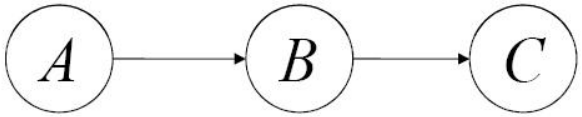
Reasoning with Bayesian Networks

Lecture 3: Inference by Factor Elimination, Inference by Conditioning

Jinbo Huang

NICTA and ANU

Review of Variable Elimination



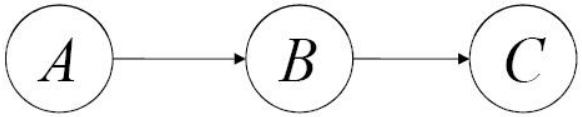
Compute Pr(C): eliminate A, then B

A	Θ_A
true	.6
false	.4

A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

Review of Variable Elimination



A	Θ_A
true	.6
false	.4

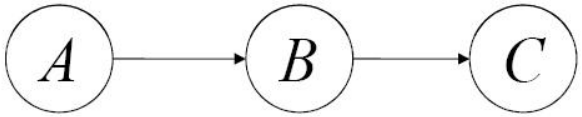
A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

Compute $\Pr(C)$: eliminate A, then B

Two factors mention A: Θ_A, $\Theta_{B|A}$

Review of Variable Elimination



A	Θ_A
true	.6
false	.4

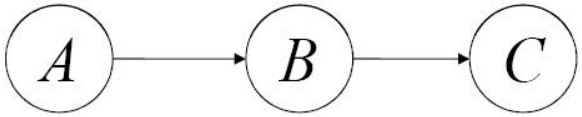
A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

Multiply Θ_A and $\Theta_{B|A}$

A	B	$\Theta_A\Theta_{B\|A}$
true	true	.54
true	false	.06
false	true	.08
false	false	.32

Review of Variable Elimination



A	Θ_A
true	.6
false	.4

A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

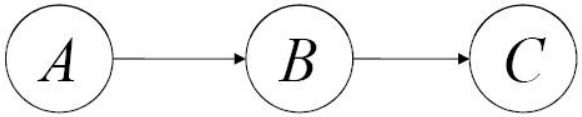
Multiply Θ_A and $\Theta_{B|A}$

A	B	$\Theta_A\Theta_{B\|A}$
true	true	.54
true	false	.06
false	true	.08
false	false	.32

Sum out A

B	$\sum_A \Theta_A\Theta_{B\|A}$
true	$.62 = .54 + .08$
false	$.38 = .06 + .32$

Review of Variable Elimination



A	Θ_A
true	.6
false	.4

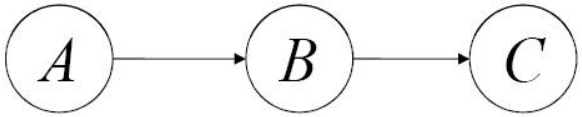
A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

Two factors left, $\Theta_{C|B}$ & $\sum_A \Theta_A \Theta_{B|A}$, multiply

B	C	$\Theta_{C\|B} \sum_A \Theta_A \Theta_{B\|A}$
true	true	.186
true	false	.434
false	true	.190
false	false	.190

Review of Variable Elimination



A	Θ_A
true	.6
false	.4

A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

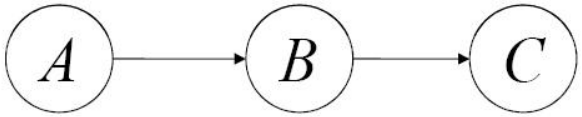
Two factors left, $\Theta_{C|B}$ & $\sum_A \Theta_A \Theta_{B|A}$, multiply

B	C	$\Theta_{C\|B} \sum_A \Theta_A \Theta_{B\|A}$
true	true	.186
true	false	.434
false	true	.190
false	false	.190

Sum out B

C	$\sum_B \Theta_{C\|B} \sum_A \Theta_A \Theta_{B\|A}$
true	.376
false	.624

Review of Variable Elimination



A	Θ_A
true	.6
false	.4

A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

Biggest factor produced: 4 rows

Two factors left, $\Theta_{C|B}$ & $\sum_A \Theta_A \Theta_{B|A}$, multiply

B	C	$\Theta_{C\|B} \sum_A \Theta_A \Theta_{B\|A}$
true	true	.186
true	false	.434
false	true	.190
false	false	.190

Sum out B

C	$\sum_B \Theta_{C\|B} \sum_A \Theta_A \Theta_{B\|A}$
true	.376
false	.624

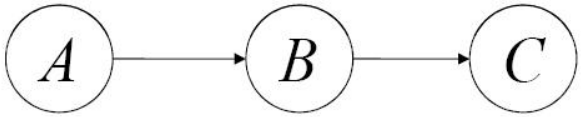
Review of Variable Elimination

- To compute marginal over variables $\mathbf{Q}$, eliminate other variables, one at a time
- Eliminate variable X by multiplying factors containing X and summing out X
- $\prod$ of remaining factors give marginal over $\mathbf{Q}$

Factor Elimination

- To compute marginal over variables $\mathbf{Q}$, eliminate all factors except one that contains $\mathbf{Q}$
- Eliminate factor Φ by summing out "private" variables and multiplying result into any other factor
- Remaining factor, projected on $\mathbf{Q}$, gives marginal over $\mathbf{Q}$

Factor Elimination



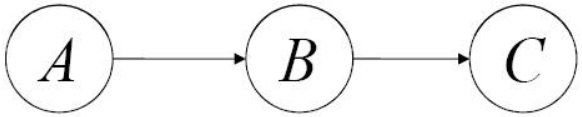
Compute $\Pr(C)$: eliminate Θ_A and $\Theta_{B|A}$

A	Θ_A
true	.6
false	.4

A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

Factor Elimination



A	Θ_A
true	.6
false	.4

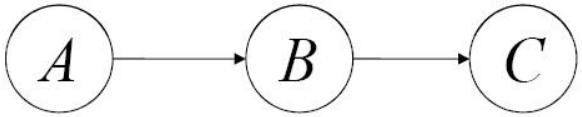
A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

Compute $\Pr(C)$: eliminate Θ_A and $\Theta_{B|A}$

Eliminate Θ_A: Sum out private variables (none), multiply into any other factor (let's choose $\Theta_{B|A}$)

Factor Elimination



A	Θ_A
true	.6
false	.4

A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

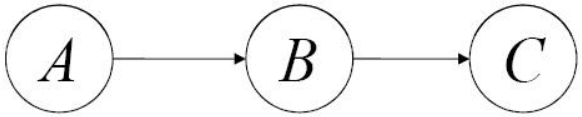
B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

Compute $\Pr(C)$: eliminate Θ_A and $\Theta_{B|A}$

Eliminate Θ_A: Sum out private variables (none), multiply into any other factor (let's choose $\Theta_{B|A}$)

A	B	$\Theta_A\Theta_{B\|A}$
true	true	.54
true	false	.06
false	true	.08
false	false	.32

Factor Elimination



A	Θ_A
true	.6
false	.4

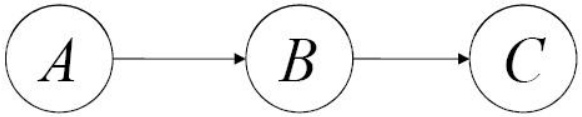
A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

Eliminate $\Theta_A\Theta_{B|A}$: Sum out private variables (A)

B	$\sum_A \Theta_A\Theta_{B\|A}$
true	$.62 = .54 + .08$
false	$.38 = .06 + .32$

Factor Elimination



A	Θ_A
true	.6
false	.4

A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

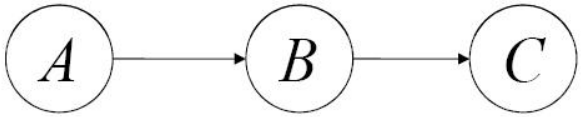
Eliminate $\Theta_A\Theta_{B|A}$: Sum out private variables (A)

B	$\sum_A \Theta_A\Theta_{B\|A}$
true	$.62 = .54 + .08$
false	$.38 = .06 + .32$

Multiply into $\Theta_{C|B}$

B	C	$\Theta_{C\|B} \sum_A \Theta_A\Theta_{B\|A}$
true	true	.186
true	false	.434
false	true	.190
false	false	.190

Factor Elimination



A	Θ_A
true	.6
false	.4

A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

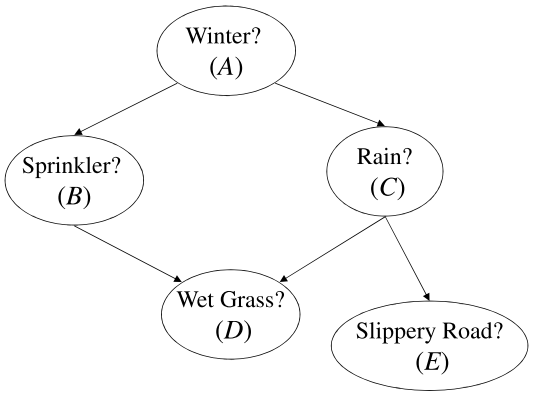
Sum out B

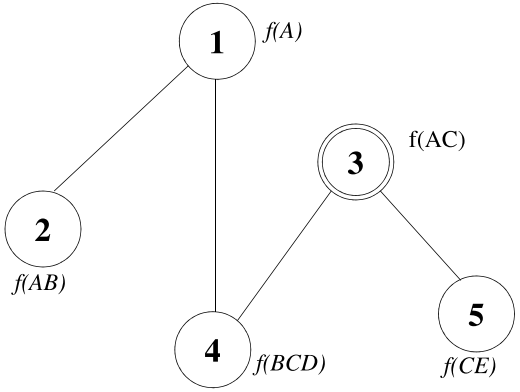
C	$\sum_B \Theta_{C\|B} \sum_A \Theta_A \Theta_{B\|A}$
true	.376
false	.624

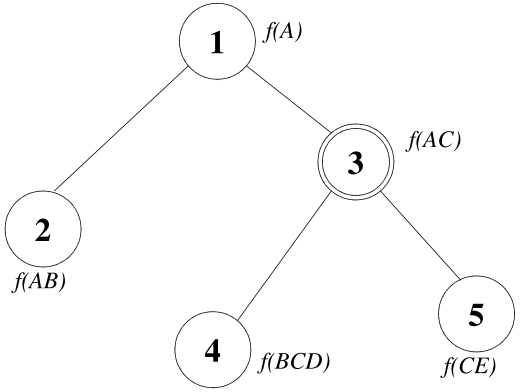
Elimination Trees

- Two choices per step: Which Φ to eliminate, which Φ to multiply into
- Specified by *elimination tree*

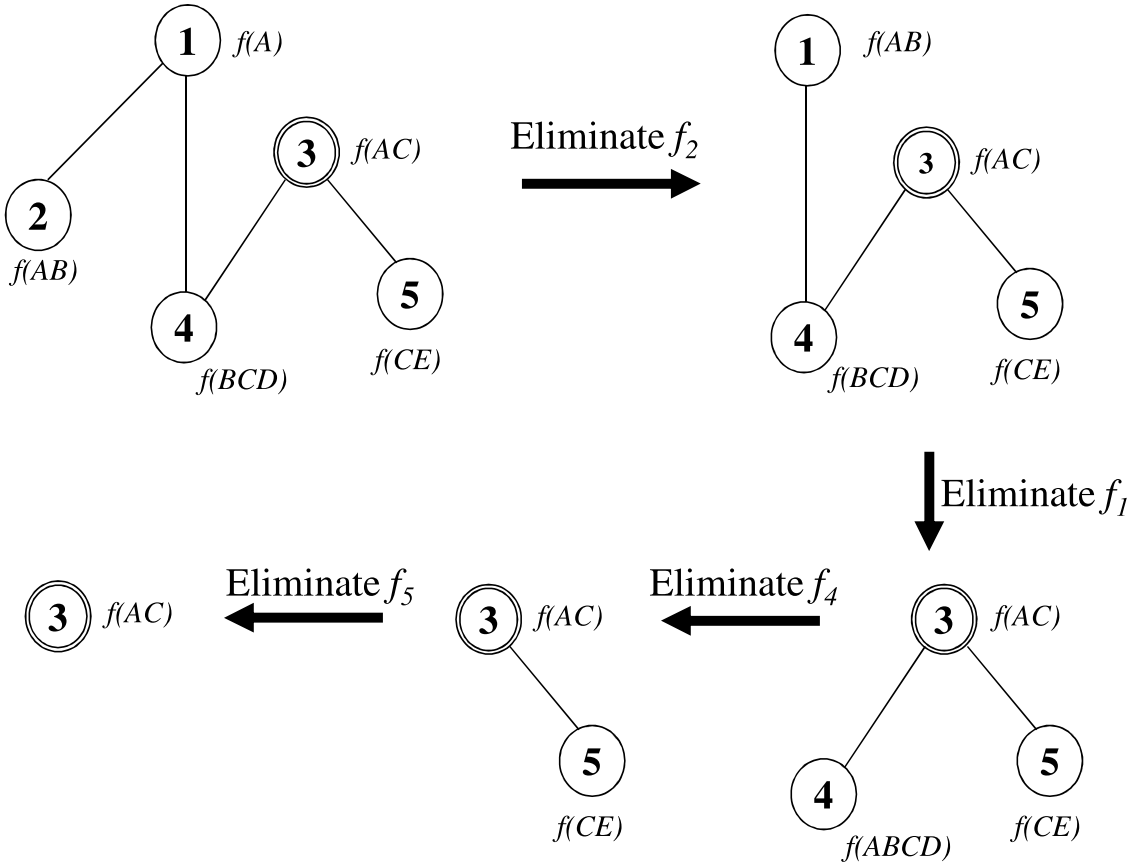
Elimination Trees





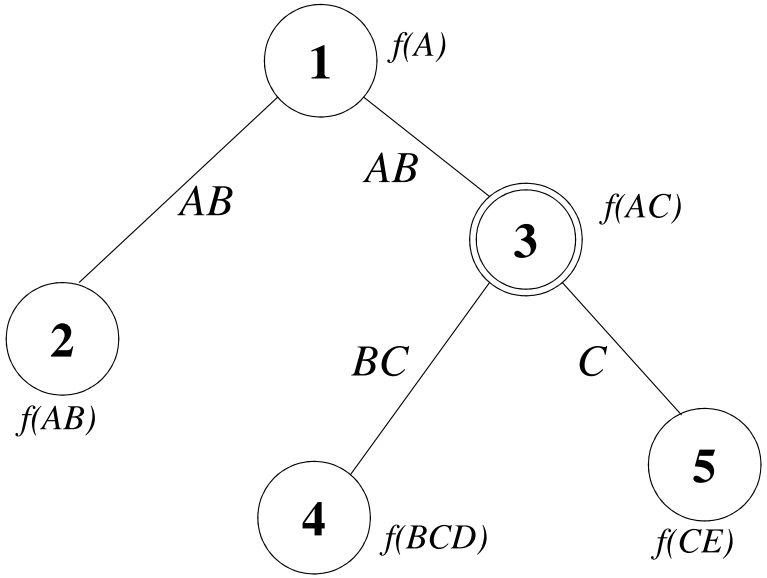


Elimination Trees



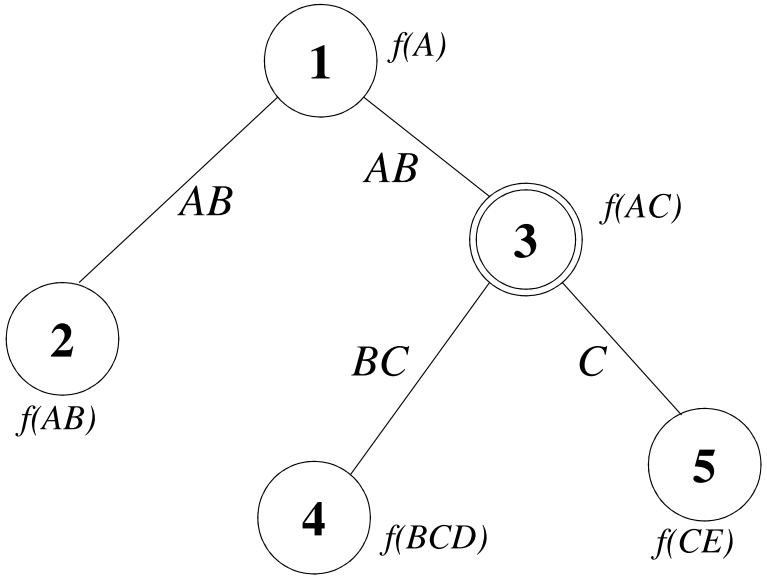
Separators

Intersection of variables on either side of edge



Separators

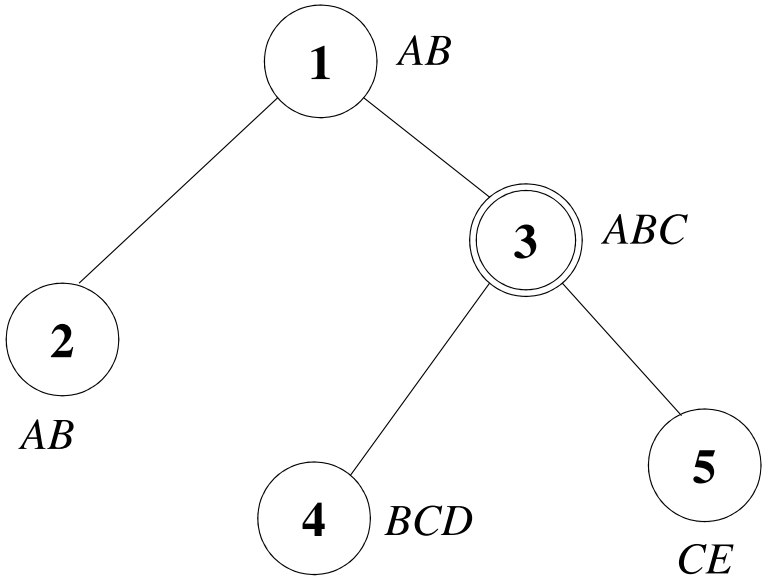
Intersection of variables on either side of edge



Between 2 steps of elimination (summing out & multiplication), factor contains precisely separator variables

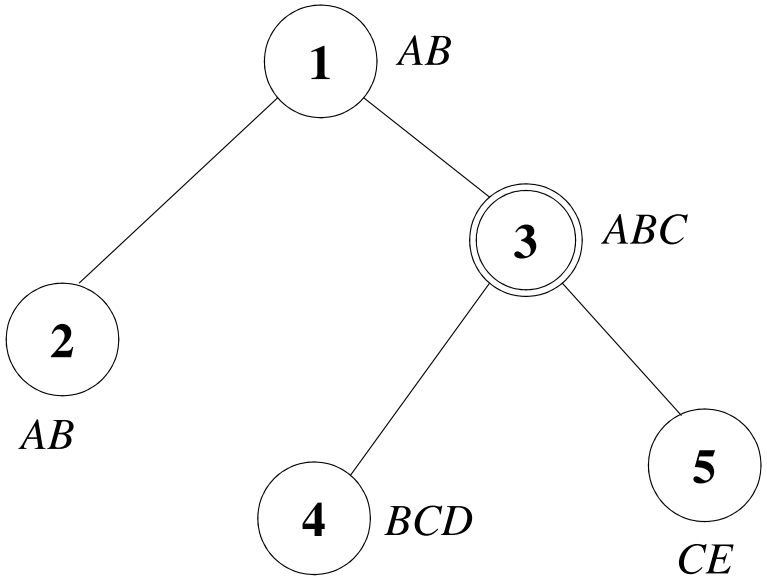
Clusters

Variables of node union all surrounding separators



Clusters

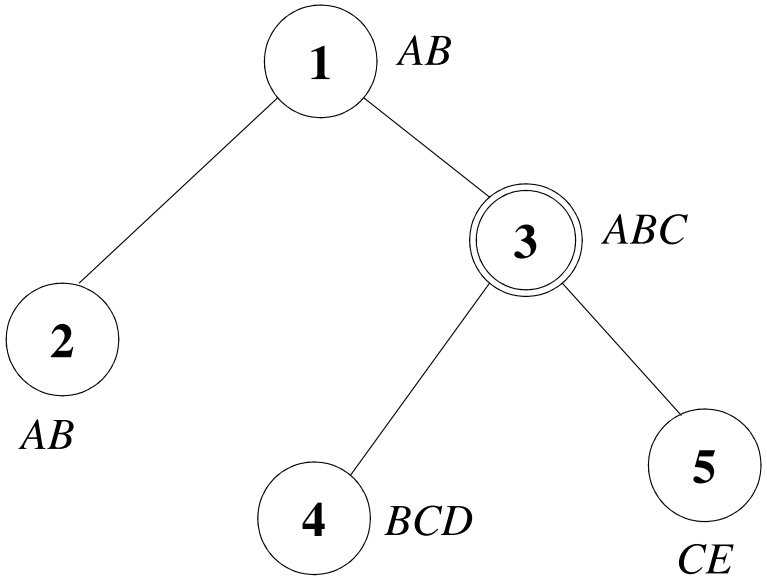
Variables of node union all surrounding separators



Before both steps of elimination (summing out & multiplication), factor contains precisely cluster variables

Clusters

Variables of node union all surrounding separators



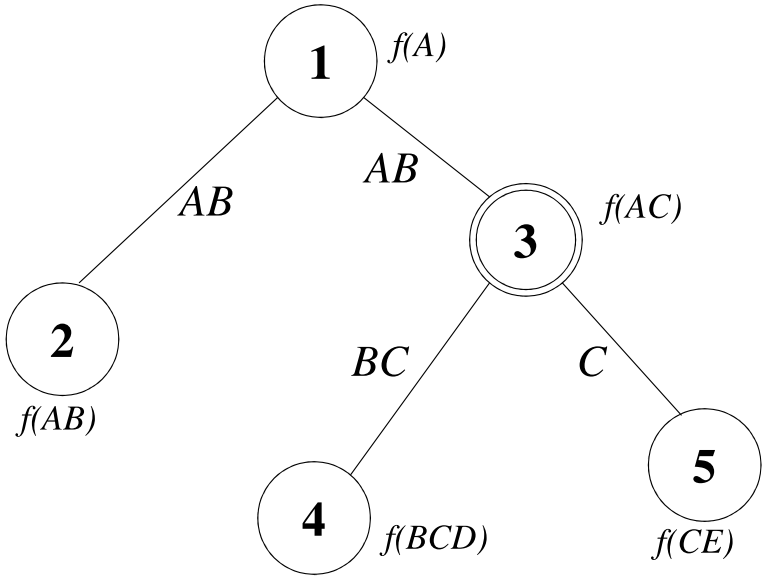
width
$= \max |cluster| - 1$

Given elimination order, can generate elimination tree of same width

Before both steps of elimination (summing out & multiplication), factor contains precisely cluster variables

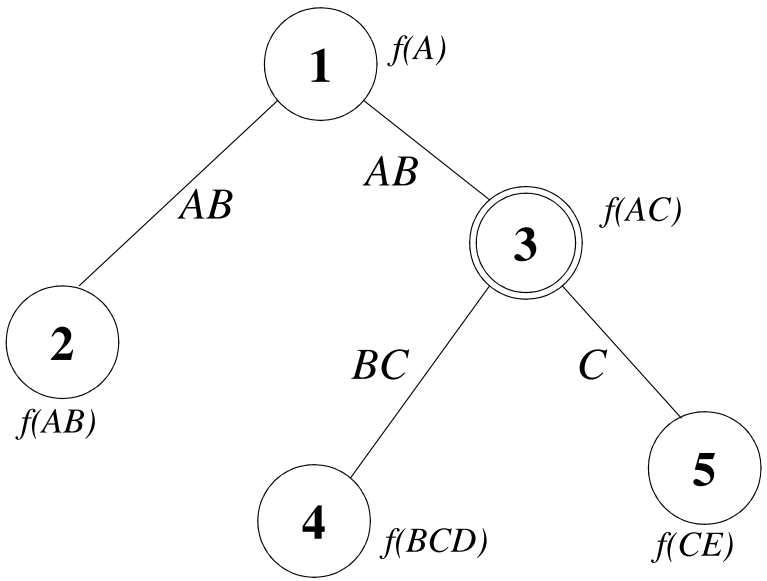
Factor Elimination as Message Passing

View eliminated factor as *message* to neighbor (to be multiplied)



Factor Elimination as Message Passing

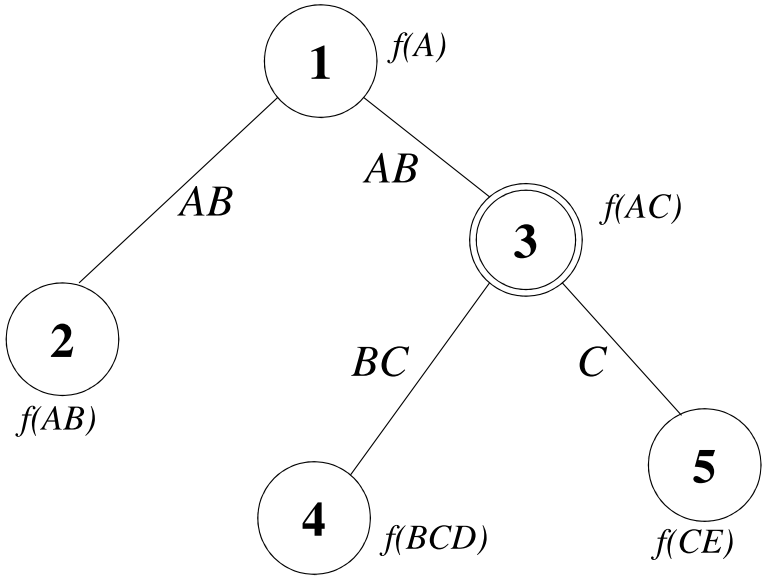
View eliminated factor as *message* to neighbor (to be multiplied)



After one message per edge, result obtained at root

Factor Elimination as Message Passing

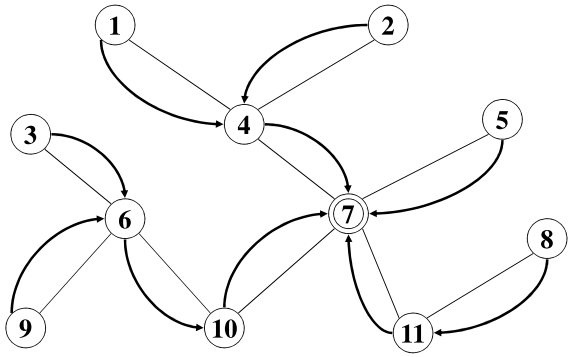
View eliminated factor as *message* to neighbor (to be multiplied)



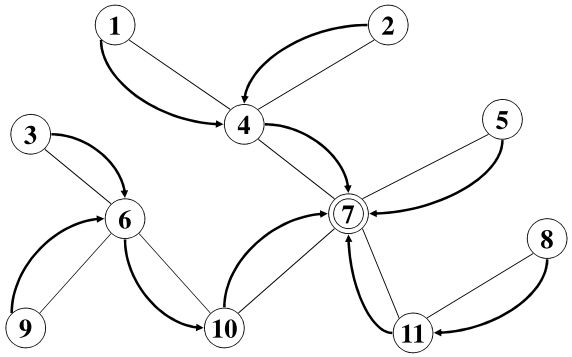
Size of any factor computed $\leq exp(width)$

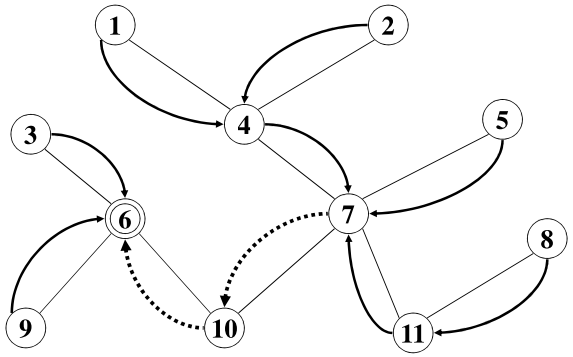
After one message per edge, result obtained at root

Multiple Queries and Message Reuse

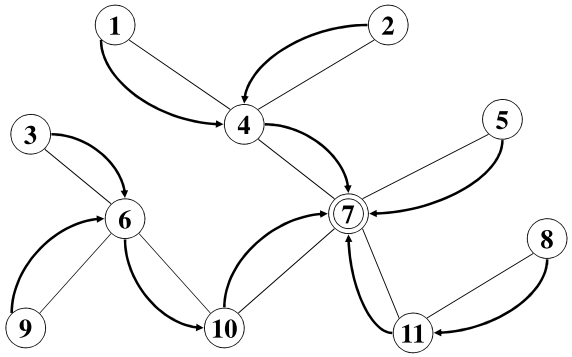


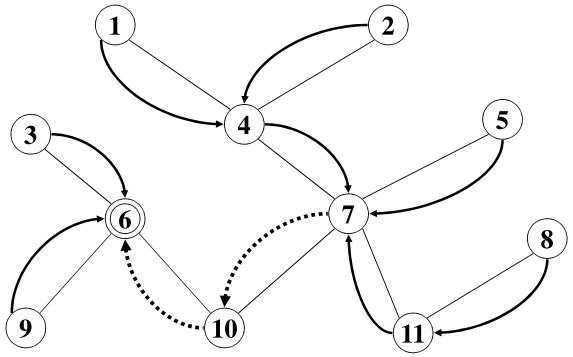
Multiple Queries and Message Reuse





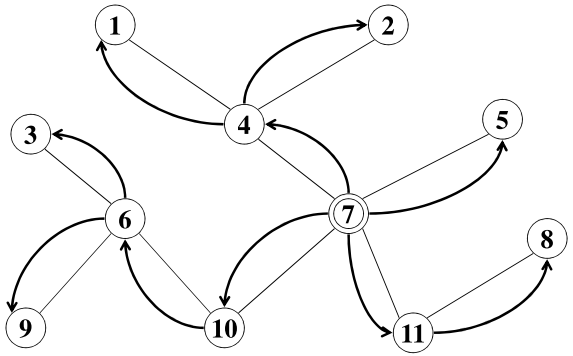
Multiple Queries and Message Reuse





Inward and outward passes

Two messages per edge answer all queries



Complexity of Message Passing

- Computing message involves $\prod$ and $\sum$
- Both $O(exp(w))$, $w = width$ bounds size of biggest factor
- Assumes each node has bounded # of neighbors
  - Given BN with n variables and treewidth w, $\exists$ elimination tree with $O(n)$ edges, width w, and bounded degree
- 2 messages/edge; total complexity $O(n\, exp(w))$
- Compared with $O(n^2\, exp(w))$ for variable elimination for all marginals

The Polytree Algorithm

- If Bayesian network has polytree structure, can use that as elimination tree (after dropping directionality)
- Width k = max # of parents of any node
- Linear complexity $O(n\,exp(k))$ for bounded k

The Jointree Connection

Jointree for DAG G is a tree where each node i is labeled with *cluster* $\mathbf{C}_i$ such that:

- Each $\mathbf{C}_i$ is a set of nodes of G
- Each family of G is contained in some $\mathbf{C}_i$
- $\mathbf{C}_i \cap \mathbf{C}_j$ is contained in every cluster on the path between nodes i and j (*jointree property, running intersection property*)

The Jointree Connection

Jointree for DAG G is a tree where each node i is labeled with *cluster* $\mathbf{C}_i$ such that:

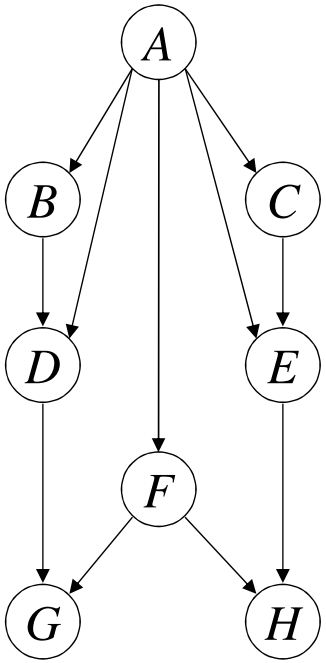
- Each $\mathbf{C}_i$ is a set of nodes of G
- Each family of G is contained in some $\mathbf{C}_i$
- $\mathbf{C}_i \cap \mathbf{C}_j$ is contained in every cluster on the path between nodes i and j (*jointree property, running intersection property*)

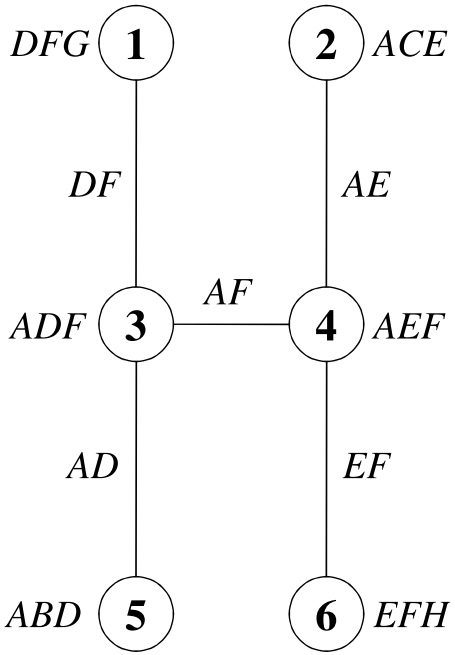
Minimal jointree: ceases to be jointree with any variable removed from any cluster

Width of jointree $\stackrel{def}{=} \max |\mathbf{C}_i| - 1$

For edge i–j, *separator* $\mathbf{S}_{ij} \stackrel{def}{=} \mathbf{C}_i \cap \mathbf{C}_j$

DAG and Its Jointree





Jointrees and Elimination Trees

- Theorem: Clusters of elimination tree satisfy the three properties of jointree
  - Every elimination tree induces a jointree of same width

Jointrees and Elimination Trees

- Theorem: Clusters of elimination tree satisfy the three properties of jointree
  - Every elimination tree induces a jointree of same width
- Definition: Elimination tree $(\mathcal{T}, \phi)$ is *embedded* in jointree $(\mathcal{T}, \mathbf{C})$ if $vars(\phi_i) \subseteq \mathbf{C}_i$
  - To construct embedded elimination tree, adopt $\mathcal{T}$ of jointree, assign ϕ to node i such that $vars(\phi) \subseteq \mathbf{C}_i$

Jointrees and Elimination Trees

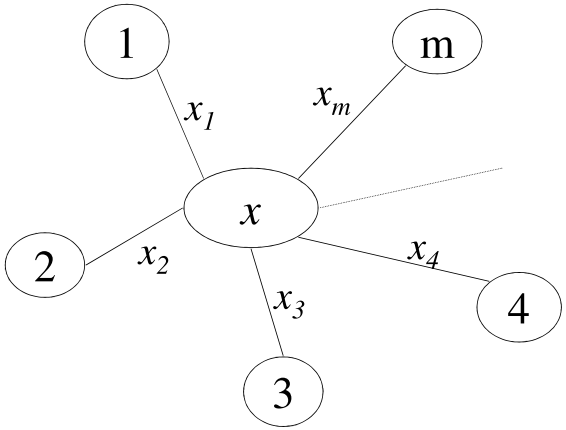
- Theorem: Clusters of elimination tree satisfy the three properties of jointree
  - Every elimination tree induces a jointree of same width
- Definition: Elimination tree $(\mathcal{T}, \phi)$ is *embedded* in jointree $(\mathcal{T}, \mathbf{C})$ if $vars(\phi_i) \subseteq \mathbf{C}_i$
  - To construct embedded elimination tree, adopt $\mathcal{T}$ of jointree, assign ϕ to node i such that $vars(\phi) \subseteq \mathbf{C}_i$
- Theorem: $\mathbf{S}_{ij}$ and $\mathbf{C}_i$ of embedded elimination tree $\subseteq$ jointree counterparts
  - Every jointree leads to an elimination tree of no greater width

The Jointree Algorithm

- Construct jointree for given Bayesian network
- Assign each CPT $\Theta_{X|\mathbf{U}}$ to a cluster that contains X and $\mathbf{U}$ (guaranteed by properties of jointree)
- Two methods for message passing
  - Shenoy-Shafer: same as factor elimination
  - Hugin: uses division to compute messages

Shenoy-Shafer

- Inward messages toward root, outward messages from root, totaling 2 messages/edge
- Node x sends message to m after receiving messages $x_1, \ldots, x_{m-1}$ from all other neighbors

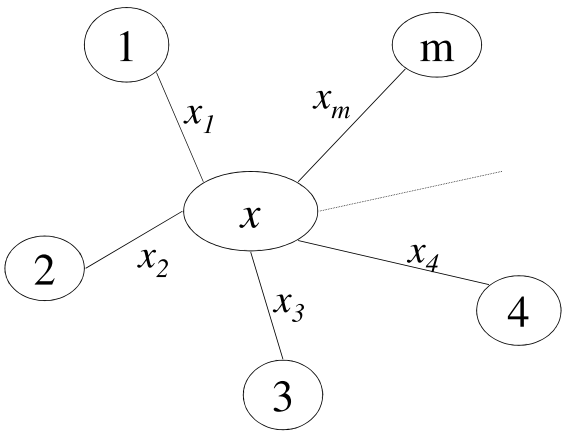


x_m = product of $x_1, \ldots, x_{m-1}$ and ϕ_x, projected on S_{xm}

Shenoy-Shafer

- Time complexity $O(n\,exp(w))$ as factor elimination, using appropriate jointree (with bounded degree)
- Stored messages (for reuse in outward pass) over separator variables; space complexity exponential only in separator size ($\leq w$)

Hugin



x_m = product of ϕ_x and $x_1, \ldots, x_m$, projected on S_{xm}, *divided* by x_m

Inward pass same as Shenoy-Shafer
Outward pass

- Root already has $\prod$ of ϕ_x and $x_1, \ldots, x_m$; computes each message by projection and division
- Other node already has $\prod$ of ϕ_x and messages from all but one neighbor; last one received from direction of root

Hugin

- Factor in cluster gives $\Pr(\mathbf{C}_i, \mathbf{e})$ as Shenoy-Shafer
- In addition, factor on edge (saved just before division is performed to compute message) gives $\Pr(\mathbf{S}_{ij}, \mathbf{e})$
- Space complexity $O(n\, exp(w))$, higher than Shenoy-Shafer
- Time complexity $O(n\, exp(w))$, does not require bounded degree

Inference by Factor Elimination: Summary

- Marginals by factor elimination
- Elimination trees, separators, clusters, width
- Factor elimination as message passing
- Connection to inward and outward message passing on jointree
- Two flavors of jointree algorithm: Shenoy-Shafer and Hugin

Conditioning

- Also known as *case analysis*

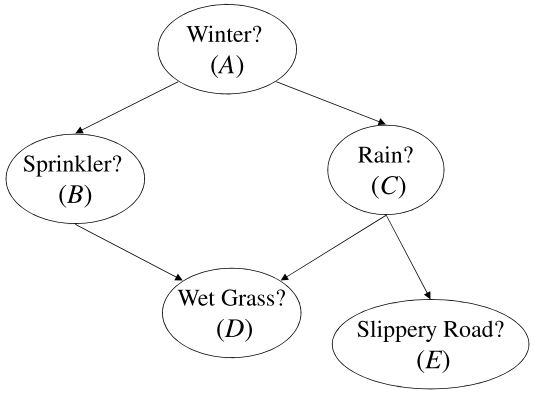
$$\Pr(\alpha) = \sum_{i=1}^{n} \Pr(\alpha \wedge \beta_i)$$

- Making assumption β_i can simplify problem
- Simplifying problem in different ways: *cutset conditioning* and *recursive conditioning*

Review of Network Pruning

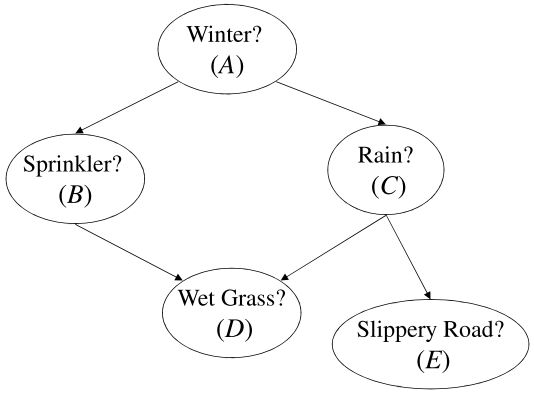
- Consider computation of $\Pr(\mathbf{Q}, \mathbf{e})$ (includes prior marginals and probability of evidence as special cases)
- Pruning nodes: All leaves $\notin \mathbf{Q} \cup \mathbf{E}$, interatively
  - Worst case: All leaves $\in \mathbf{Q} \cup \mathbf{E}$, no pruning
  - Best case: All $\mathbf{Q} \cup \mathbf{E}$ are roots, every node $\notin \mathbf{Q} \cup \mathbf{E}$ pruned
- Pruning edges: For each edge $U \rightarrow X$, $U \in \mathbf{E}$
  - Remove edge, shrink CPT $\Theta_{X|\mathbf{U}}$ by removing rows inconsistent with $\mathbf{e}$ and removing column U
- *Effective treewidth* is treewidth of pruned network given query

Cutset Conditioning



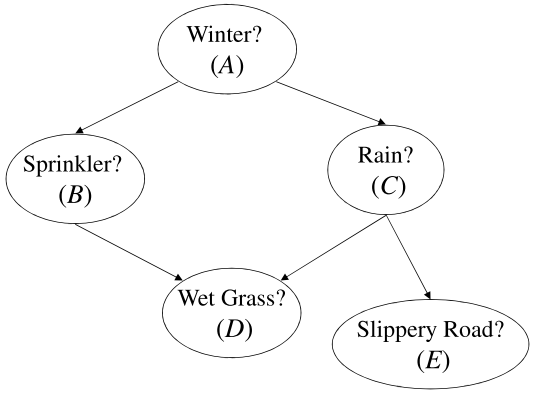
- Try to reduce network to polytree
- $\Pr(E, D = \mathit{true})$?

Cutset Conditioning



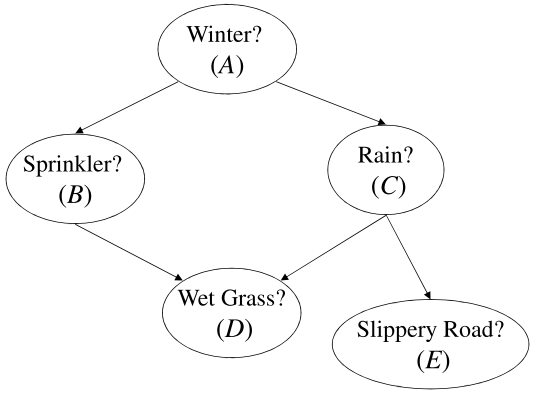
- Try to reduce network to polytree
- $\Pr(E, D = \mathit{true})$? No pruning available

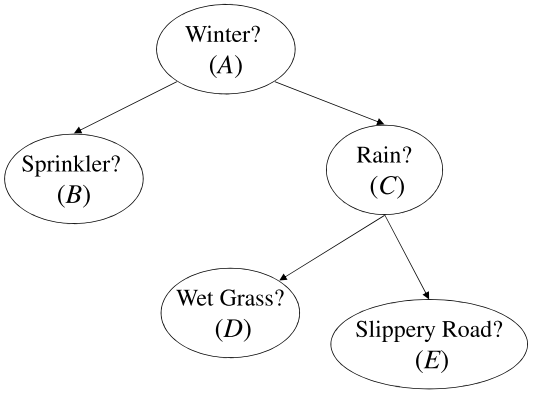
Cutset Conditioning



- Try to reduce network to polytree
- $\Pr(E, D = \mathit{true})$? No pruning available
- How about $\Pr(E, D = \mathit{true}, B = \mathit{true})$ $+ \Pr(E, D = \mathit{true}, B = \mathit{false})$

Cutset Conditioning

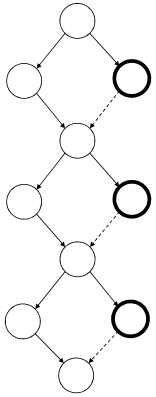


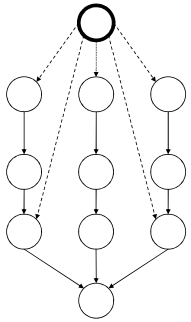


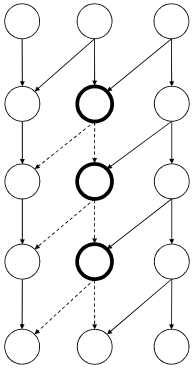
- Try to reduce network to polytree
- $\Pr(E, D = \mathit{true})$? No pruning available
- How about $\Pr(E, D = \mathit{true}, B = \mathit{true})$ $+ \Pr(E, D = \mathit{true}, B = \mathit{false})$

Loop-Cutsets

Nodes **C** is a *loop-cutset* if removing edges outgoing from **C** results in polytree





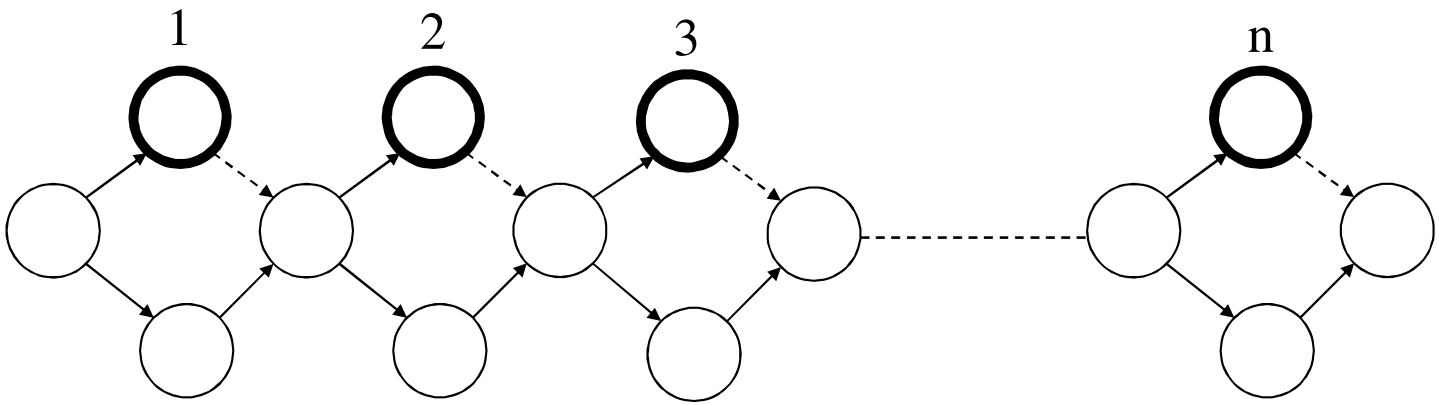


Cutset Conditioning

- Case analysis over loop-cutset
- Time complexity ($s = |\text{loop-cutset}|$)
  $O(n\,exp(k))O(exp(s)) = O(n\,exp(k+s))$
- Finding smallest loop-cutset is NP-hard
- Space complexity $O(n\,exp(k))$; compares favorably with $O(n\,exp(w))$ for elimination

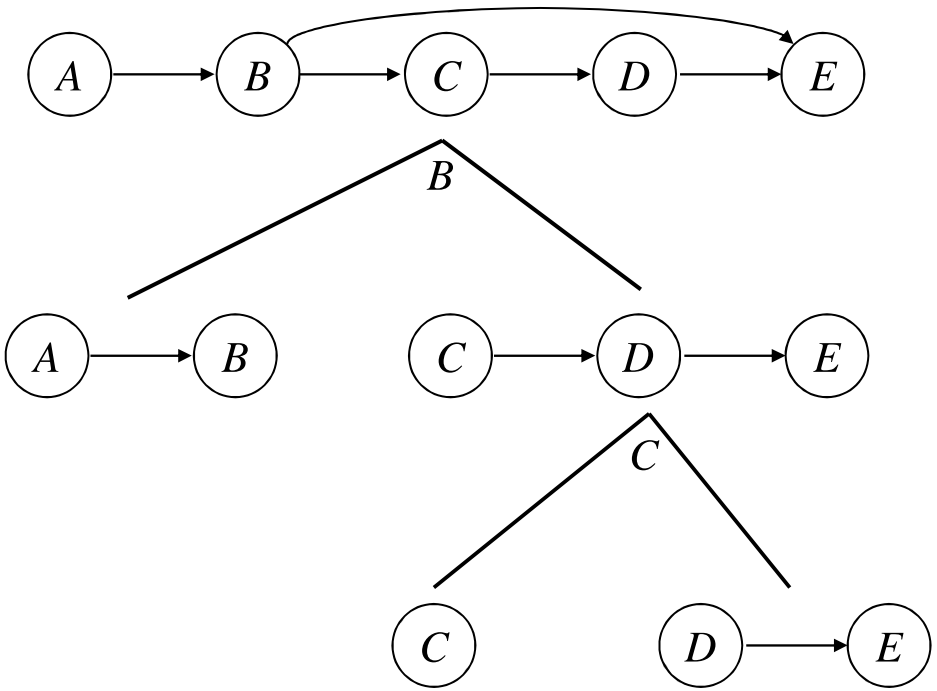
Cutset Conditioning

Inefficient when loop-cutset is large



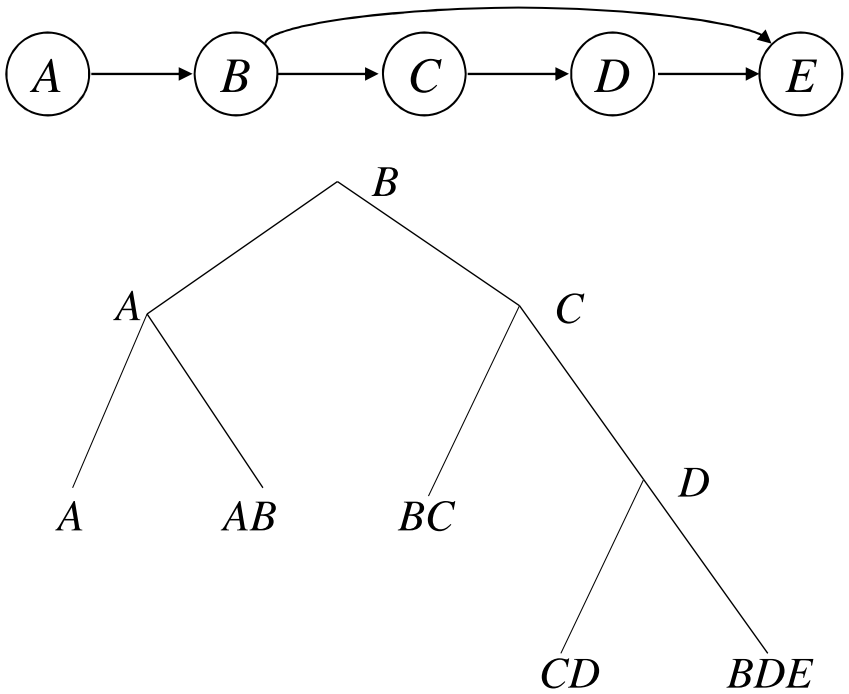
Recursive Conditioning

Different way to simplify problem: divide-and-conquer



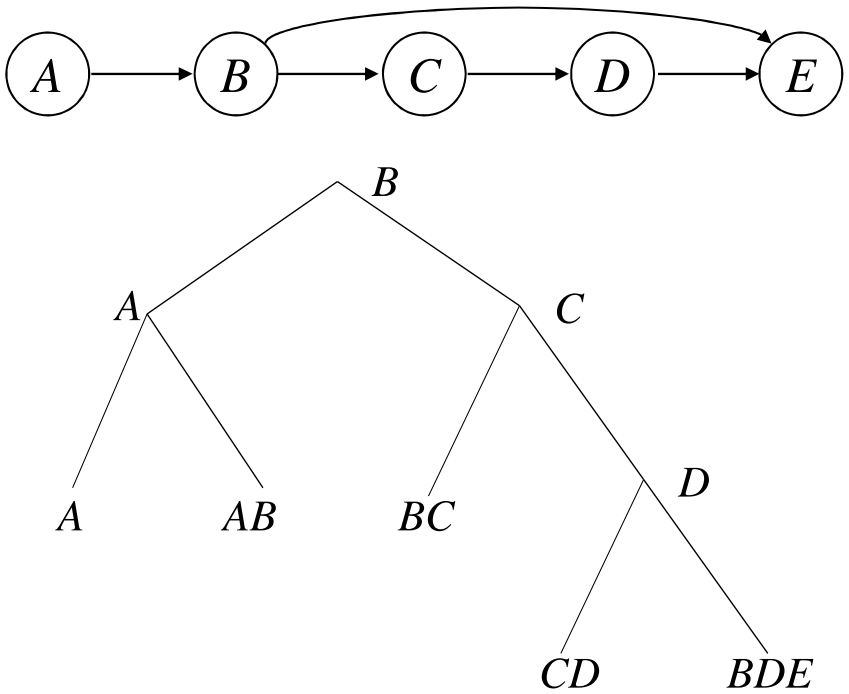
Dtrees

Full binary tree whose leaves correspond to network CPTs



Dtrees

Cutset: $\cap$ of variables on either side minus *acutset* ($\cup$ of ancestor cutsets)



Recursive Conditioning: Basic Algorithm

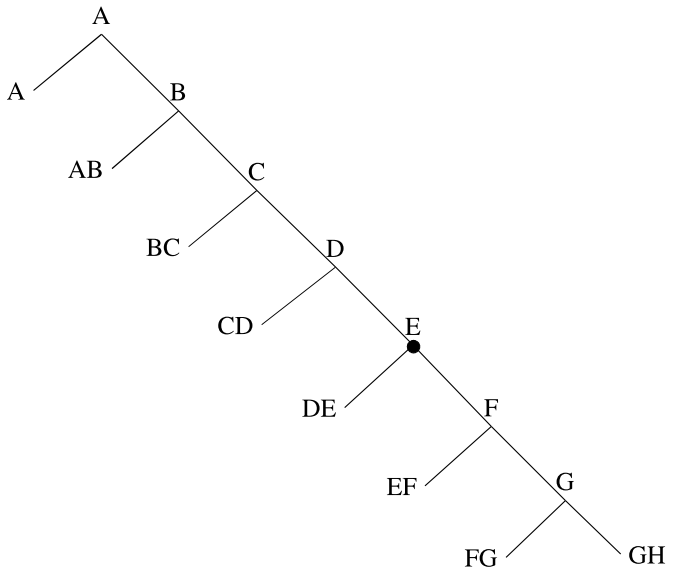
- Case analysis on root cutset, considering subset of $exp(|cutset|)$ cases compatible with evidence
- Each case returns $\prod$ of results from two recursive calls on left and right child
- Return $\sum$ of results from all cases

Complexity

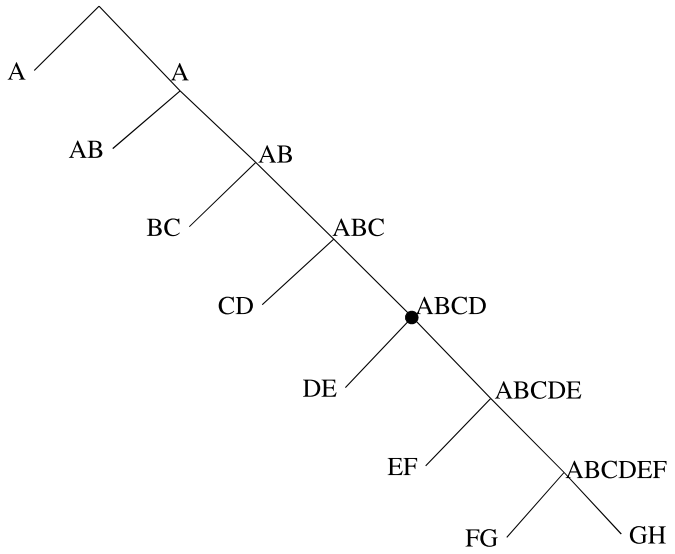
- # of recursive calls to non-root node $\leq$ acutset$^{\#}$
- Time $O(n\, exp(w \log n))$, space $O(wn)$, where w is *cutset width*, assuming *balanced* dtrees
  - Can turn elimination order of width w into dtree with cutset width $\leq w + 1$
  - Can balance dtree while keeping cutset width $\leq w + 1$
- Potentially more time, but less space, than variable elimination
- Not comparable to cutset conditioning

Redundant Computation

A→B→C→D→E→F→G→H



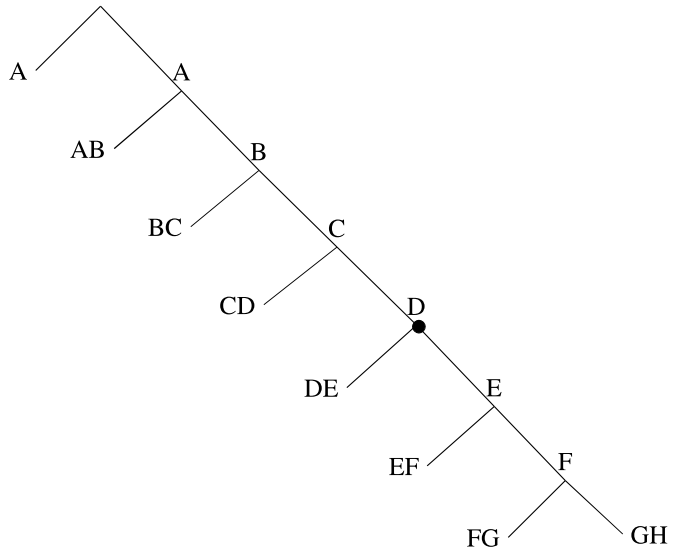
cutsets



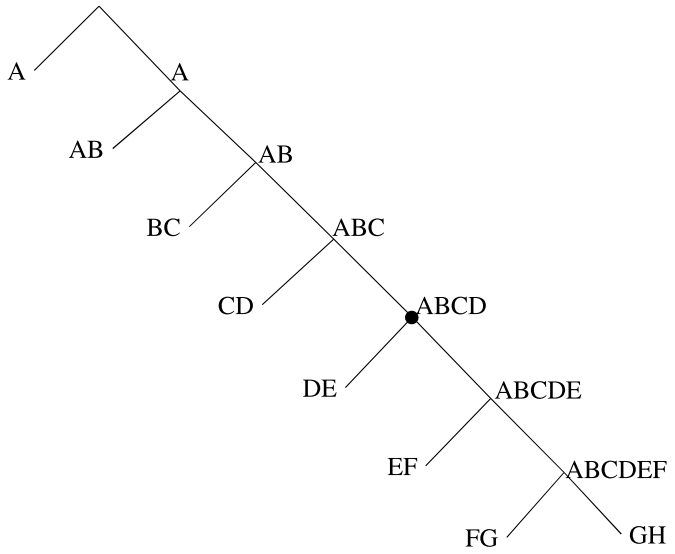
a-cutsets

Improving Time Complexity: Caching

Context: $\cap$ of variables and acutset



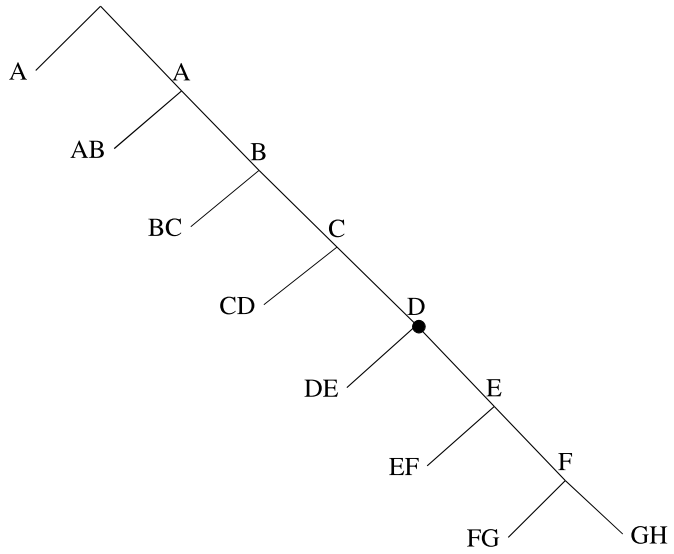
contexts



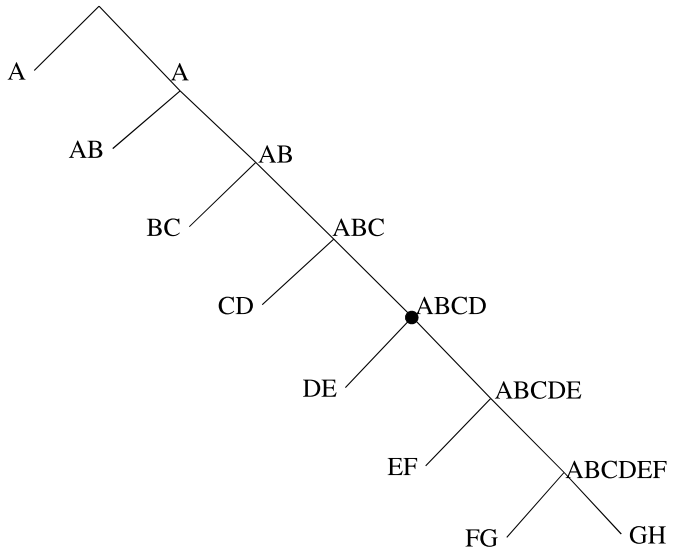
a-cutsets

Improving Time Complexity: Caching

Cache result for each instantiation of context



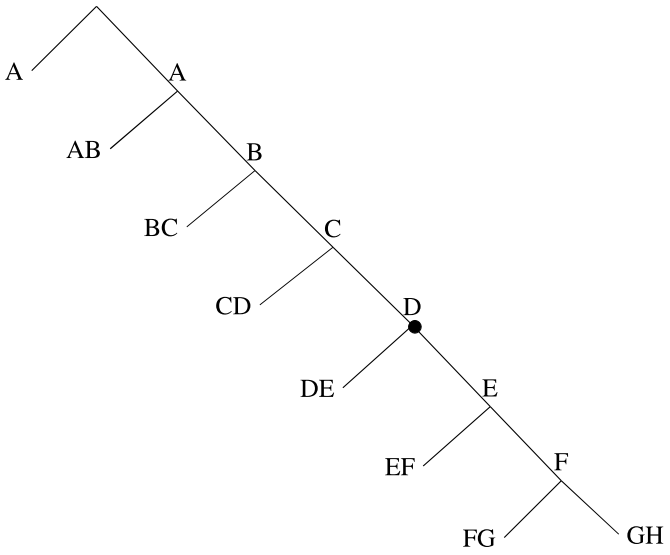
contexts



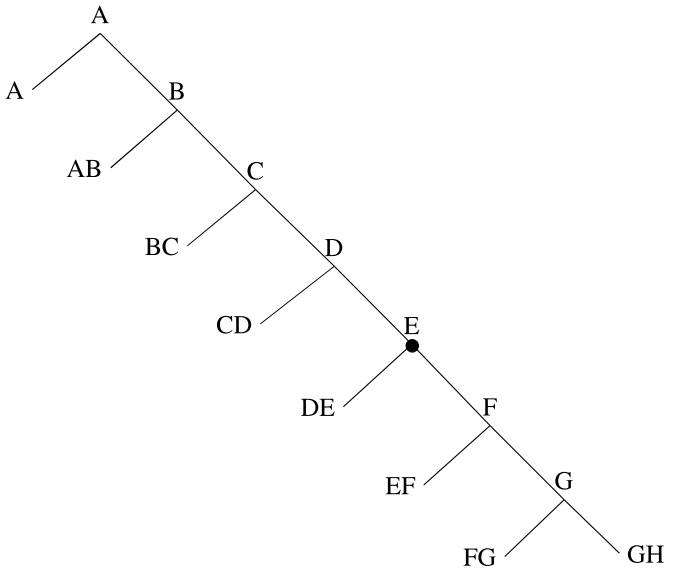
a-cutsets

Improving Time Complexity: Caching

$\#$ of recursive calls to node T
$\leq$ cutset$(T^p)^{\#}$ context$(T^p)^{\#}$



contexts



cutsets

Full Caching: Complexity

- $Cluster \stackrel{def}{=}$ variables, for leaf
- $Cluster \stackrel{def}{=}$ cutset $\cup$ context, for internal node
  - Cutset and context always disjoint
- Width of dtree is $\max |cluster| - 1$
- Time and space complexity $O(n\, exp(w))$, same as variable elimination
  - Elimination order of width w can be turned into dtree of no greater width
  - Space complexity actually exponential only in *context width*

Any-space Inference

Two extremes of recursive conditioning

- No caching: $O(n\, exp(w\, log\, n))$ time, $O(wn)$ space
- Full caching: $O(n\, exp(w))$ time and space

Can be anywhere in between

Cache factor, $0 \leq cf(T) \leq 1$, is fraction of cache entries that will be filled

$cf(T) = 0$ for all nodes: no caching
$cf(T) = 1$ for all nodes: full caching

Any-space Inference: Complexity

Average # of recursive calls to node T, $\text{ave}(T) \leq$

$$\text{cutset}(T^p)^{\#}[cf(T^p)\text{context}(T^p)^{\#} + (1 - cf(T^p))\text{ave}(T^p)]$$

Exact for *discrete* cache factors (0 or 1)

No caching

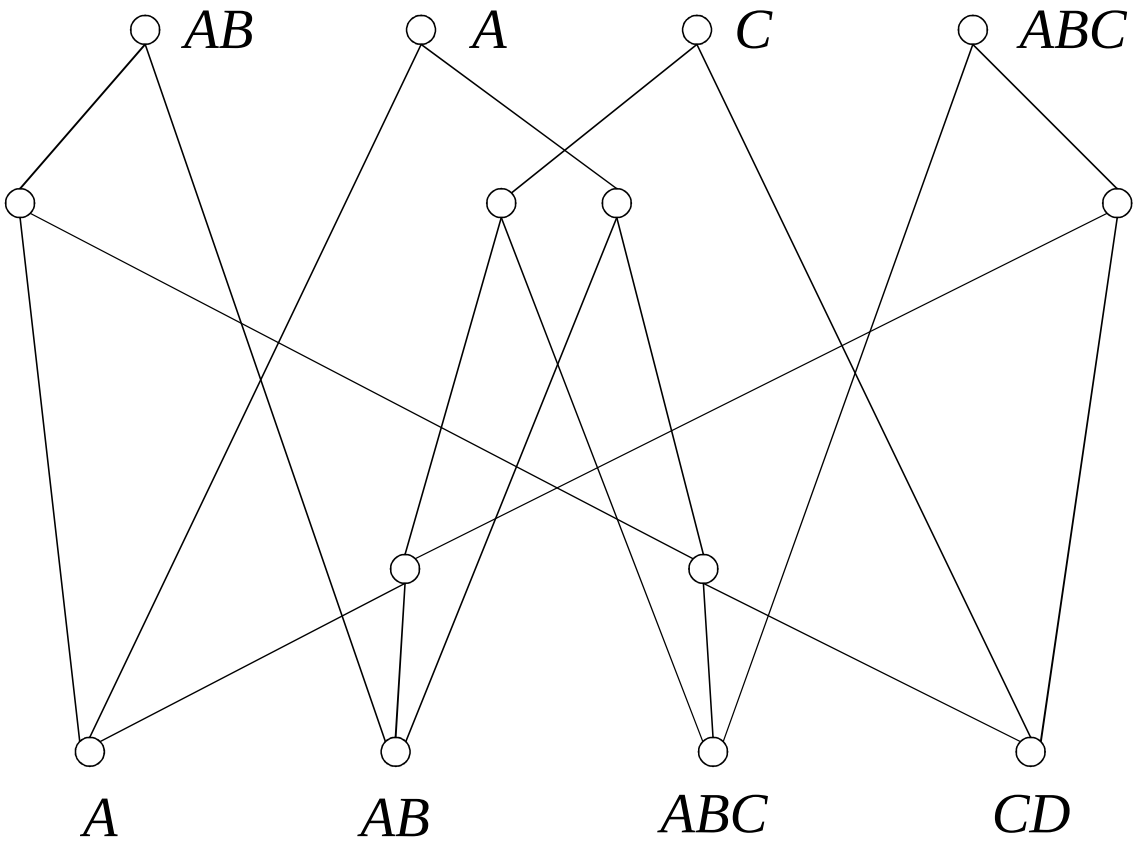
- $\text{ave}(T) \leq \text{cutset}(T^p)^{\#}\text{ave}(T^p)$
- Solution: $\text{ave}(T) \leq \text{acutset}(T)^{\#}$

Full caching

- $\text{ave}(T) \leq \text{cutset}(T^p)^{\#}\text{context}(T^p)^{\#}$

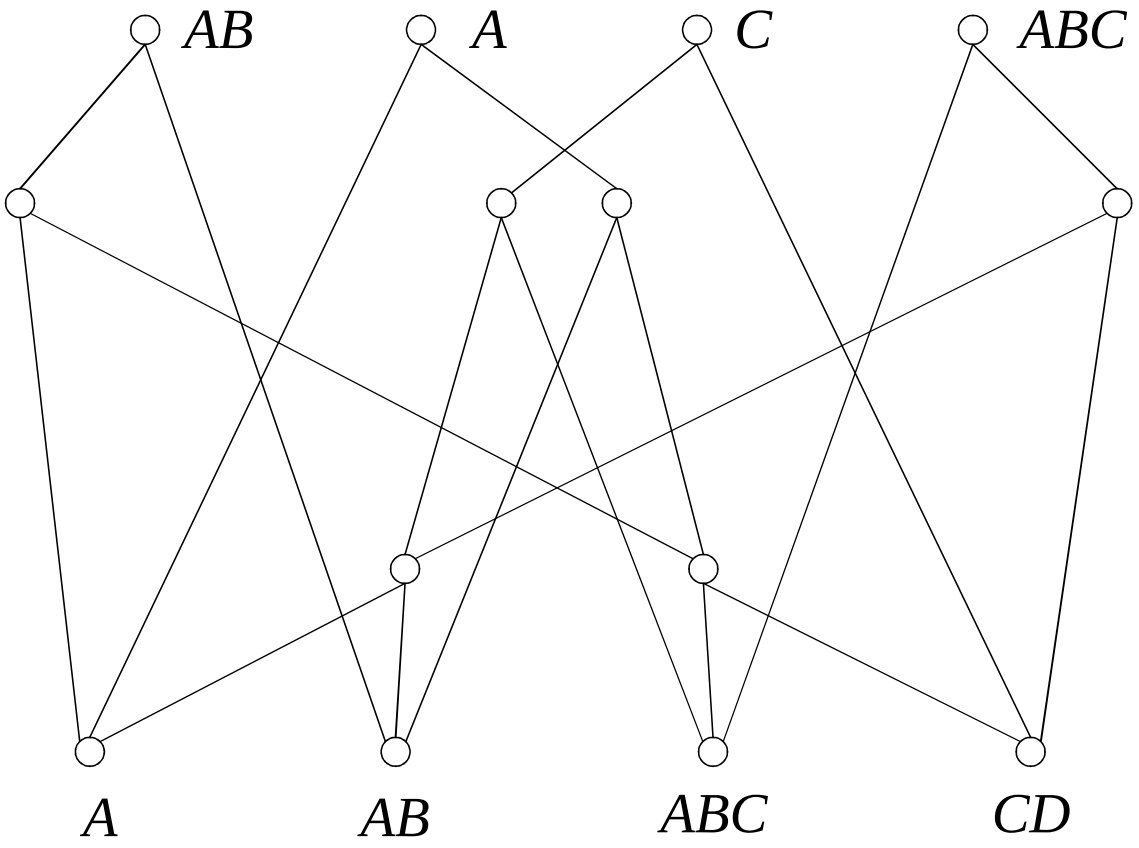
Decomposition Graphs

A *dgraph* is a set of dtrees that share structure



Decomposition Graphs

RC on dgraph amounts to RC on every dtree



Decomposition Graphs: Multiple Marginals

- RC on dgraph more efficient than on separate dtrees, due to structure sharing
- Besides $\Pr(\mathbf{e})$, RC computes $\Pr(\mathbf{C}, \mathbf{e})$ for root cutset $\mathbf{C}$
- Hence RC on dgraph computes multiple marginals, similar to jointree

Decomposition Graphs: Construction

- Given dtree, can construct dgraph that allows computation of all family marginals
  - Each dtree leaf (network family) becomes child of one new dgraph root
- For internal node X, will have root cutset $X\mathbf{U}$
- For leaf X, will have root cutset $\mathbf{U}$
  - $\Pr(X\mathbf{U}, \mathbf{e}) = \Pr(\mathbf{U}, \mathbf{e}) \Pr(X|\mathbf{U}, \mathbf{e})$
  - $\Pr(X|\mathbf{U}, \mathbf{e}) = \Pr(X|\mathbf{U})$

Decomposition Graphs: Complexity

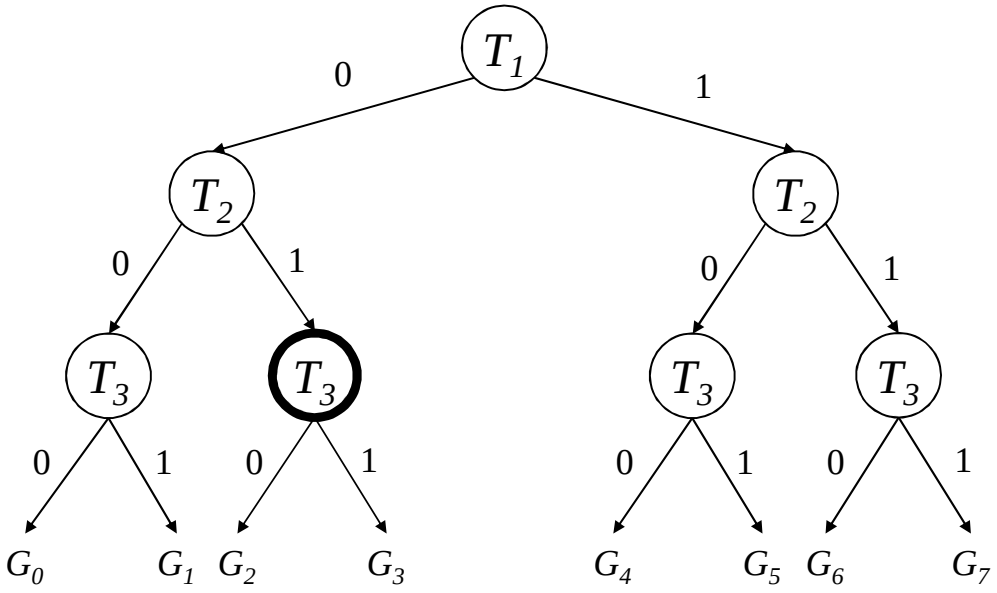
- Given dtree with $n \geq 5$ and width w, can construct such a dgraph with $(5n - 7)/2$ nodes and $4(n - 2)$ edges, and width w
- RC with full caching on dgraph runs in $O(n\, exp(w))$ time and space, same as jointree

Cache Allocation

- No enough memory for full caching, what caches to keep?
- Caches aren't equally useful
  - In particular, *dead cache* if $\text{context}(T) = \text{cluster}(T^p)$
- Consider discrete cache factors only
- Search for cache factors that minimize running time, computed by summing # of recursive calls to each node

Cache Allocation: Systematic Search

Depth-first search for assignment of *cf* 0/1 to each node in dgraph



Cache Allocation: Systematic Search

Prune with lower bound computed by assigning 1 to all undecided nodes

Branch on all parents before their child

- Lower bound monotonically improves
- Lower bound computed in constant time

Variable ordering: choose node requiring largest cache (context$^{\#}$)

Value ordering: try $cf = 1$ before $cf = 0$

Cache Allocation: Greedy Search

- Score for each node: reduced # of recursive calls (to any node) per cache entry
- Greedily assign $cf = 1$ to node with highest score
- Update scores after each assignment
  - $O(n^2)$ if done naively
  - Can be done in $O(n)$ by lazy updates
- Total complexity $O(n^2)$

Inference by Conditioning: Summary

- Simplify problem by case analysis
- Loop-cutsets and cutset conditioning: reduce to polytree
- Dtrees and recursive conditioning: divide-and-conquer
- Caching and time-space tradeoff
- Multiple marginals using dgraphs
- Systematic and greedy search for caching allocation