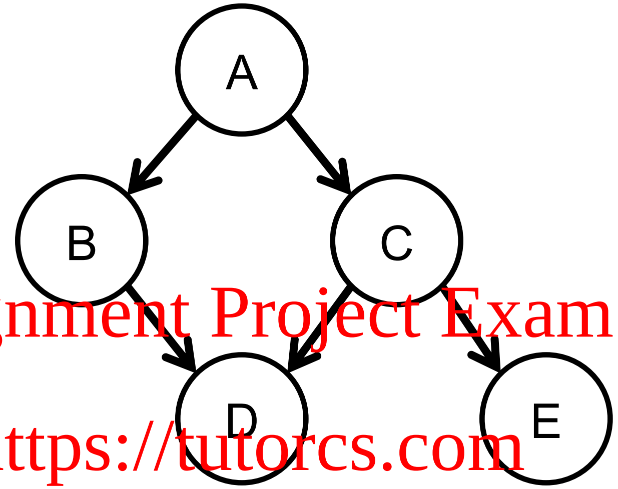


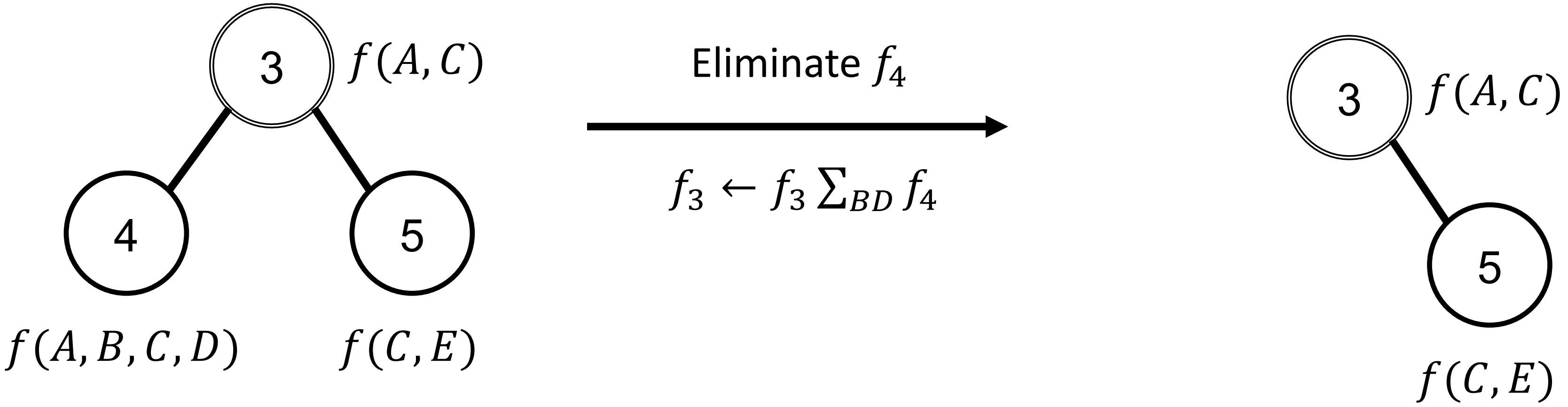
Elimination Trees: Example



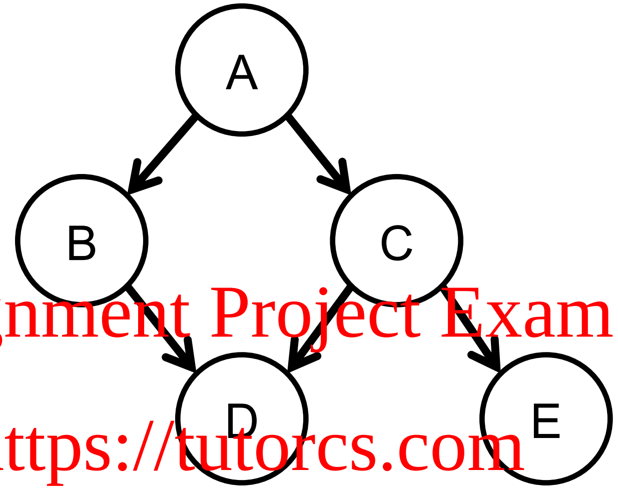


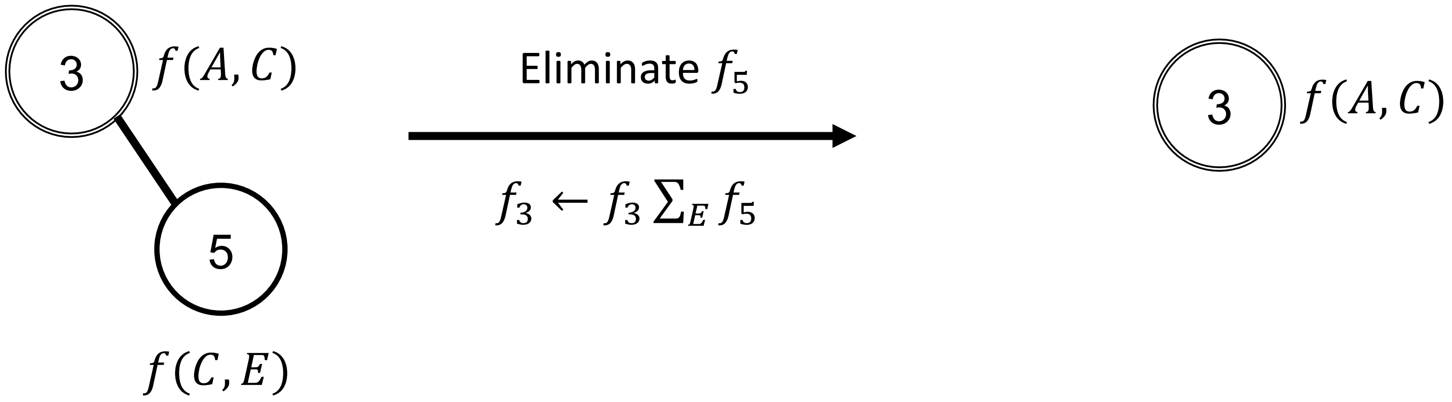
Elimination Trees: Example



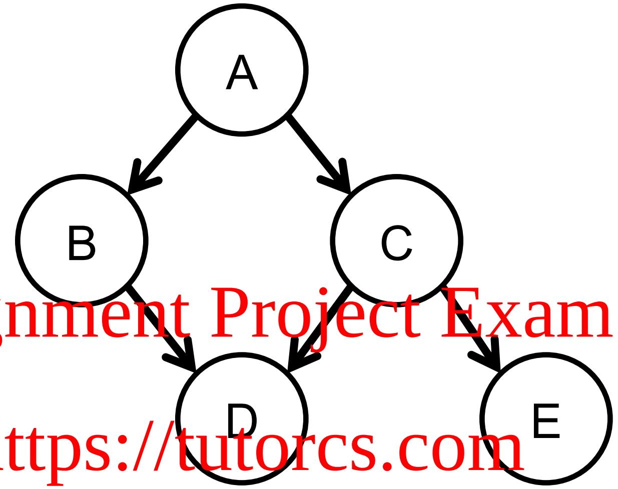


Elimination Trees: Example

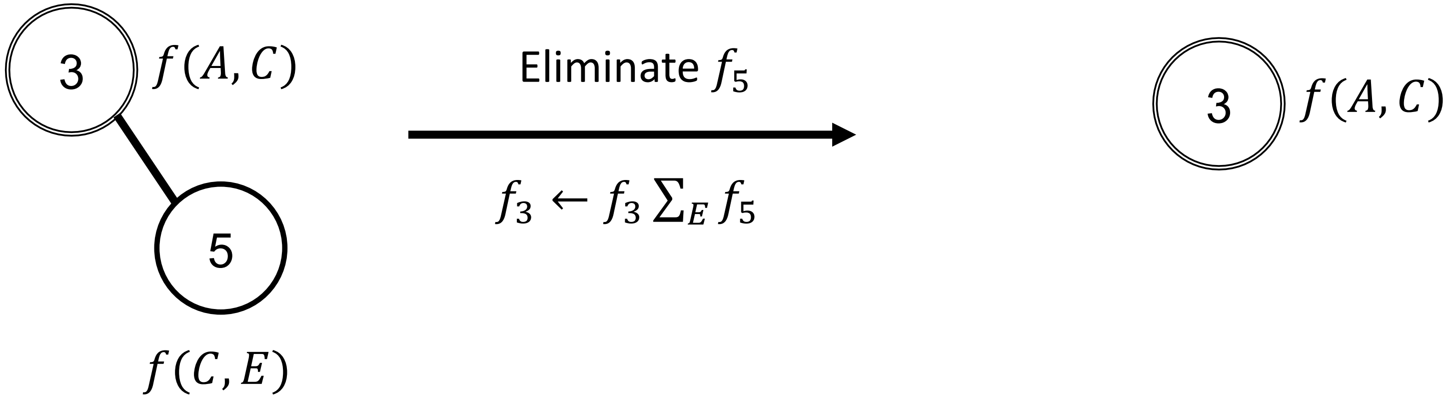




Elimination Trees: Example



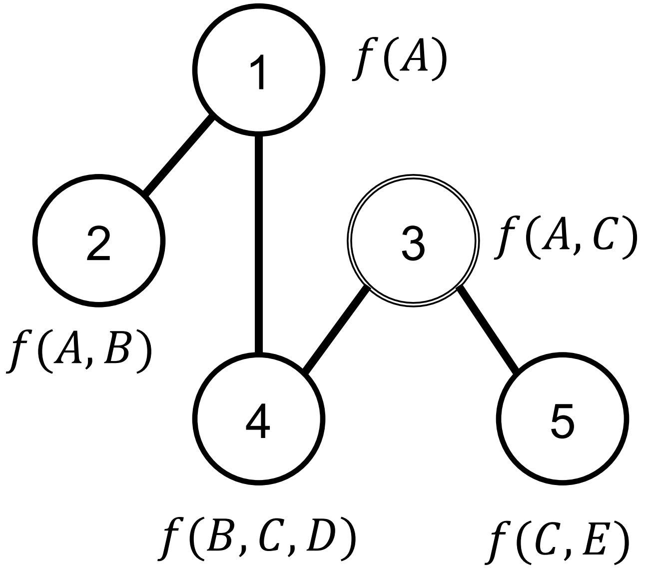
The final factor is over variables A, C.
Eliminating A gives the desired result.

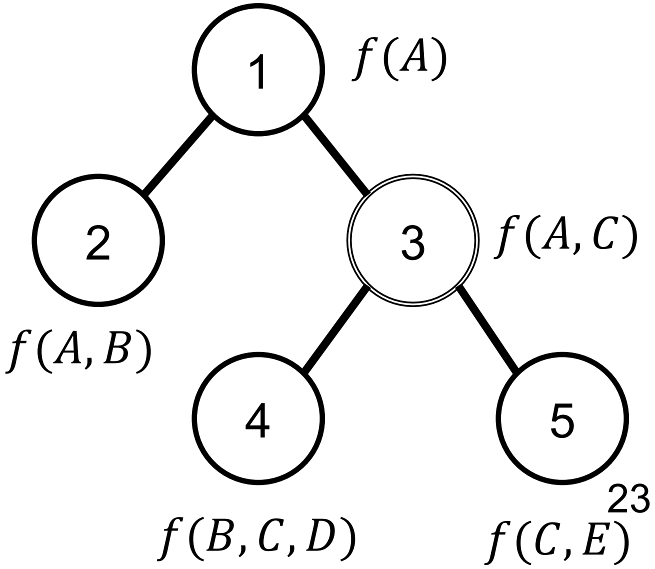


Elimination Trees and Runtime

- With VE, any elimination order will lead to correct results
  - Yet a specific order may have a considerable effect on the amount of work

- FE is similar
  - Any elimination tree will lead to correct results
  - Yet some trees will lead to less work
  - We will return to this topic later



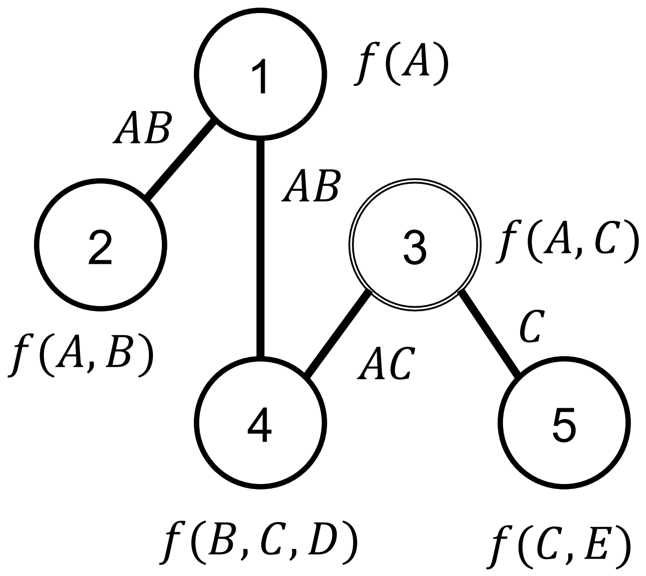


Separator

- The separator of edge $i - j$ in an elimination tree is a set of variables defined as follows

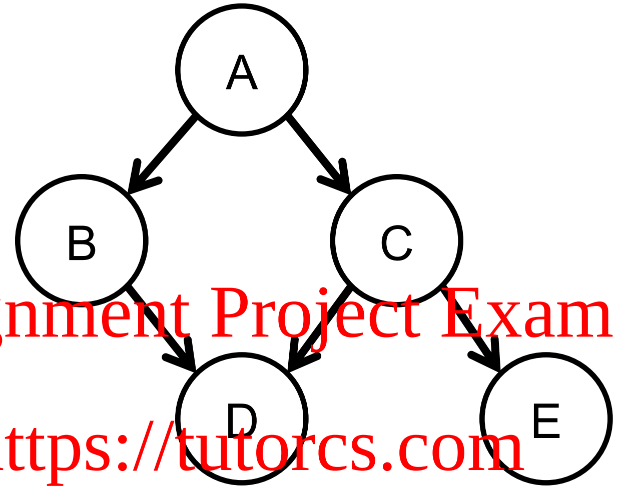
$$\boldsymbol{S}_{ij} = vars(i,j) \cap vars(j,i)$$

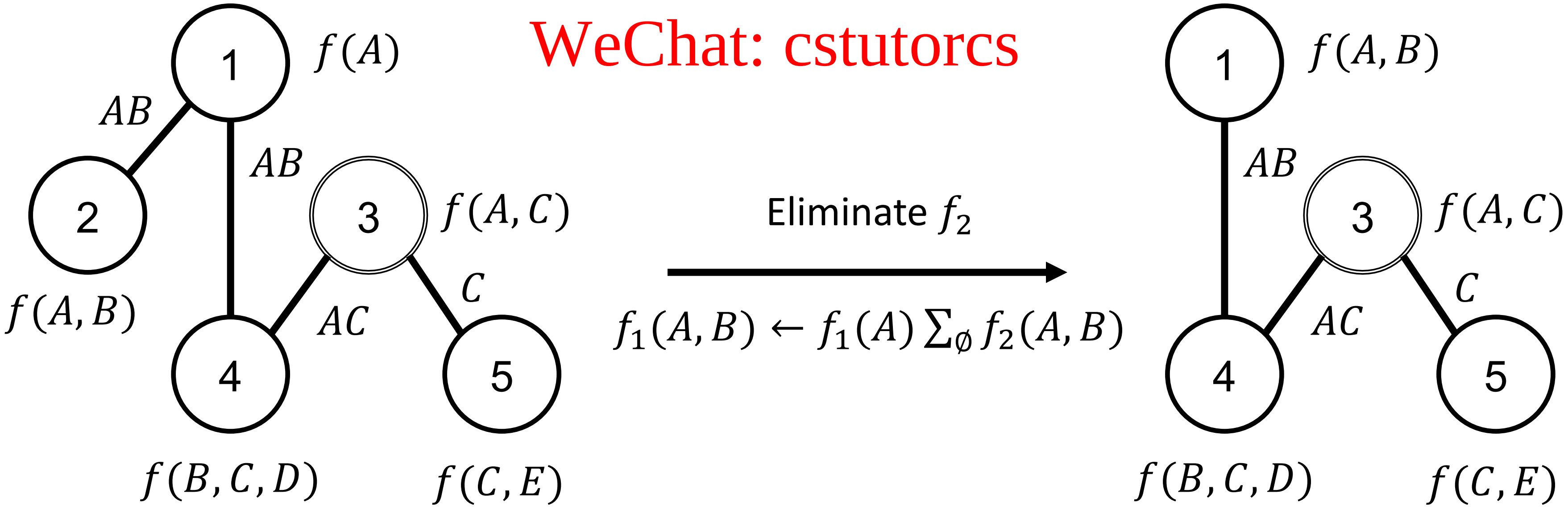
  - $vars(i,j)$ are variables that appear on the i–side of edge $i - j$
  - $vars(j,i)$ are variables that appear on the j–side of edge $i - j$



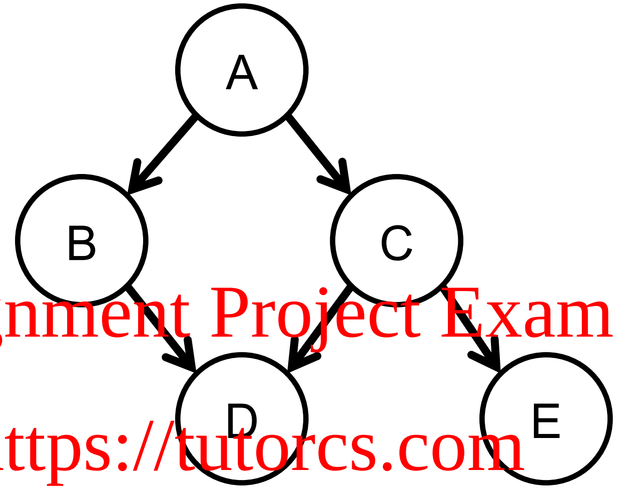
- When variables $\boldsymbol{V}$ are summed out of factor f_i before it is eliminated, the resulting factor is guaranteed to be over separator $\boldsymbol{S}_{ij}$

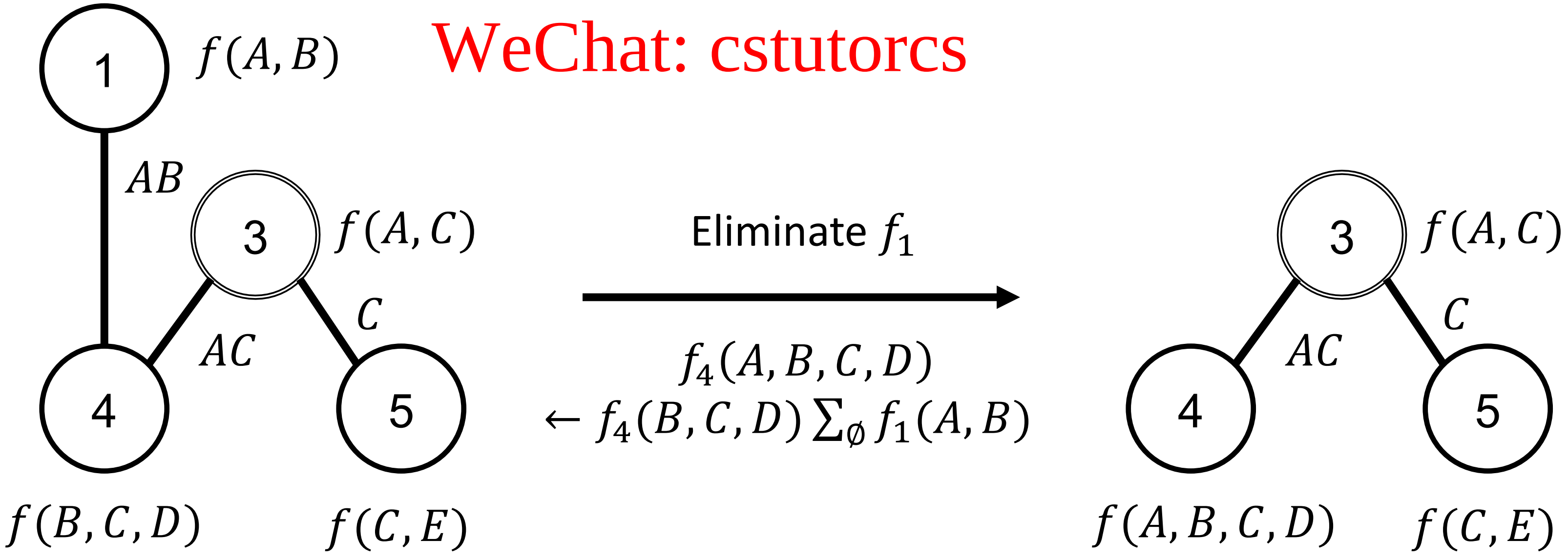
Separator: Example



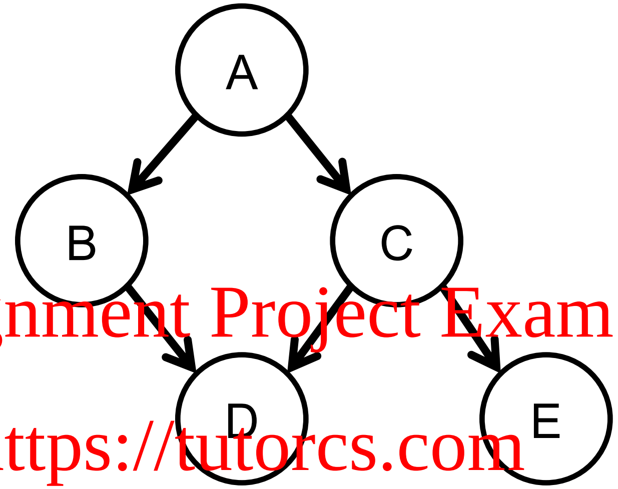


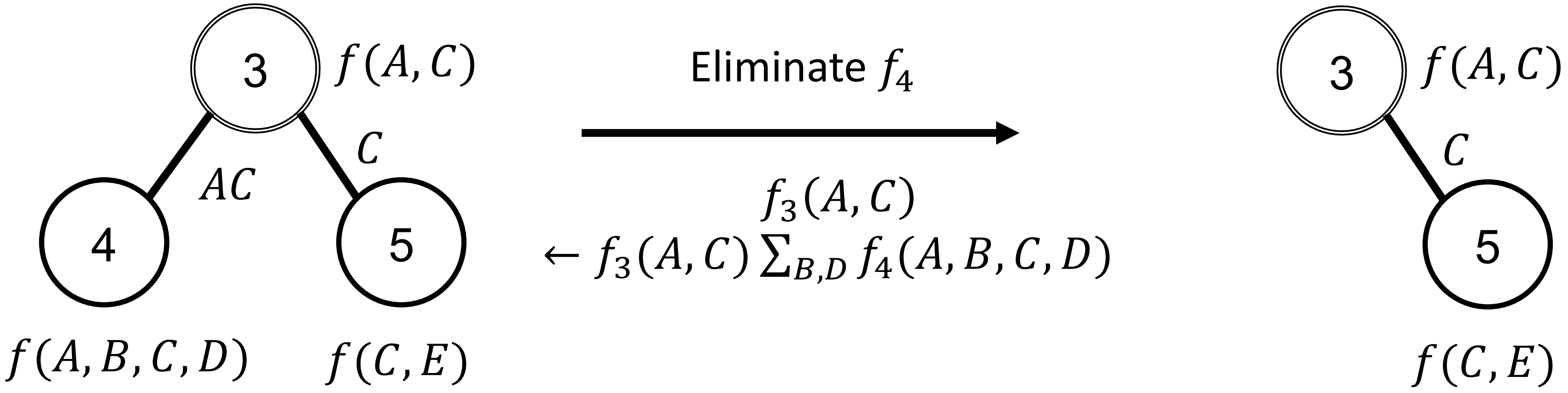
Separator: Example



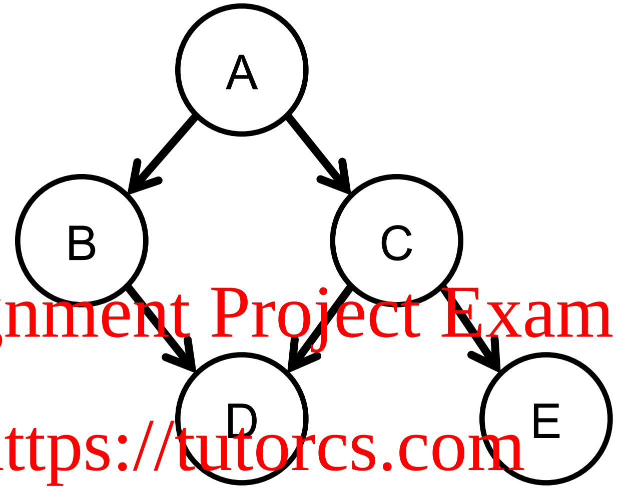


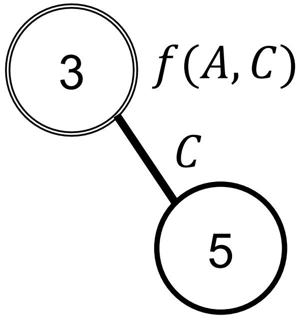
Separator: Example





Separator: Example





$f(A,C)$

$f(C,E)$

Eliminate f_5

$f_3(A,C) \leftarrow f_3(A,C) \sum_E f_5(C,E)$

3 $f(A,C)$

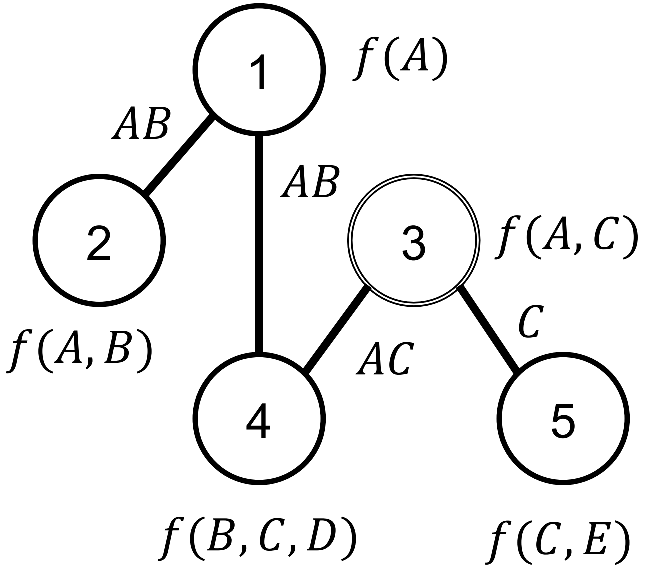
Cluster

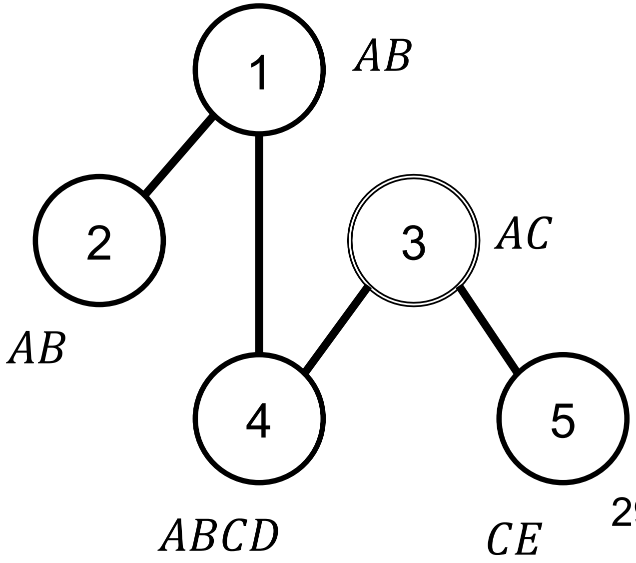
- The cluster of a node i in an elimination tree is a set of variables defined as follows:

$$\boldsymbol{C}_i = vars(i) \cup \bigcup_j \boldsymbol{S}_{ij}$$

- The width of an elimination tree is the size of its largest cluster $-$ 1

- This elimination tree has width = 3





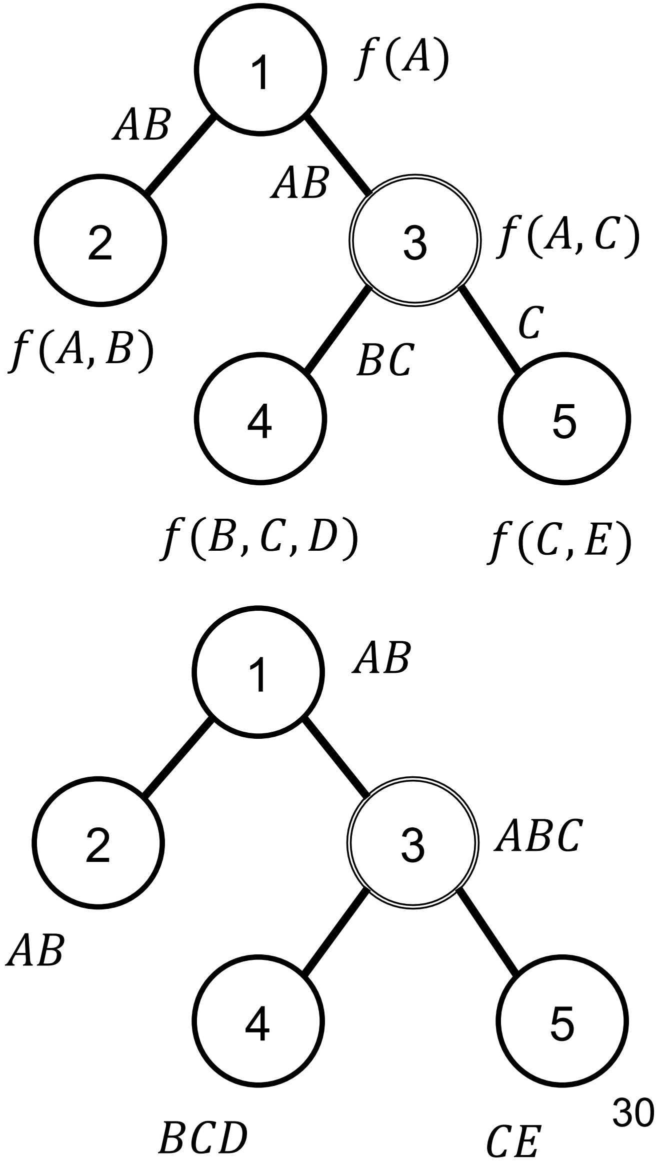
Cluster

- The cluster of a node i in an elimination tree is a set of variables defined as follows:

$$\boldsymbol{C}_i = vars(i) \cup \bigcup_j \boldsymbol{S}_{ij}$$

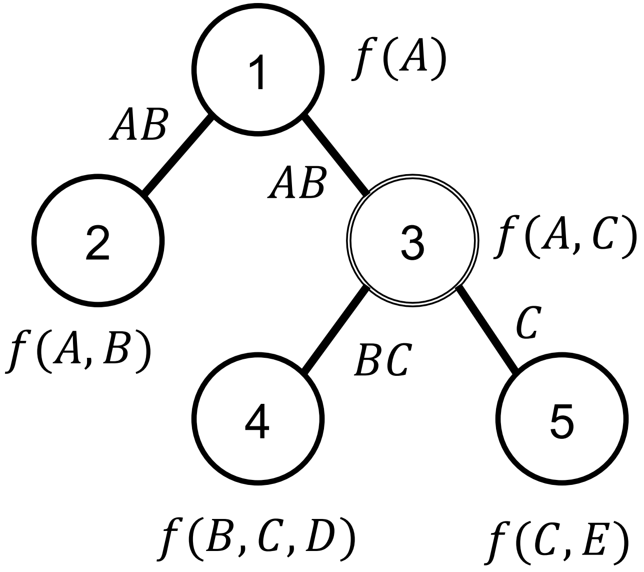
- The width of an elimination tree is the size of its largest cluster – 1

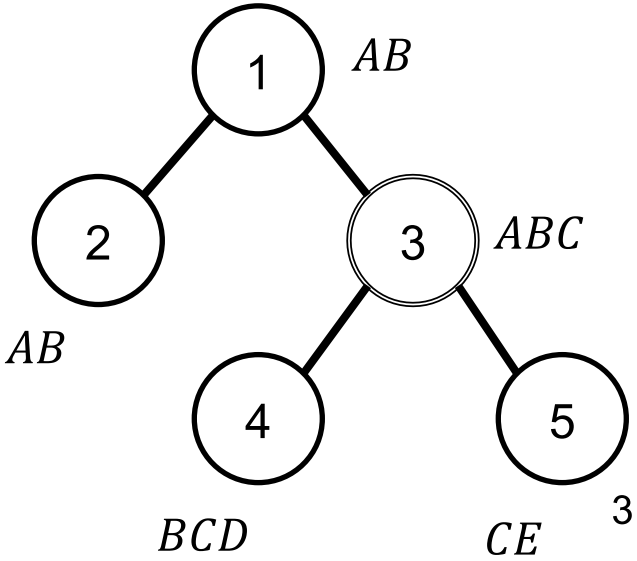
- This elimination tree has width = 3
- And this one has width = 2



Cluster

- Two key observations about clusters
  - When we are about to eliminate node i, the variables of factor ϕ_i are exactly the cluster of node i, $\boldsymbol{C}_i$
  - The factor ϕ_r must be over the cluster of root r, $\boldsymbol{C}_r$
- Hence, FE2 can be used to compute the marginal over any subset of cluster $\boldsymbol{C}_r$
  - These observations allow us to rephrase FE2
  - The new formulation takes advantages of both separators and clusters





Factor Elimination Algorithm: FE3

Input: Network N, a set of variables $\boldsymbol{Q}$ in the network, an elimination tree (T, ϕ), a root node r in T where $\boldsymbol{Q} \subseteq \boldsymbol{C}_r$

Output: prior marginal $P(\boldsymbol{Q})$

$\boldsymbol{C}_i$ is the cluster of node i in tree T

$\boldsymbol{S}_{ij}$ is the separator of edge $i - j$ in tree T

while tree T has more than one node **do**

remove a node $i \neq r$ having a single neighbour j from T

$\phi_j \leftarrow \phi_j \text{project}(\phi_i, \boldsymbol{S}_{ij})$

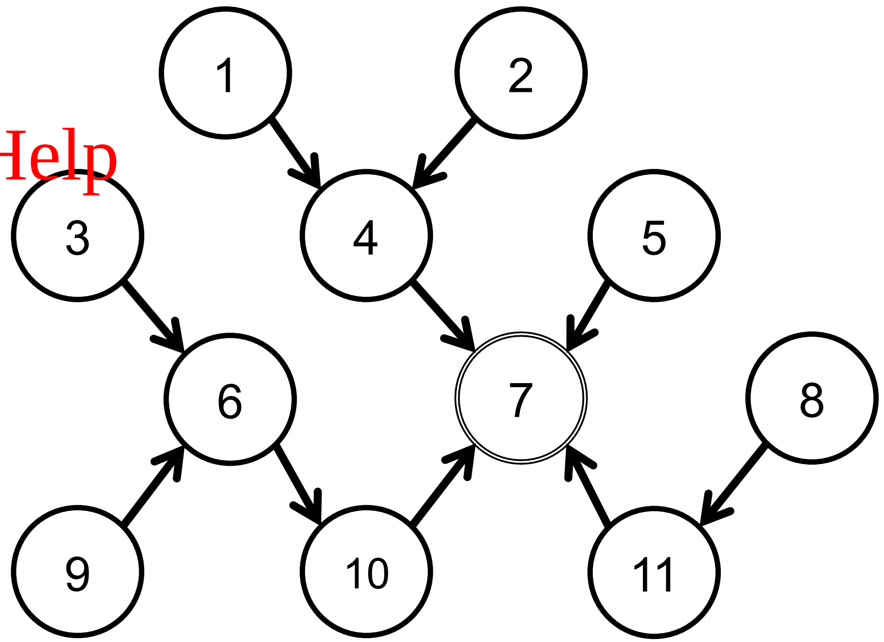
return $\text{project}(\phi_r, \boldsymbol{Q})$

Message-passing Formulation

- We will now rephrase our algorithm using a message passing paradigm
  - It will allow us to execute the algorithm without destroying the elimination tree in the process
  - This is important when computing multiple marginals
  - We can save intermediate computations and reuse them across different queries
- This reuse will be the key to achieving the complexity
  - Given an elimination tree of width w we will be able to compute the marginal over every cluster in $O(m \exp(w))$ time and space, where m is the number of nodes in the elimination tree

Message-passing Formulation

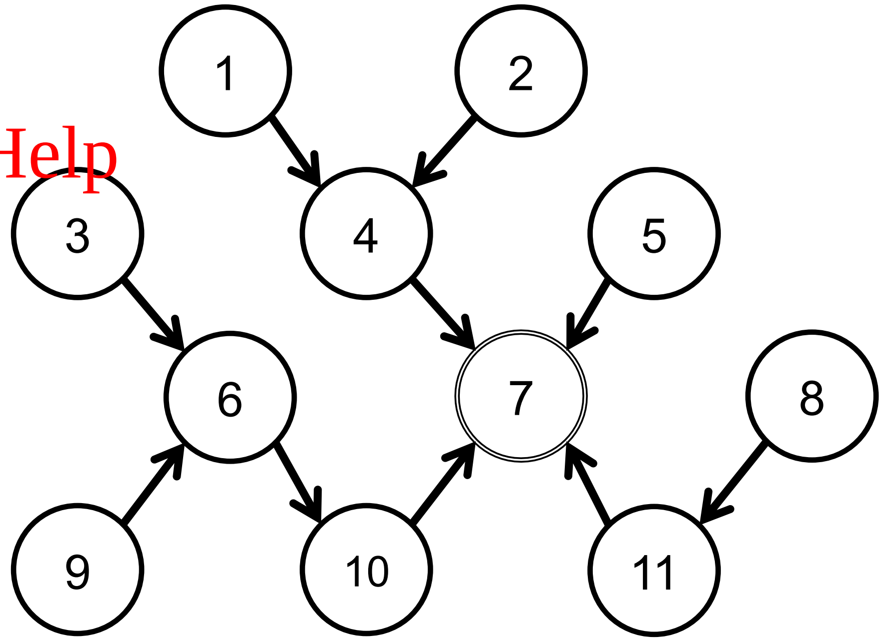
- Given an elimination tree (T, ϕ) with root r
  - For each node $i \neq r$ in the elimination tree, there is a unique neighbor of i that is closest to root r
  - A node i will be eliminated from the tree only after all its neighbors, except the one closest to the root, have been eliminated
  - When a node i is about to be eliminated, it will have a single neighbor j. Its current factor will have all variables but the separator $\mathbf{S}_{ij}$ eliminated and it will be multiplied by factor j



Elimination tree with directed edges pointing to neighbour closest to root node 7

Message-passing Formulation

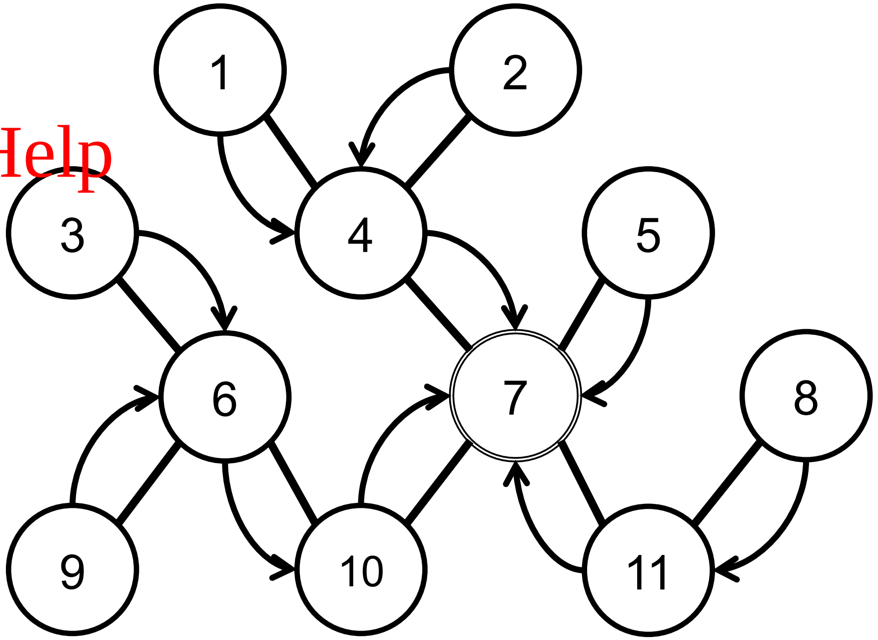
- Now, we view the elimination of node i with a single neighbor j as a process of passing a message M_{ij} from node i to j
  - When j receives a message, it multiplies it into its current factor ϕ_j
  - Node i cannot send a message to j until it has received all messages from neighbors $k \neq j$
  - After i receives these messages, its current factor will be $\phi_i \prod_{k \neq j} M_{ki}$
  - The message i send to j will be $\sum_{C_i \backslash S_{ij}} \phi_i \prod_{k \neq j} M_{ki}$



Elimination tree with directed edges pointing to neighbour closest to root node 7

Message-passing Algorithm

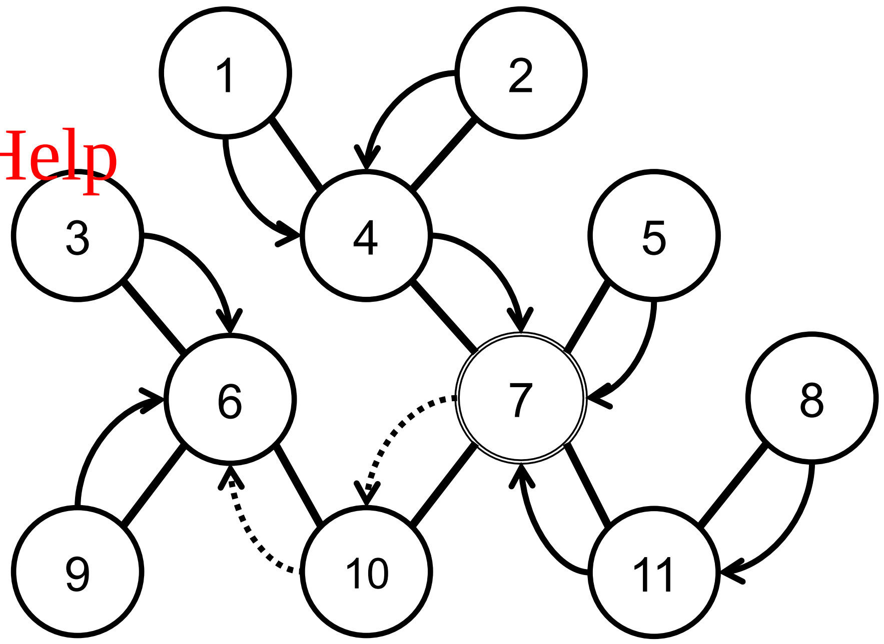
- We can now formulate FE as message-passing algorithm
  - To compute the marginal over some variables $\boldsymbol{Q}$
  - Select a root r in the elimination tree such that $\boldsymbol{Q} \subseteq \boldsymbol{C}_r$
  - Push messages towards the root r
  - When all messages into the root are available, we multiply them by ϕ_r and eliminate variables not in $\boldsymbol{Q}$
  - If our elimination tree has m nodes and $m-1$ edges, then $m-1$ messages need to be sent



10 messages are sent toward root node 7 in this 11-node elimination tree

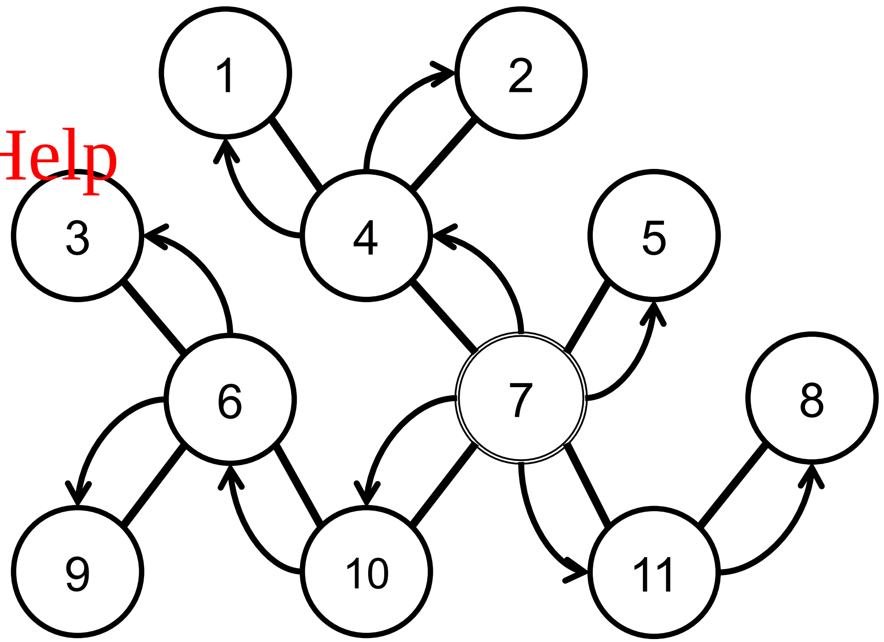
Message-passing and Reuse

- Suppose we want to compute the marginal over some other cluster $C_i, i \neq r$
  - We choose i as the new root and repeat the message-passing process
  - Some additional messages need to be passed, but not as many as $m - 1$, assuming we saved the messages sent to r
  - In the figure, out 10 messages sent toward node 6, eight messages have already been computed when 7 was the root



Message-passing and Reuse

- The key observation is that if we choose every node as a root, the total number of messages is $2(m-1)$
  - There are $m-1$ edges and two distinct messages per edge
  - These messages are usually computed in two phases
    - Inward: we direct messages toward a root r
    - Outward: we direct messages away from the root r



Outward phase with node 7 as root

Message-passing: Complexity

- A message from i to j is computed by multiplying a few factors
  - The factor that results from the multiplication must be over the cluster of node i
  - Hence, the complexity of both multiplication and summation is $O(\exp(w))$, where w is the size of cluster C_i
  - The width of an elimination tree is the size of its maximal cluster -1
  - Hence, if w is the width, then the cost of any message is $O(\exp(w))$
  - Since we have $2(m-1)$, the total cost is $O(m\exp(w))$

Joint Marginals and Evidence

- Given some evidence e we want to use factor elimination to compute the joint marginal $P(C_i, e)$ for each cluster C_i in the elimination tree, we can
  1. Reduce each factor f given the evidence $\boldsymbol{e}$, leading to a set of reduced factors $f^{\boldsymbol{e}}$
  2. Introduce an evidence indicator λ_E for every variable E in evidence $\boldsymbol{e}$. λ_E is a factor over variable E that captures the value of E in evidence e: $\lambda_E(e) = 1$ if e is consistent with evidence $\boldsymbol{e}$ and $\lambda_E(e) = 0$ otherwise

$\boldsymbol{e}: \{A = true, B = false\}$

A	λ_A
a	1
$\bar{a}$	0

B	λ_B
b	0
$\bar{b}$	1

Joint Marginals and Evidence

- The first method is more efficient if we plan to compute marginals with respect to only one piece of evidence $\boldsymbol{e}$
- The second method is more efficient to compute marginals with respect to multiple pieces of evidence, $\boldsymbol{e}_1 , \dots , \boldsymbol{e}_n$
  - While trying to reuse messages across different pieces of evidence
  - This method is implemented by assigning the evidence indicator λ_E to a node i in the elimination tree while ensuring that $E \in C_i$.
  - As a result, the clusters and separators of the elimination tree will remain intact and so will its width

$\boldsymbol{e}: \{A = true, B = false\}$

A	λ_A
a	1
$\bar{a}$	0

B	λ_B
b	0
$\bar{b}$	1

Factor Elimination Algorithm: FE

Input: Network N, a set of variables $\boldsymbol{Q}$ in the network, an elimination tree (T, ϕ)

Output: joint marginal $P(\boldsymbol{C}_i, \boldsymbol{e})$ for each node i in the elimination tree

for each variable E in evidence $\boldsymbol{e}$ **do**

  $i \leftarrow$ node in tree T such that $E \in \ C_i$

  $\lambda_E \leftarrow$ evidence indicator for variable E # $\lambda_E = 1$ if $e \sim \boldsymbol{e}$ and $\lambda_E(e) = 0$ otherwise

  $\phi_i \leftarrow \phi_i \lambda_E$ # entering evidence at node i

choose a root node r in the tree T

pull/collect messages towards root r

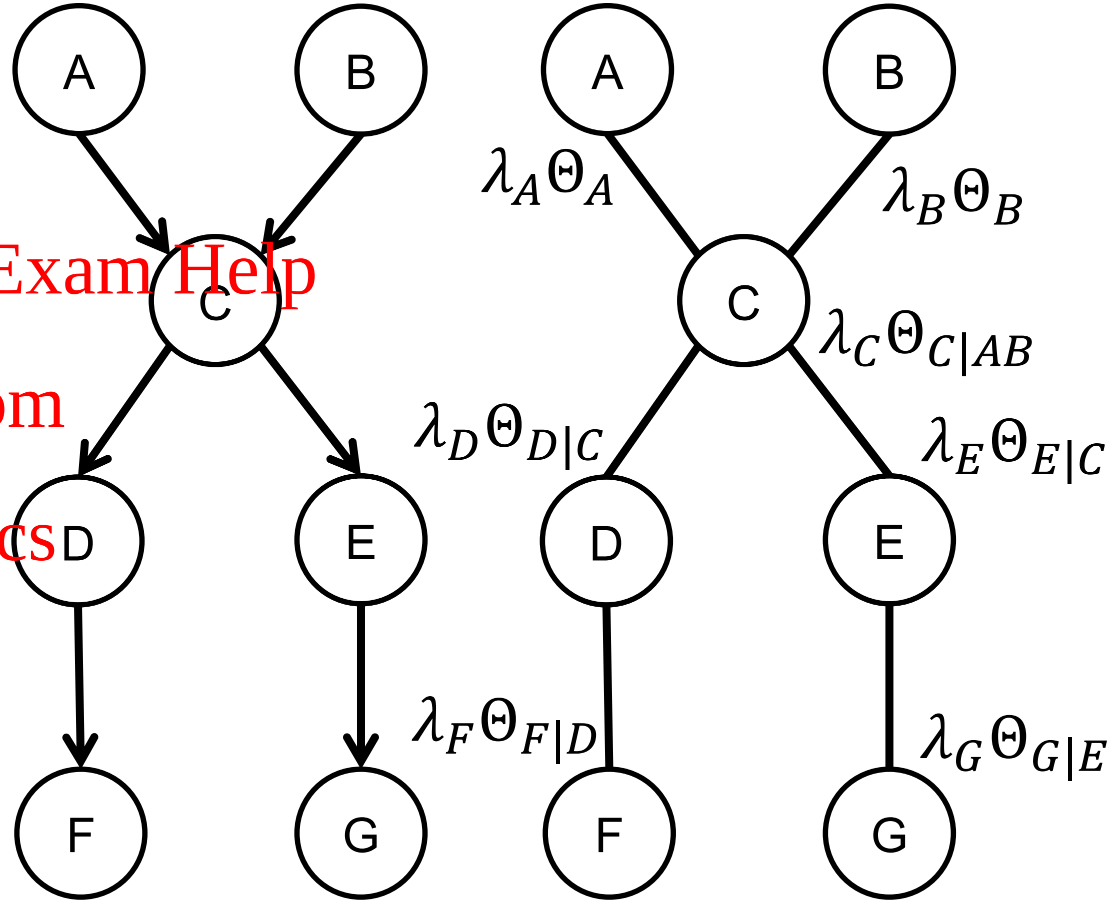
push/distribute messages away from root r

return $\phi_i \prod_k M_{ki}$ for each node i in the tree T # joint marginal $P(\boldsymbol{C}_i, \boldsymbol{e})$

- This algorithm uses the second method for accommodating evidence
- It computes joint marginals using two phases of message passing
- If we save the messages across different runs of the algorithm, then we can reuse these messages as long as they are not invalidated when the evidence changes
- When the evidence at node i changes, we need to invalidate all messages that depend on the factor at that node
- These messages happen to be the ones directed away from node i in the elimination tree

Polytree Algorithm

- When the network has a polytree structure
  - We can use an elimination tree that corresponds to the polytree structure
  - This special case of the algorithm is known as polytree algorithm or belief propagation algorithm. We will discuss them later
  - If k is the maximum number of parents in any node in the polytree, then k is also the width of the elimination tree
  - The time and space complexity are $O(n\exp(k))$, where n is the number of nodes in the polytree

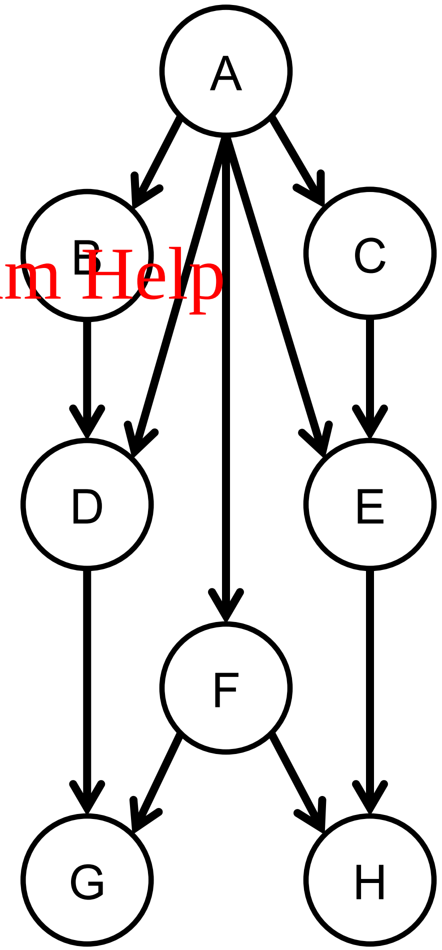


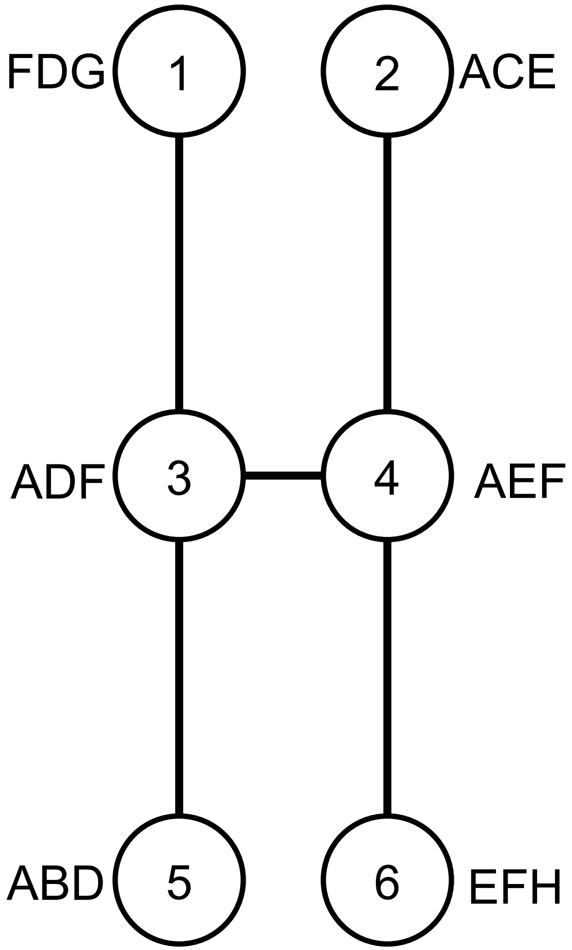
Jointree

- There are different methods for constructing elimination trees
  - But the method we discuss next will be based on an influential tool known as a *jointree*
  - It is this tool that gives factor elimination its traditional name: *the jointree algorithm*
- It is possible to phrase the factor elimination algorithm directly on jointrees without explicit mention elimination trees
  - This is indeed how the algorithm is classically described and we provide such a description
  - We start defining a jointree

Jointree

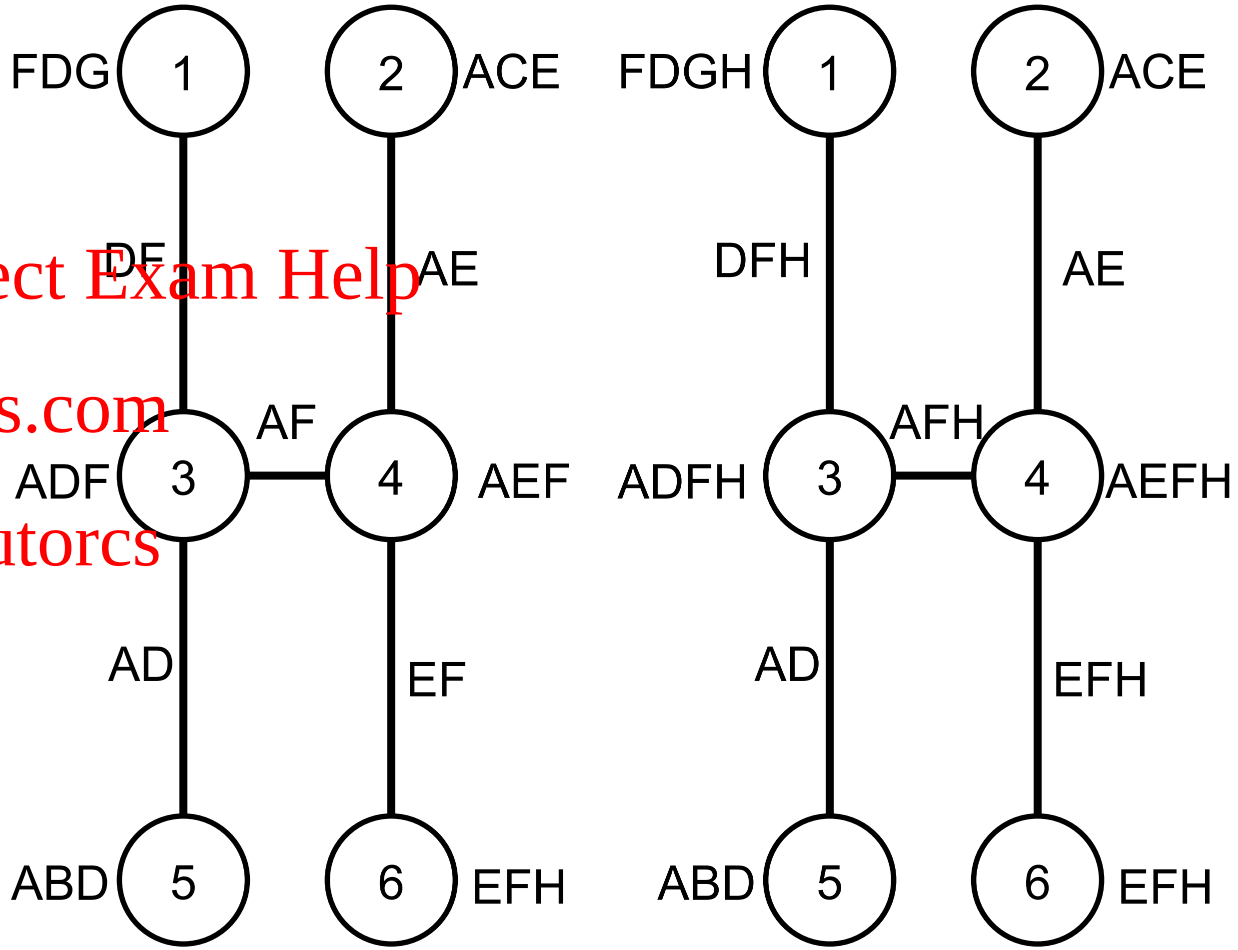
- A *jointree* for a network G is a pair $(T, \boldsymbol{C})$ where T is a tree and $\boldsymbol{C}$ is a function that maps each node i in the tree T into a label $\boldsymbol{C}_i$, called *cluster*.
- The jointree must satisfy the properties
  - The cluster $\boldsymbol{C}_i$ is a set of nodes from G
  - Each factor in G must appear in some cluster $\boldsymbol{C}_i$
  - If a variable appears in two clusters $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$, it must appear in every cluster $\boldsymbol{C}_k$ on the path connecting nodes i and j in the jointree. This is known as *jointree* or *running intersection* property





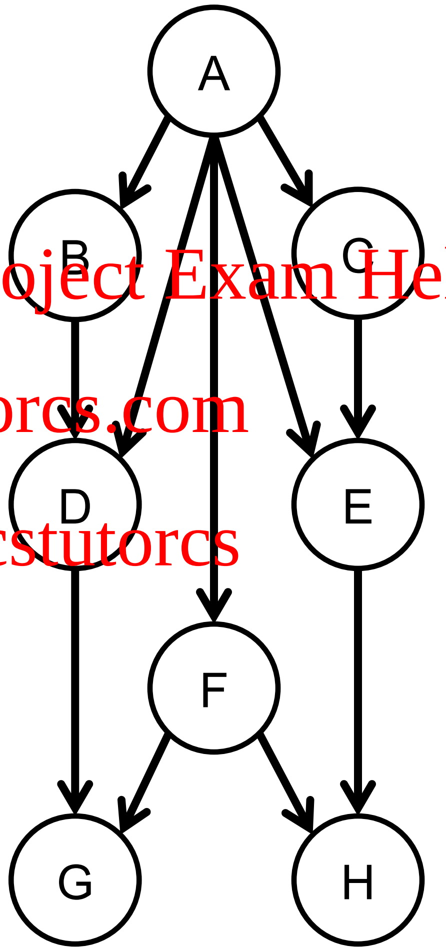
Jointree

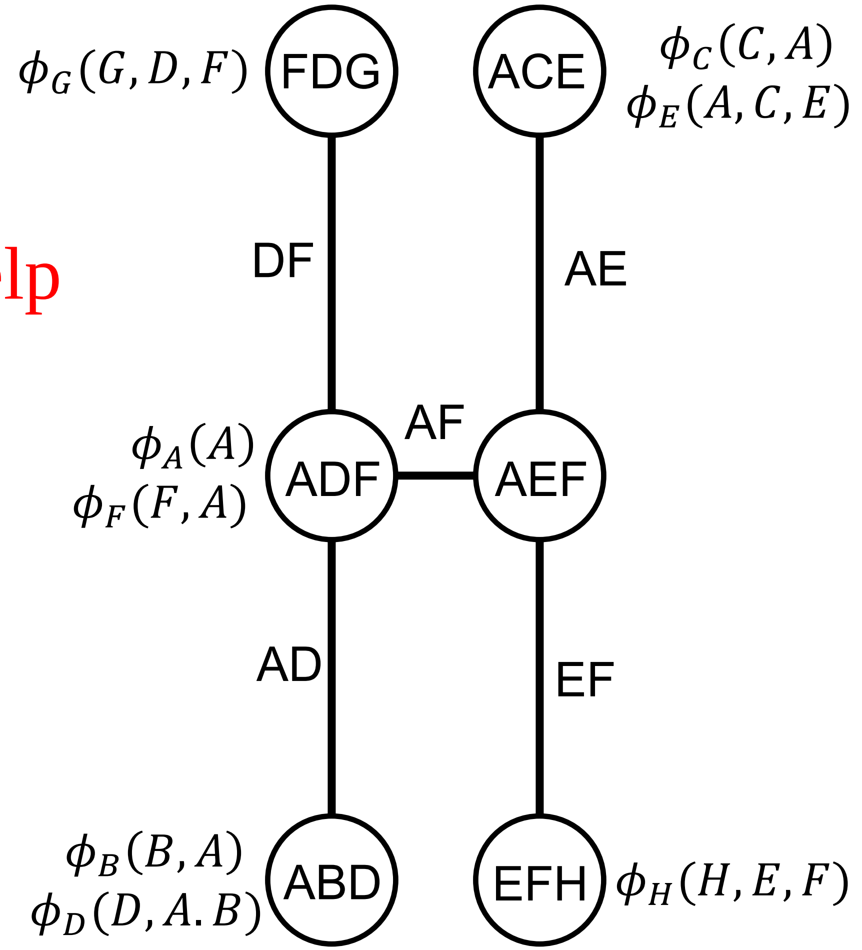
- The separator of edge $i - j$ in a jointree is denoted by $\boldsymbol{S}_{ij}$ and defined as $\boldsymbol{C}_i \cap \boldsymbol{C}_j$
  - The width of a jointree is defined as the size of its largest cluster minus one
- A jointree is *minimal if* it ceases to be a jointree for G once we remove a variable from one of its cluster
  - Left jointree is minimal but the right one is not



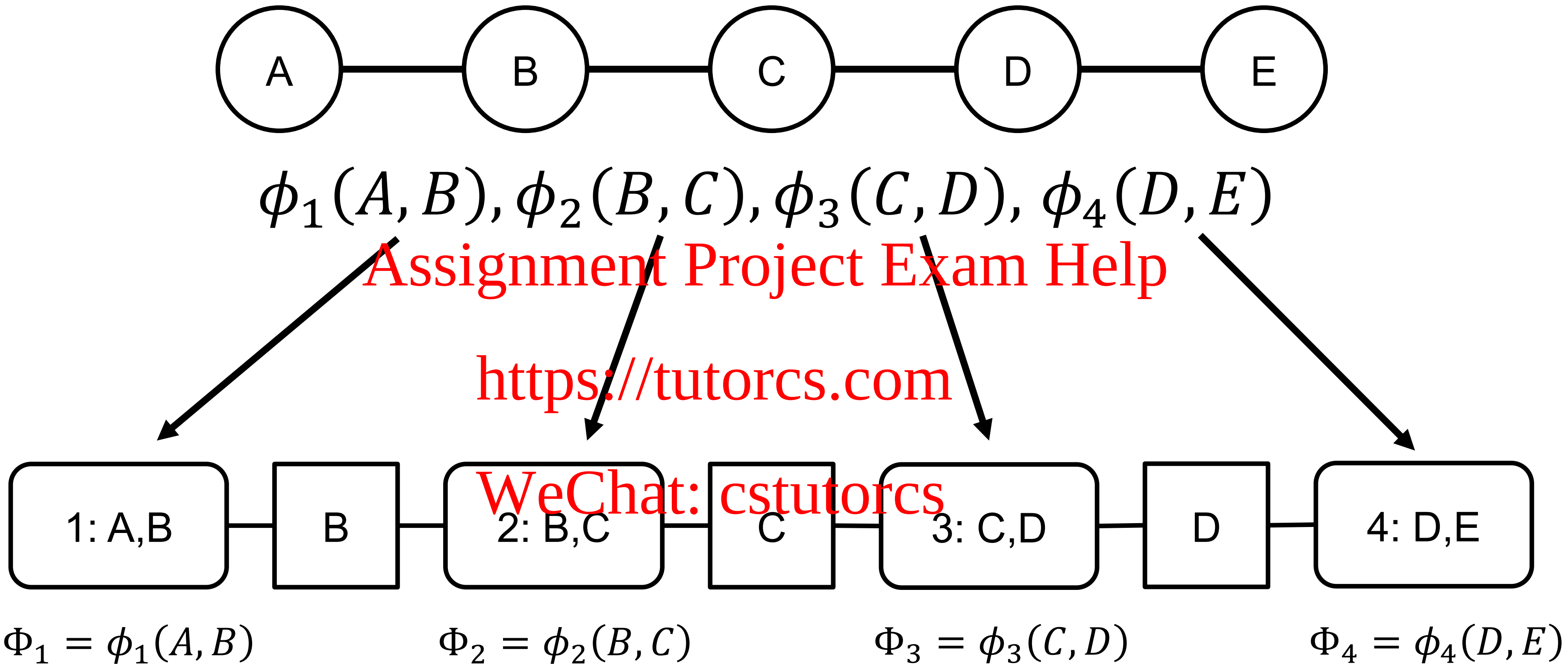
The Jointree Algorithm

- The classical jointree algorithm is:
  - Construct jointree (T, C) for a given network
  - Assign each factor ϕ_i to a cluster that contains $vars(\phi_i)$
  - Assign each evidence indicator λ_X to a cluster that contains X
  - Propagate messages in the jointree between the clusters
  - After passing two messages per edge in the jointree, we can compute the marginals $P(\boldsymbol{C}, \boldsymbol{e})$ for every cluster $\boldsymbol{C}$

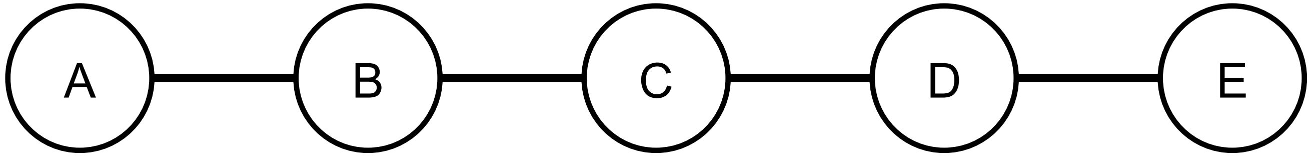


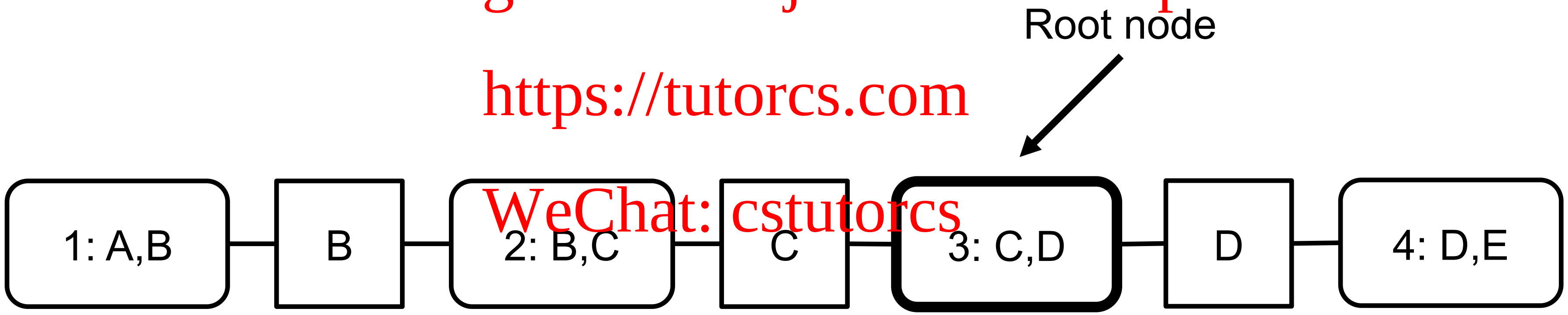


Message-passing: Example



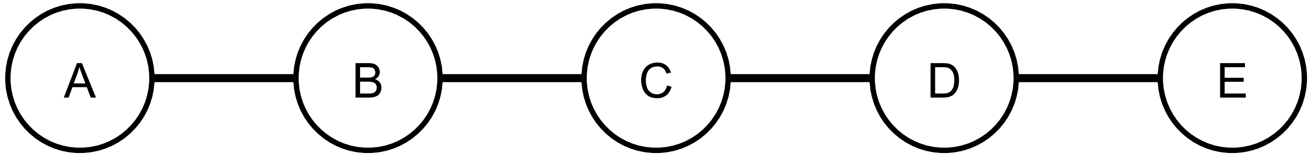
Message-passing: Example



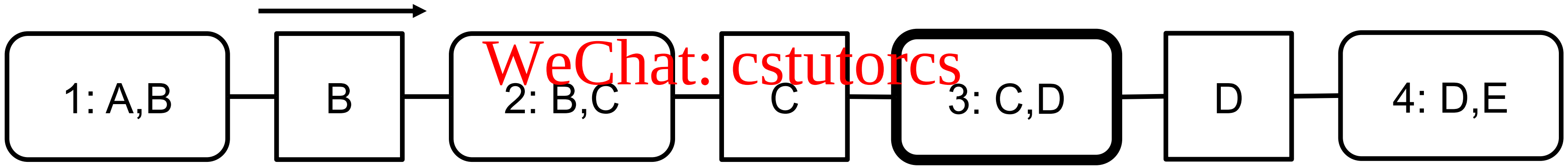


$\Phi_1 = \phi_1(A, B)$ $\Phi_2 = \phi_2(B, C)$ $\Phi_3 = \phi_3(C, D)$ $\Phi_4 = \phi_4(D, E)$

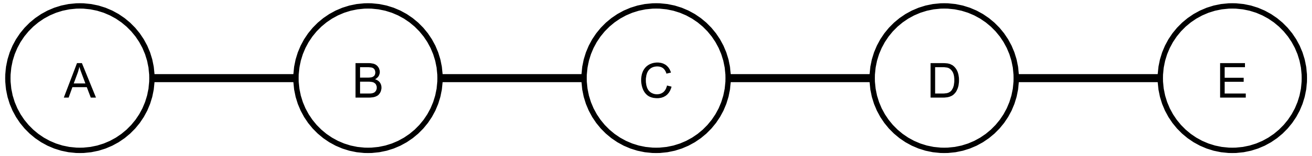
Message-passing: Example



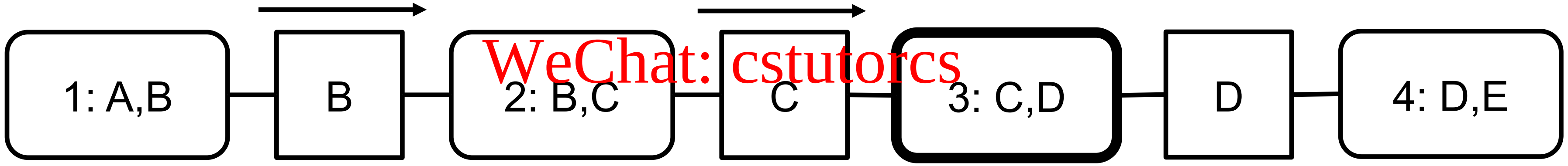
$$M_{1,2} = \sum_{A} \Phi_1$$



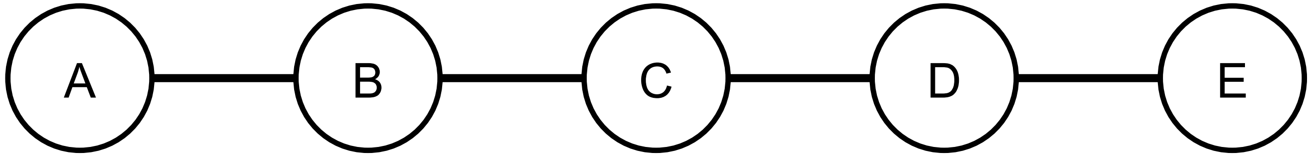
Message-passing: Example



$$M_{1,2} = \sum_{A} \Phi_1 \qquad M_{2,3} = \sum_{B} \Phi_2 \, M_{1,2}$$

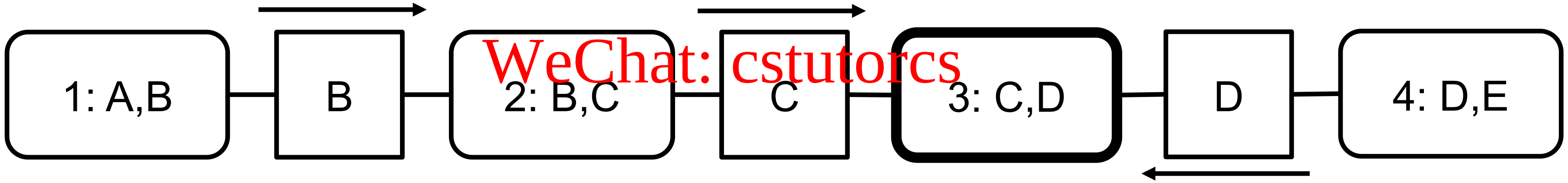


Message-passing: Example



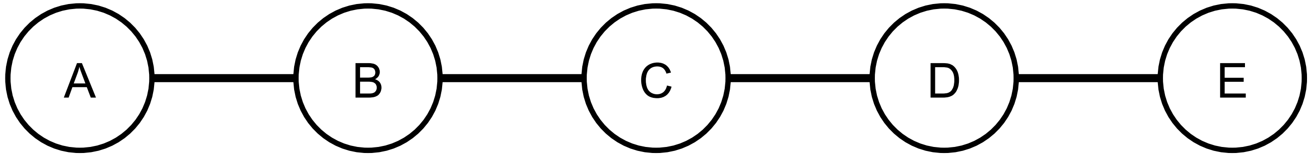
$$M_{1,2} = \sum_{A} \Phi_1$$

$$M_{2,3} = \sum_{B} \Phi_2 \, M_{1,2}$$



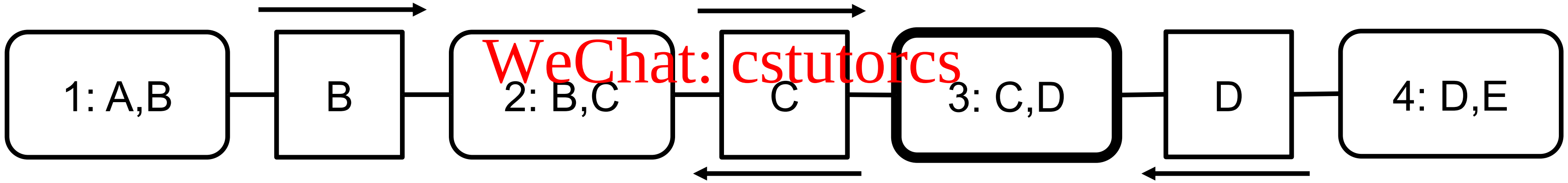
$$M_{4,3} = \sum_{E} \Phi_4$$

Message-passing: Example



$$M_{1,2} = \sum_{A} \Phi_1$$

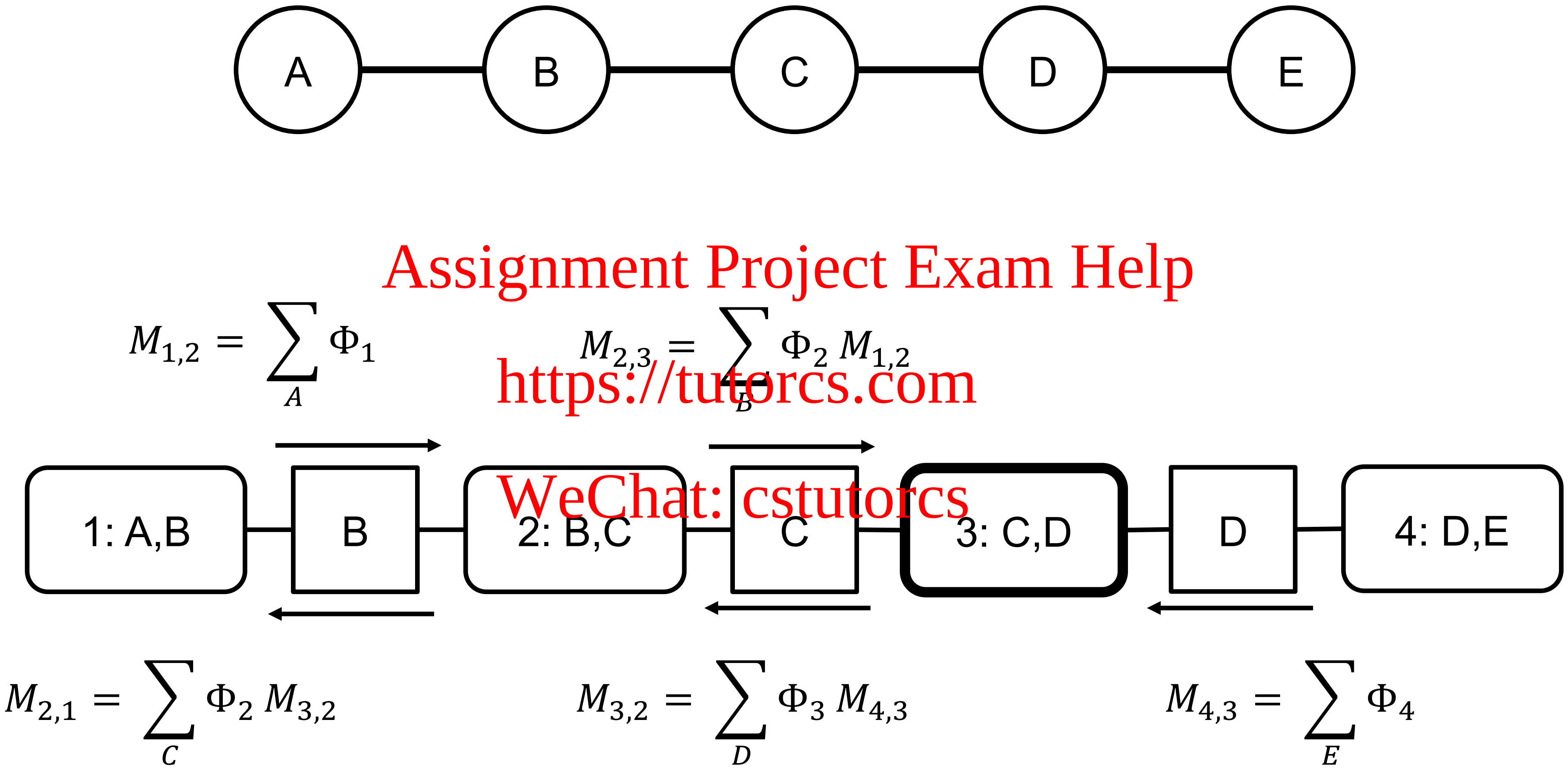
$$M_{2,3} = \sum_{B} \Phi_2 \, M_{1,2}$$



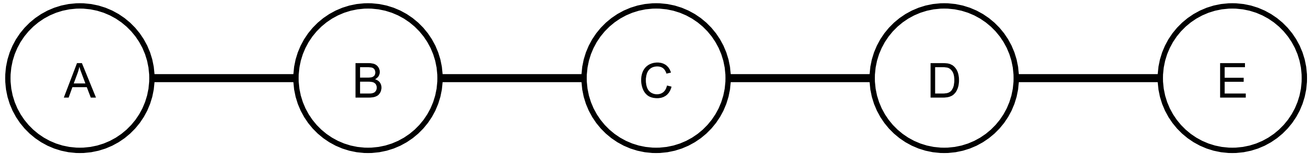
$$M_{3,2} = \sum_{D} \Phi_3 \, M_{4,3}$$

$$M_{4,3} = \sum_{E} \Phi_4$$

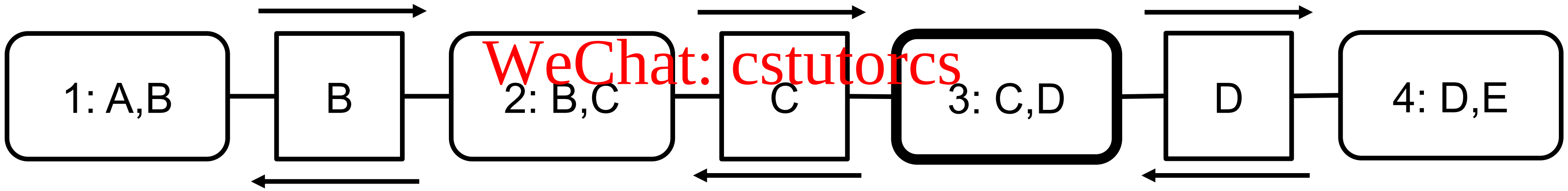
Message-passing: Example



Message-passing: Example



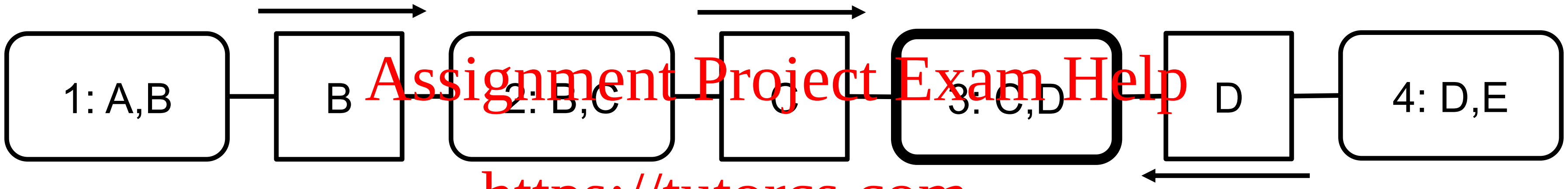
$$M_{1,2} = \sum_{A} \Phi_1 \qquad M_{2,3} = \sum_{B} \Phi_2 \, M_{1,2} \qquad M_{3,4} = \sum_{C} \Phi_3 \, M_{2,3}$$



$$M_{2,1} = \sum_{C} \Phi_2 \, M_{3,2} \qquad M_{3,2} = \sum_{D} \Phi_3 \, M_{4,3} \qquad M_{4,3} = \sum_{E} \Phi_4$$

Message-passing: Example

$$M_{1,2} = \sum_{A} \Phi_1 \qquad M_{2,3} = \sum_{B} \Phi_2 \, M_{1,2}$$



$$M_{4,3} = \sum_{E} \Phi_4$$

$$B_3(C,D) = \Phi_3 M_{2,3} M_{4,3}$$

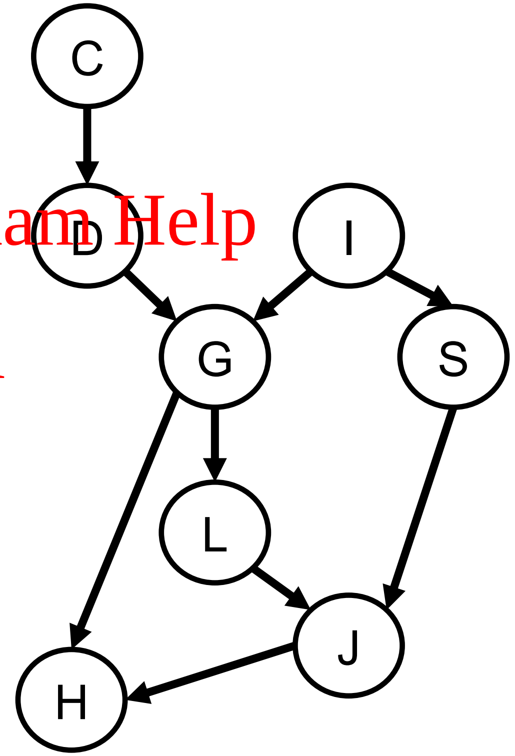
$$= \Phi_3 \left(\sum_{B} \Phi_2 \, M_{1,2} \right) \sum_{E} \Phi_4$$

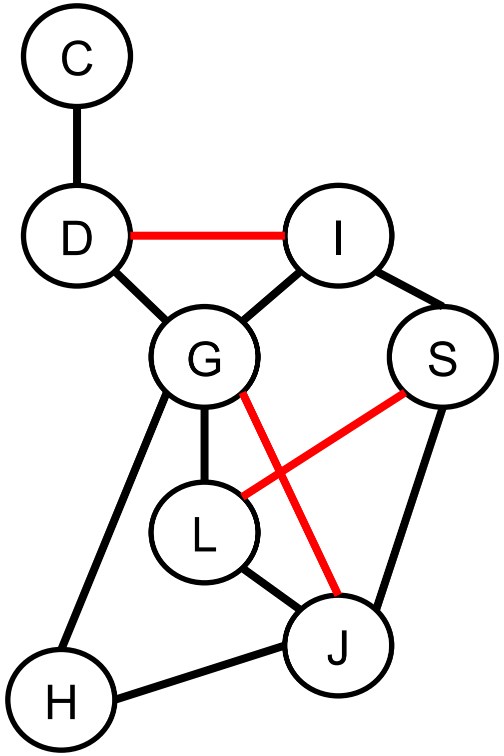
$$= \Phi_3 \left(\sum_{B} \Phi_2 \sum_{A} \Phi_1 \right) \sum_{E} \Phi_4$$

Product of all factors marginalising hidden variables in the correct order

Another Jointree Example

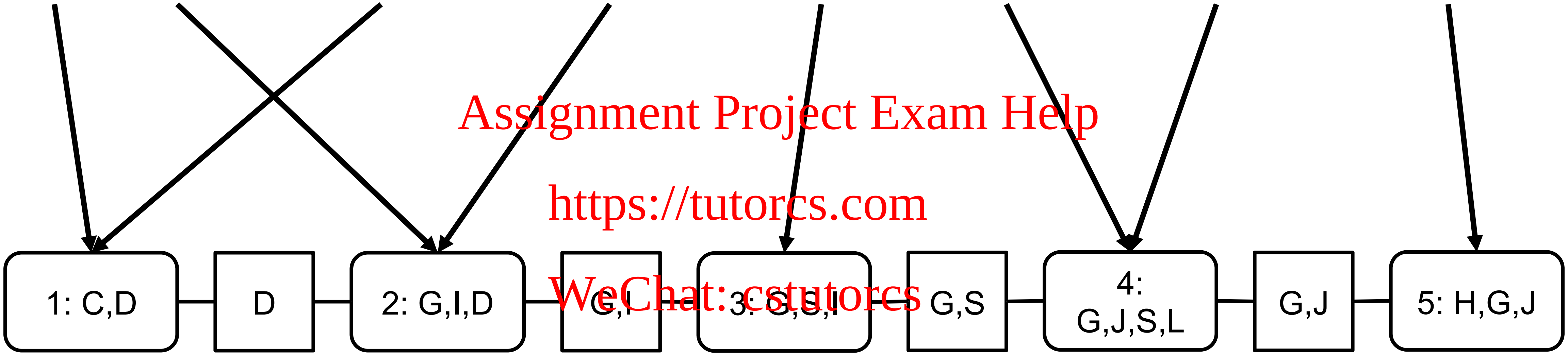
- Let's see another example with a larger Bayesian network
  - These are the factors
  - $\phi_c(C)$, $\phi_I(I)$, $\phi_D(C,D)$, $\phi_G(G,D,I)$, $\phi_s(I,S)$, $\phi_L(L,G)$, $\phi_J(J,L,S)$, $\phi_H(H,G,J)$

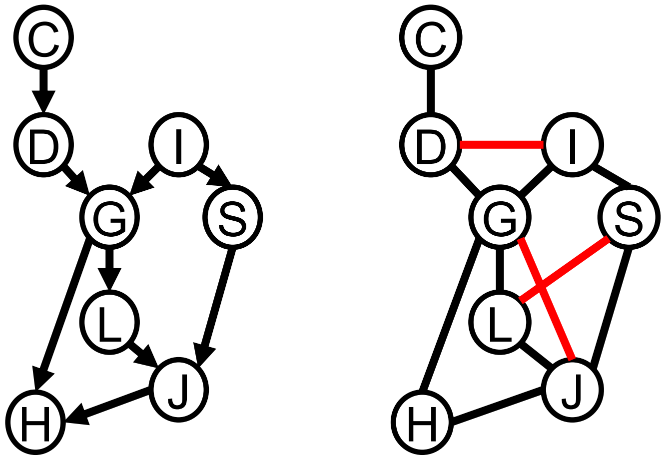




Another Jointree Example

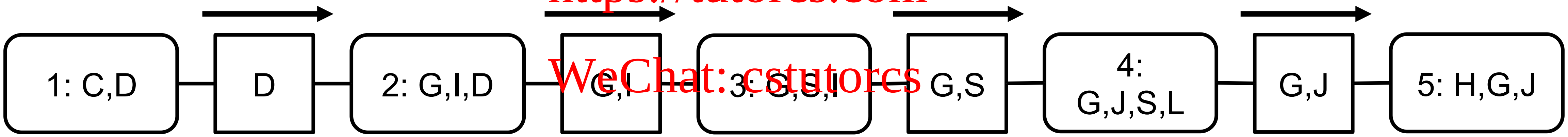
$$\phi_C(C)\ \phi_I(I)\ \phi_D(C,D)\ \phi_G(G,D,I)\ \phi_S(I,S)\ \phi_L(L,G)\ \phi_J(J,L,S)\ \phi_H(H,G,J)$$





Another Jointree Example

$M_{1,2} = \sum_{C} \Phi_1$ $\quad M_{2,3} = \sum_{D} \Phi_2 \, M_{1,2}$ $\quad M_{3,4} = \sum_{I} \Phi_3 \, M_{2,3}$ $\quad M_{4,5} = \sum_{S,L} \Phi_4 \, M_{3,4}$



$M_{2,1} = \sum_{G,I} \Phi_2 \, M_{3,2}$ $\quad M_{3,2} = \sum_{S} \Phi_3 \, M_{4,3}$ $\quad M_{4,3} = \sum_{J,L} \Phi_4 M_{5,4}$ $\quad M_{5,4} = \sum_{H} \Phi_5$

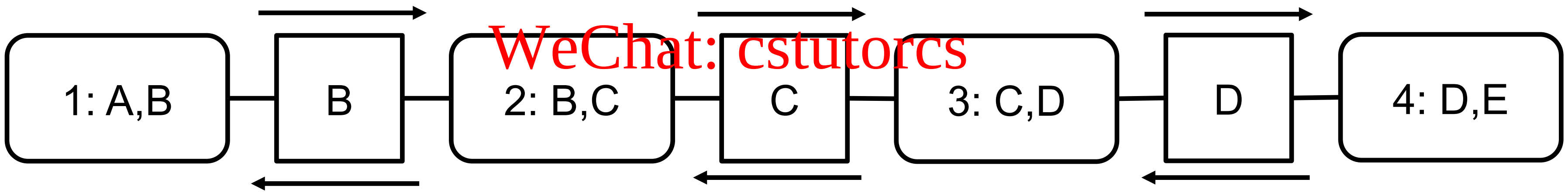
Answering Queries

- For marginal probability queries on variables that appear together in a clique
  - Sum out irrelevant variables from any clique containing those variables
- For posterior marginal queries
  - We have evidence $\boldsymbol{E} = \boldsymbol{e}$ and query $\boldsymbol{Q}$
- If $\boldsymbol{Q}$ and $\boldsymbol{E}$ appear in a same cluster
  - That is, $\boldsymbol{Q} \cup \boldsymbol{E} \subseteq \boldsymbol{C}_i$ for some cluster $\boldsymbol{C}_i$
  - We can eliminate entries that do not agree with evidence $\boldsymbol{e}$
  - Sum out irrelevant variables and renormalize
- If $\boldsymbol{Q}$ does not appear in a cluster with $\boldsymbol{E}$
  - Set evidence indicators in one or more clusters containing $\boldsymbol{E}$
  - Propagate messages along path to cluster containing $\boldsymbol{Q}$
  - Sum out irrelevant variables

Answering Queries

- Suppose we want to compute $P(C, a)$
  - We need set evidence and propagate some messages again
  - Let's now evidence indicators instead of elimination rows

$$M_{1,2} = \sum_A \Phi_1 \qquad M_{2,3} = \sum_B \Phi_2 \, M_{1,2} \qquad M_{3,4} = \sum_C \Phi_3 \, M_{2,3}$$



$$M_{2,1} = \sum_C \Phi_2 \, M_{3,2} \qquad M_{3,2} = \sum_D \Phi_3 \, M_{4,3} \qquad M_{4,3} = \sum_E \Phi_4$$

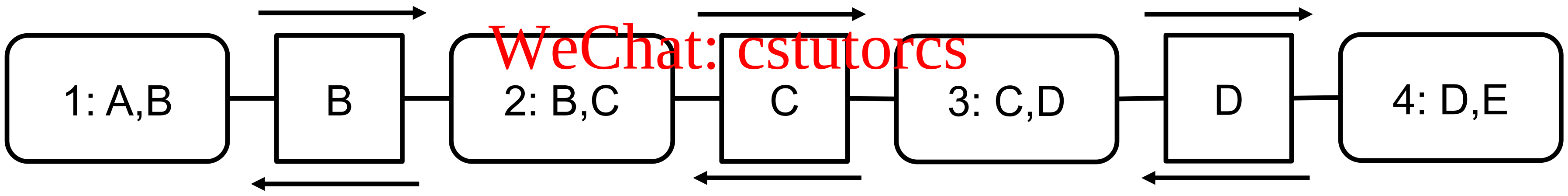
A	B	$B(A,B)$
a	b	0.8
a	$\bar{b}$	0.2
$\bar{a}$	b	0.1
$\bar{a}$	$\bar{b}$	0.9

Answering Queries

- Suppose we want to compute $P(C, a)$
  - We need set evidence and propagate some messages again
  - Let's now evidence indicators instead of elimination rows

$$M_{1,2} = \sum_A \Phi_1 \qquad M_{2,3} = \sum_B \Phi_2\, M_{1,2} \qquad M_{3,4} = \sum_C \Phi_3\, M_{2,3}$$

A	$\lambda_A(A)$
a	1
$\bar{a}$	0



A	B	$B(A,B)$
a	b	0.8
a	$\bar{b}$	0.2
$\bar{a}$	b	0.1
$\bar{a}$	$\bar{b}$	0.9

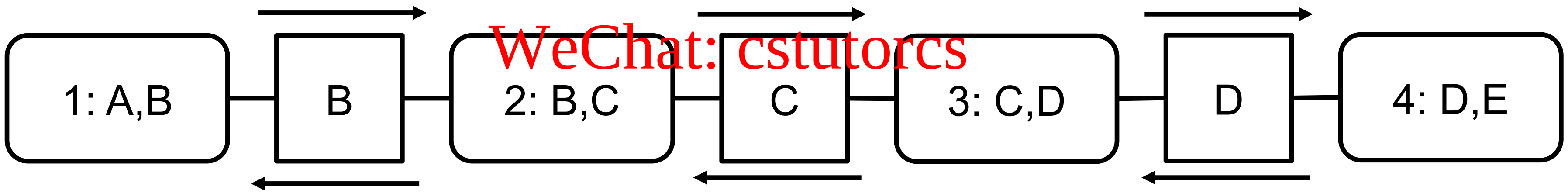
$$M_{2,1} = \sum_C \Phi_2\, M_{3,2} \qquad M_{3,2} = \sum_D \Phi_3\, M_{4,3} \qquad M_{4,3} = \sum_E \Phi_4$$

Answering Queries

update: $M_{1,2} = \sum_{A} \Phi_1$

update: $M_{2,3} = \sum_{B} \Phi_2 \, M_{1,2}$

irrelevant: $M_{3,4} = \sum_{C} \Phi_3 \, M_{2,3}$



A	$\lambda_A(A)$
a	1
$\bar{a}$	0

A	B	$B(A,B)$
a	b	0.8
a	$\bar{b}$	0.2
$\bar{a}$	b	0.1
$\bar{a}$	$\bar{b}$	0.9

$M_{2,1} = \sum_{C} \Phi_2 \, M_{3,2}$: same

$M_{3,2} = \sum_{D} \Phi_3 \, M_{4,3}$: same

$M_{4,3} = \sum_{E} \Phi_4$: same

The Jointree Algorithm

- There are two main methods for propagating messages in a jointree, known as
  - The Shenoy-Shafer architecture and
  - The Hugin architecture

- The methods differ in both their space and time complexity
  - The Shenoy-Shafer architecture generally require less space but more time on an arbitrary jointree

The Shenoy-Shafer Architecture

- Evidence $\boldsymbol{e}$ is entered into the jointree through evidence indicators
- A cluster is selected as the root
- Message propagation in two phases
  - Inward: toward the root
  - Outward: away from the root
- Inward phase is also known as the *collect* or *pull* phase
  - The outward phase is known as the *distribute* or *push* phase
- Node i send a message to j only when it has received messages from all other neighbors k

- A message from node i to j is a factor

$$M_{ij} \stackrel{\text{def}}{=} \sum_{\boldsymbol{C}_i \backslash \boldsymbol{S}_{ij}} \Phi_i \prod_{k \neq j} M_{ki}$$

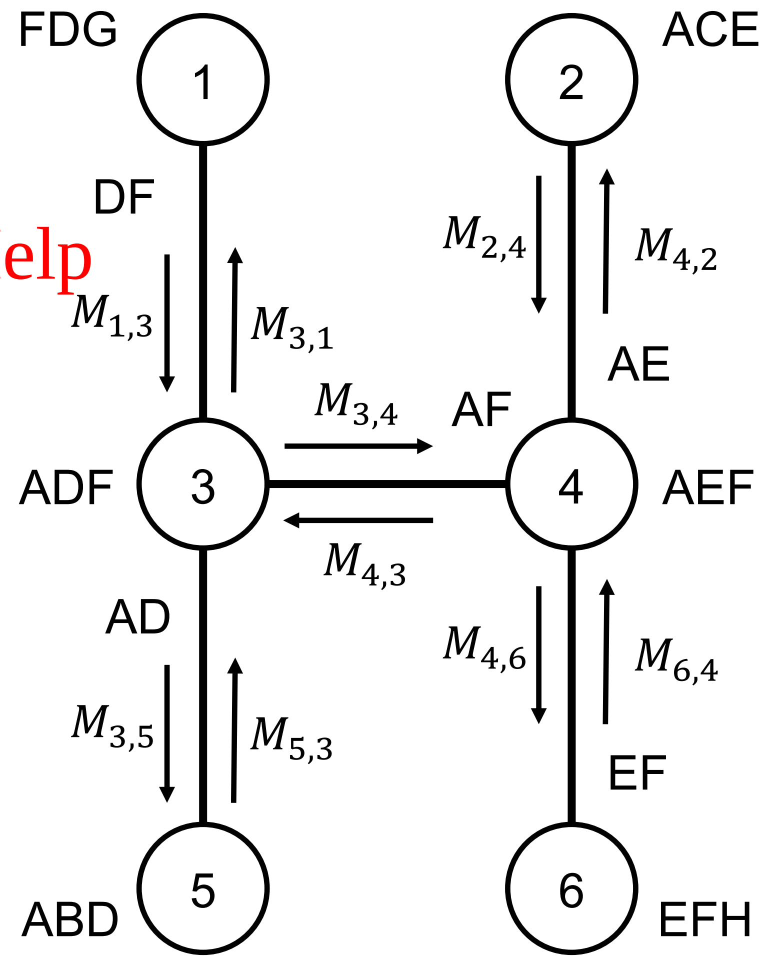
  - Φ_i is the multiplication of all factors associated with node i (including evidence indicators)
- Once finished, we have the following for each cluster i in the jointree

$$P(\boldsymbol{C}_i) = \Phi_i \prod_k M_{ki}$$

- We can compute the joint marginal for any subset of variables that is included in a cluster

Shenoy-Shafer: Space

- We need two factors for each separator $\boldsymbol{S}_{ij}$
  - One factor stores the message from cluster i to cluster j
  - The other stores the message from j to i
- There is no need to construct a factor over all cluster variables
  - The space complexity is not exponential in the size of jointree clusters
  - But only in the size of jointree separators



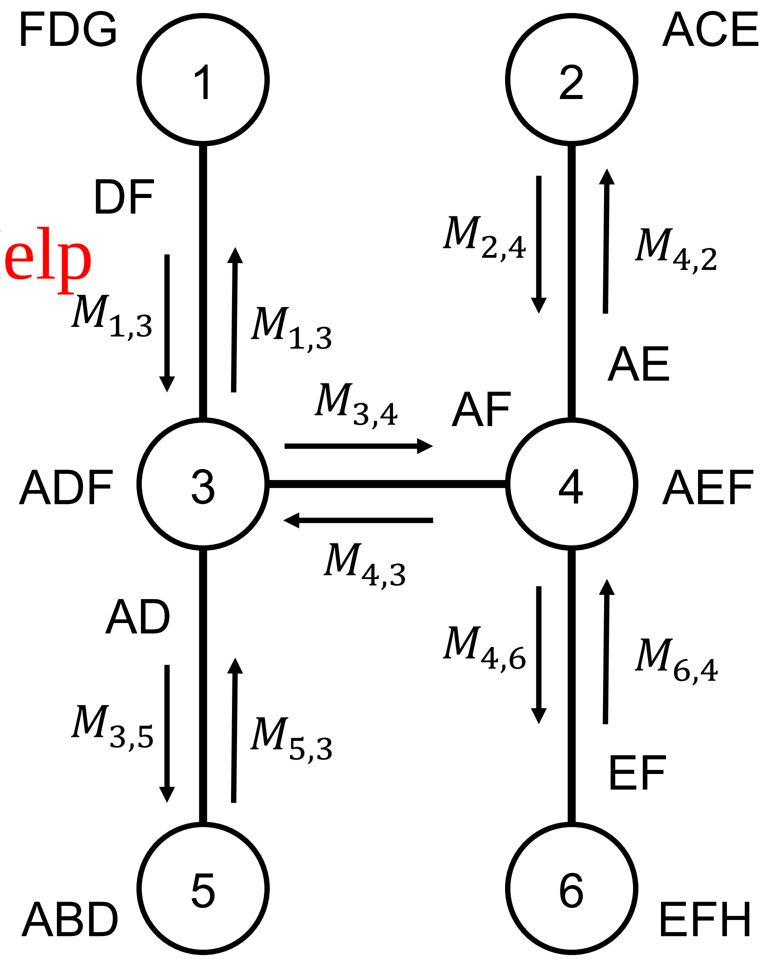
The Hugin Architecture

- Evidence $\boldsymbol{e}$ is entered into the jointree through evidence indicators
- A cluster is selected as the root
- Message propagation in two phases
  - Inward: toward the root
  - Outward: away from the root
- Inward phase is also known as the *collect* or *pull* phase
  - The outward phase is known as the *distribute* or *push* phase
- Node i send a message to j only when it has received messages from all other neighbors k

- Each separator $\boldsymbol{S}_{ij}$ has a single factor Ψ_{ij}
  - With each entry initialized to 1
- Each cluster $\boldsymbol{C}_i$ has a factor Ψ_i
  - Which is initialized to $\Phi_i \prod_j \Psi_{ij}$
  - Where Φ_i is the product of factors (including evidence indicators) assigned to node i
- When node i is ready to send a message to node j, it does the following:
  - Saves the factor Ψ_{ij} into Ψ_{ij}^{old}
  - Computes the new factor $\Psi_{ij} \leftarrow \sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{ij}} \Psi_i$
  - Computes the message: $M_{ij} = \Psi_{ij}/\Psi_{ij}^{old}$
  - Multiplies M_{ij} into node j: $\Psi_j \leftarrow \Psi_i M_{ij}$

The Hugin Architecture

- After the inward and outward passes, we have the following for each node i
  - $P(\boldsymbol{C}_i, \boldsymbol{e}) = \Psi_i$
- The Hugin propagation scheme also guarantees the following for each edge $i - j$:
  - $P(\boldsymbol{S}_{ij}, \boldsymbol{e}) = \Psi_{ij}$
- The space requirements for the Hugin architecture
  - One factor for each cluster and one factor for each separator
  - The cluster factors are usually much larger than separator factors. More space than Shenoy-Shafer
  - However, Hugin is faster. We do not need to multiply all the factors to compute a new message



Conclusion

- Factor elimination is an alternate view of variable elimination
  - We decompose the graphs eliminating one factor at a time, instead of one variable
  - This view provides an efficient approach to answer queries over cluster variables
- The key idea is to use a message-passing formulation
  - Saving the intermediate computations that can be used later to answer queries
  - Message-passing also forms the basis of approximate algorithm known as belief propagation
- Tasks
  - Read Chapter 7 from the textbook (Darwiche)