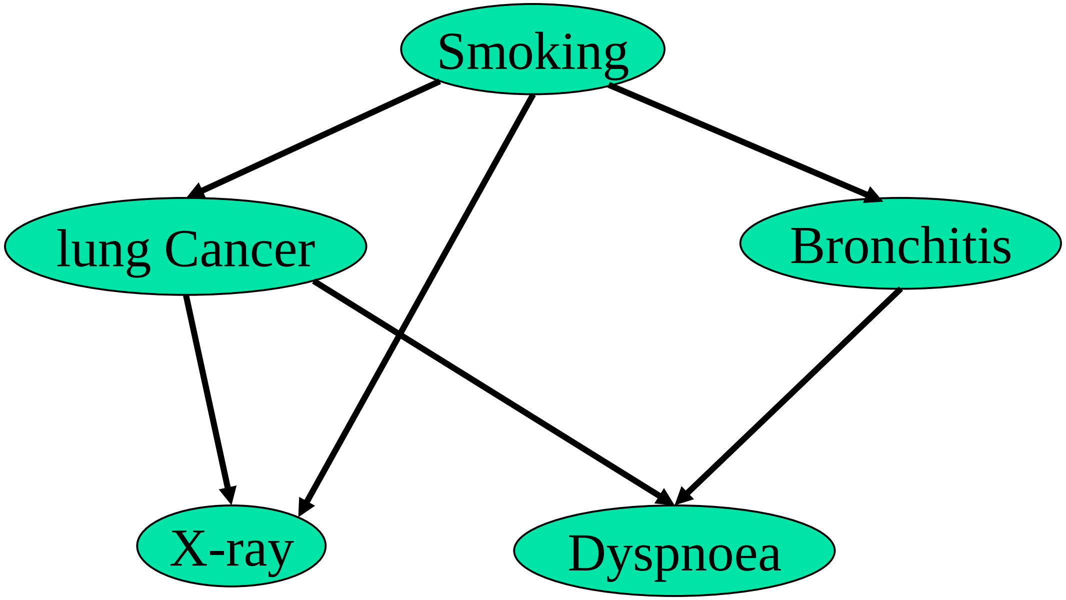
Exact Inference Algorithms Bucket-elimination

COMPSCI 276, Spring 2017

Class 5: Rina Dechter

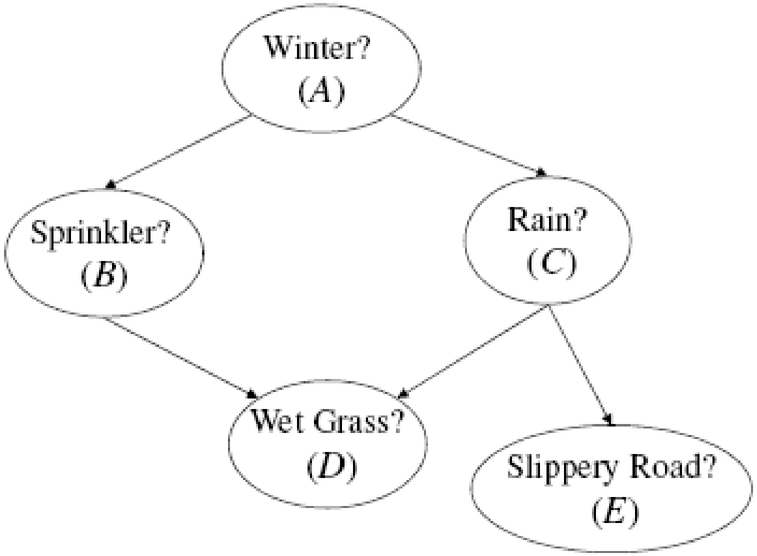
(Reading: **Dechter chapters 4** , Darwiche chapter 6, **Dechter Section 3.4**)

Belief Updating



P (lung cancer=yes | smoking=no, dyspnoea=yes) = ?

A Bayesian Network



A	Θ_A
true	.6
false	.4

A	B	$\Theta_{B\|A}$
true	true	.2
true	false	.8
false	true	.75
false	false	.25

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

B	C	D	$\Theta_{D\|BC}$
true	true	true	.95
true	true	false	.05
true	false	true	.9
true	false	false	.1
false	true	true	.8
false	true	false	.2
false	false	true	0
false	false	false	1

C	E	$\Theta_{E\|C}$
true	true	.7
true	false	.3
false	true	0
false	false	1

Queries

1. **Posterior marginals, or belief updating.** For every X_i not in E the belief is defined by $bel(X_i) = P_{\mathcal{B}}(X_i|e)$.

$$P(X_i|e) = \sum_{\mathbf{X}-X_i} \prod_j P(X_j|X_{pa_j}, e)$$

2. The **probability of evidence** is $P_{\mathcal{B}}(E = e)$. Formally,

$$P_{\mathcal{B}}(E = e) = \sum_{\mathbf{X}} \prod_j P(X_j|X_{pa_j}, e)$$

3. The **most probable explanation** (mpe) is an assignment $\mathbf{x}^o = (x^o{}_1, ..., x^o{}_n)$ satisfying

$$\mathbf{x}^o = argmax_{\mathbf{X}} \mathbf{P}_{\mathcal{B}} = argmax_{\mathbf{X}} \prod_j P(X_j|X_{pa_j}, e).$$

The mpe value is $\mathbf{P}_{\mathcal{B}}(\mathbf{x}^o)$, sometime also called MAP.

4. **Maximum a posteriori hypothesis (marginal** map**).** Given a set of hypothesized variables $\mathbf{A} = \{A_1, ..., A_k\}$, $\mathbf{A} \subseteq \mathbf{X}$, the map task is to find an assignment $\mathbf{a}^o = (a^o{}_1, ..., a^o{}_k)$ such that

$$\mathbf{a}^o = argmax_{\mathbf{A}} \sum_{\mathbf{X}-\mathbf{A}} \mathbf{P}(\mathbf{X}|\mathbf{e}) = argmax_{\mathbf{A}} \sum_{\mathbf{X}-\mathbf{A}} \prod_j P(X_j|X_{pa_j}, e)$$

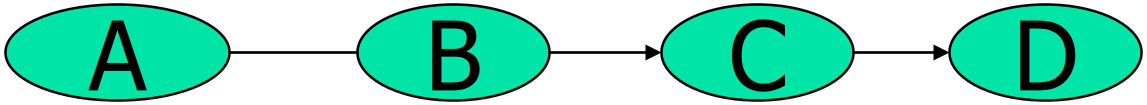
Belief Updating is NP-hard

- Each SAT formula can be mapped into a belief updating query in a Bayesian network

- Example $(\neg u \vee \neg w \vee y) \wedge (u \vee \neg v \vee w)$

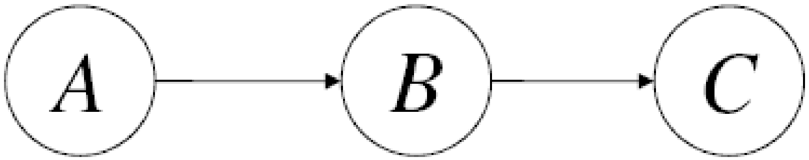
A Simple Network

Given:



- How can we compute P(D)?, P(D|A=0)? P(A|D=0)?
- Brute force O(k^4)
- Maybe O($4k^2$)

Elimination as a Basis for Inference



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

To compute the prior marginal on variable C, $\Pr(C)$

we first eliminate variable A and then variable B

Elimination as a Basis for Inference

- There are two factors that mention variable A, Θ_A and $\Theta_{B|A}$
- We multiply these factors first and then sum out variable A from the resulting factor.
- Multiplying 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

- Summing out variable A:

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

Elimination as a Basis for Inference

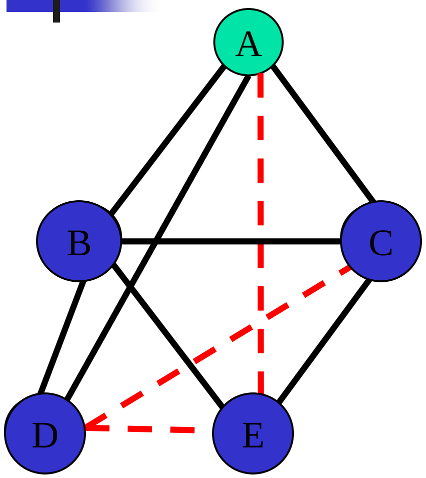
- We now have two factors, $\sum_A \Theta_A \Theta_{B|A}$ and $\Theta_{C|B}$, and we want to eliminate variable B
- Since B appears in both factors, we must multiply them first and then sum out B from the result.
- Multiplying:

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

- Summing out:

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

Belief Updating: P(X|evidence)=?



“Moral” graph

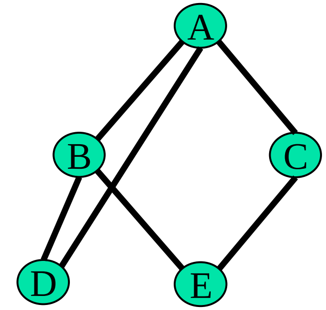
$P(a|e=0)$ $P(a,e=0)=$

$$\sum_{e=0,d,c,b} P(a)P(b|a)P(c|a)P(d|b,a)P(e|b,c)=$$

$$P(a)\sum_{e=0}\sum_{d}\sum_{c}P(c|a)\sum_{b}P(b|a)P(d|b,a)P(e|b,c)$$

Variable Elimination

$h^B(a,d,c,e)$



Bucket Elimination

Algorithm *BE-bel* (Dechter 1996)

$\sum_b \prod$ ← Elimination operator

bucket B: $P(b|a)$ $P(d|b,a)$ $P(e|b,c)$

bucket C: $P(c|a)$ $h^B(a,d,c,e)$

bucket D: $h^C(a,d,e)$

bucket E: $e=0$ $h^D(a,e)$

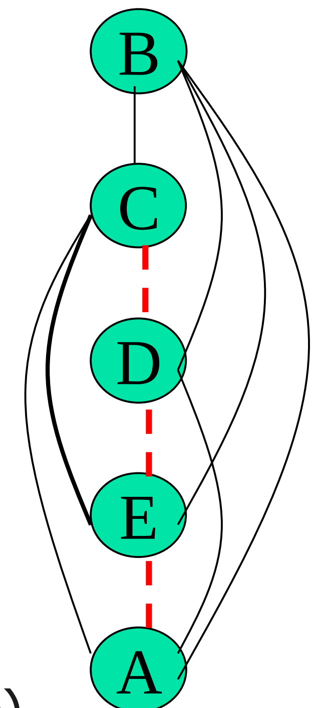
bucket A: $P(a)$ $h^E(a)$

$\boldsymbol{P(a|e=0)}$

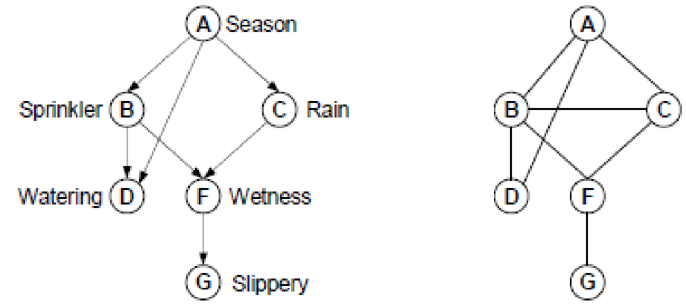
W*=4

"induced width"

(max clique size)



A Bayesian Network
Ordering: A,C,B,E,D,G



(a) Directed acyclic graph (b) Moral graph

$$P(a, g=1) = \sum_{c,b,e,d,g=1} P(a,b,c,d,e,g) = \sum_{c,b,f,d,g=1} P(g|f)P(f|b,c)P(d|a,b)P(c|a)P(b|a)P(a).$$

$$P(a, g=1) = P(a)\sum_{c} P(c|a)\sum_{b} P(b|a)\sum_{f} P(f|b,c)\sum_{d} P(d|b,a)\sum_{g=1} P(g|f). \quad (4.1)$$

$$P(a, g=1) = P(a)\sum_{c} P(c|a)\sum_{b} P(b|a)\sum_{f} P(f|b,c)\lambda_G(f)\sum_{d} P(d|b,a). \quad (4.2)$$

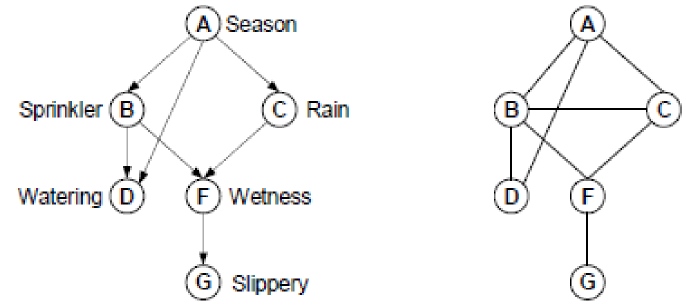
$$P(a, g=1) = P(a)\sum_{c} P(c|a)\sum_{b} P(b|a)\lambda_D(a,b)\sum_{f} P(f|b,c)\lambda_G(f) \quad (4.3)$$

$$P(a, g=1) = P(a)\sum_{c} P(c|a)\sum_{b} P(b|a)\lambda_D(a,b)\lambda_F(b,c) \quad (4.4)$$

$$P(a, g=1) = P(a)\sum_{c} P(c|a)\lambda_B(a,c) \quad (4.5)$$

A Bayesian Network

Ordering: A,C,B,E,D,G



(a) Directed acyclic graph (b) Moral graph

$$P(a, g = 1) = \sum_{c,b,e,d,g=1} P(a, b, c, d, e, g) = \sum_{c,b,f,d,g=1} P(g|f)P(f|b, c)P(d|a, b)P(c|a)P(b|a)P(a).$$

$$P(a, g = 1) = P(a) \sum_{c} P(c|a) \sum_{b} P(b|a) \sum_{f} P(f|b, c) \sum_{d} P(d|b, a) \sum_{g=1} P(g|f). \quad (4.1)$$

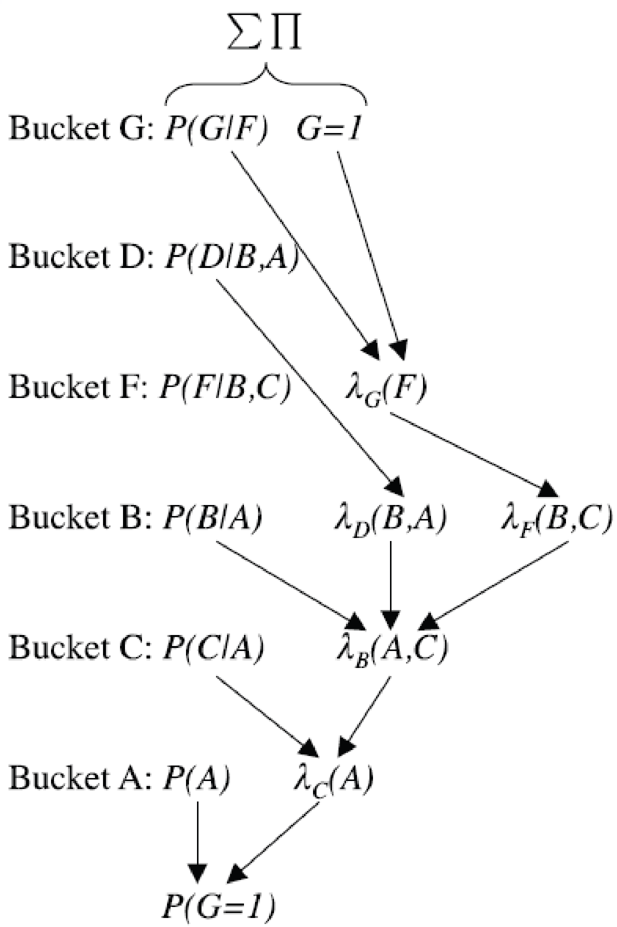
$$P(a, g = 1) = P(a) \sum_{c} P(c|a) \sum_{b} P(b|a) \sum_{f} P(f|b, c)\lambda_G(f) \sum_{d} P(d|b, a). \quad (4.2)$$

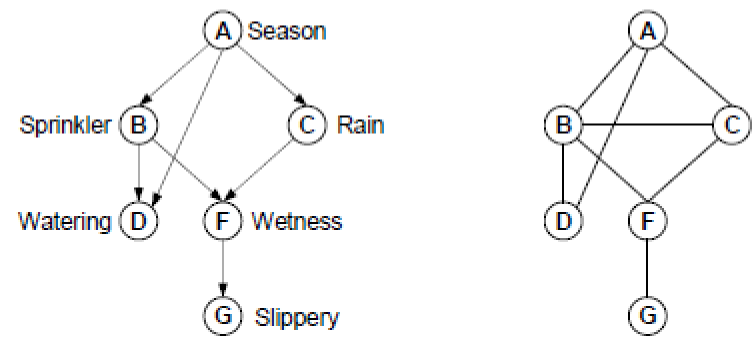
$$P(a, g = 1) = P(a) \sum_{c} P(c|a) \sum_{b} P(b|a)\lambda_D(a, b) \sum_{f} P(f|b, c)\lambda_G(f) \quad (4.3)$$

$$P(a, g = 1) = P(a) \sum_{c} P(c|a) \sum_{b} P(b|a)\lambda_D(a, b)\lambda_F(b, c) \quad (4.4)$$

$$P(a, g = 1) = P(a) \sum_{c} P(c|a)\lambda_B(a, c) \quad (4.5)$$

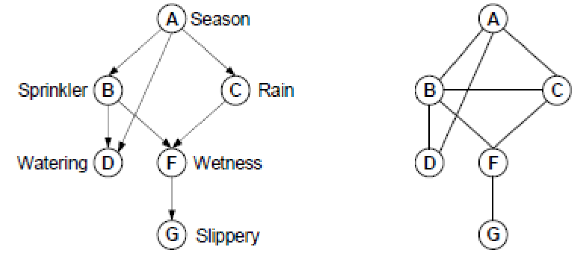
A Bayesian Network
Ordering: A,C,B,F,D,G





(a) Directed acyclic graph (b) Moral graph

A Different Ordering



(a) Directed acyclic graph (b) Moral graph

$$P(a, g=1) = P(a) \sum_f \sum_d \sum_c P(c|a) \sum_b P(b|a)\ P(d|a,b) P(f|b,c) \sum_{g=1} P(g|f)$$

$$= P(a) \sum_f \lambda_G(f) \sum_d \sum_c P(c|a) \sum_b P(b|a)\ P(d|a,b) P(f|b,c)$$

$$= P(a) \sum_f \lambda_G(f) \sum_d \sum_c P(c|a) \lambda_B(a,d,c,f)$$

$$= P(a) \sum_f \lambda_g(f) \sum_d \lambda_C(a,d,f)$$

$$= P(a) \sum_f \lambda_G(f) \lambda_D(a,f)$$

$$= P(a) \lambda_F(a)$$

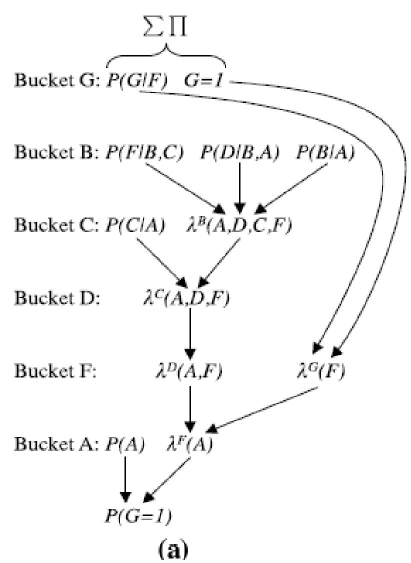
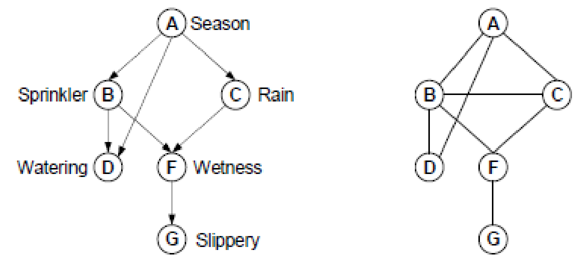


Figure 4.3: The bucket's output when processing along $d_2 = A, F, D, C, B, G$

A Different Ordering



(a) Directed acyclic graph (b) Moral graph

$$P(a, g=1) = P(a)\sum_f \sum_d \sum_c P(c|a)\sum_b P(b|a)\ P(d|a,b)P(f|b,c)\sum_{g=1} P(g|f)$$
$$= P(a)\sum_f \lambda_G(f)\sum_d \sum_c P(c|a)\sum_b P(b|a)\ P(d|a,b)P(f|b,c)$$
$$= P(a)\sum_f \lambda_G(f)\sum_d \sum_c P(c|a)\lambda_B(a,d,c,f)$$
$$= P(a)\sum_f \lambda_g(f)\sum_d \lambda_C(a,d,f)$$
$$= P(a)\sum_f \lambda_G(f)\lambda_D(a,f)$$
$$= P(a)\lambda_F(a)$$

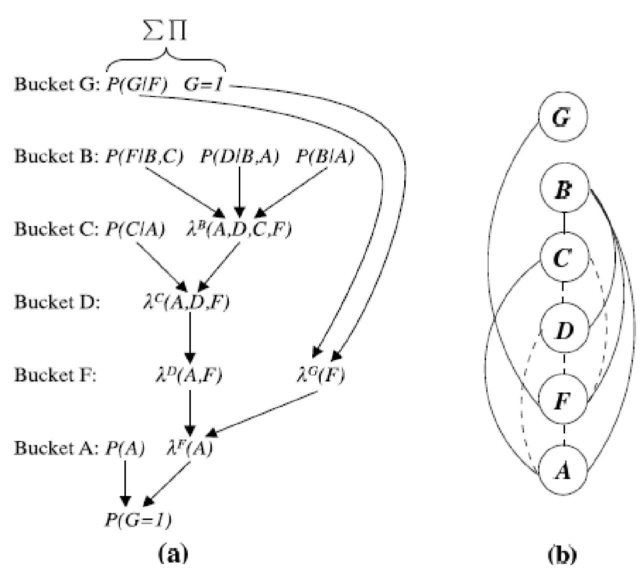
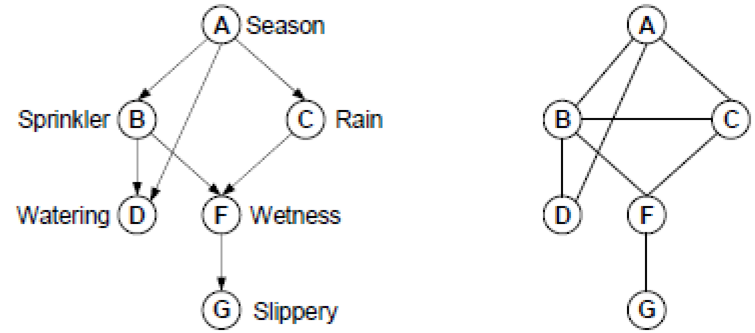
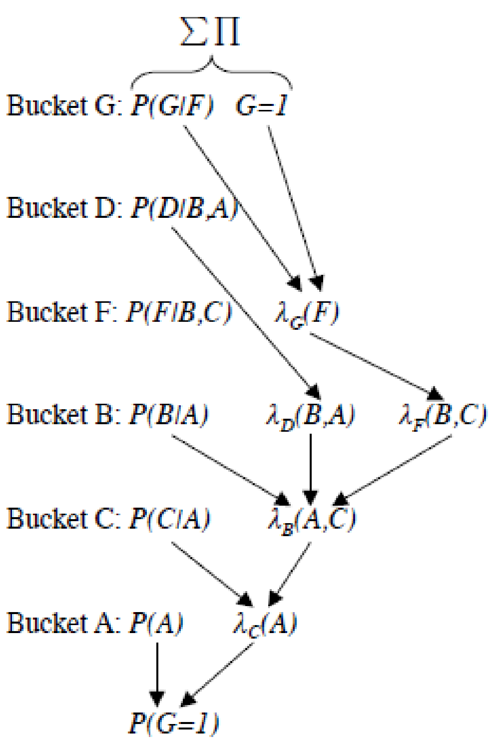


Figure 4.3: The bucket's output when processing along $d_2 = A, F, D, C, B, G$

A Bayesian Network Processed Along 2 Orderings



(a) Directed acyclic graph (b) Moral graph



d1=A,C,B,F,D,G

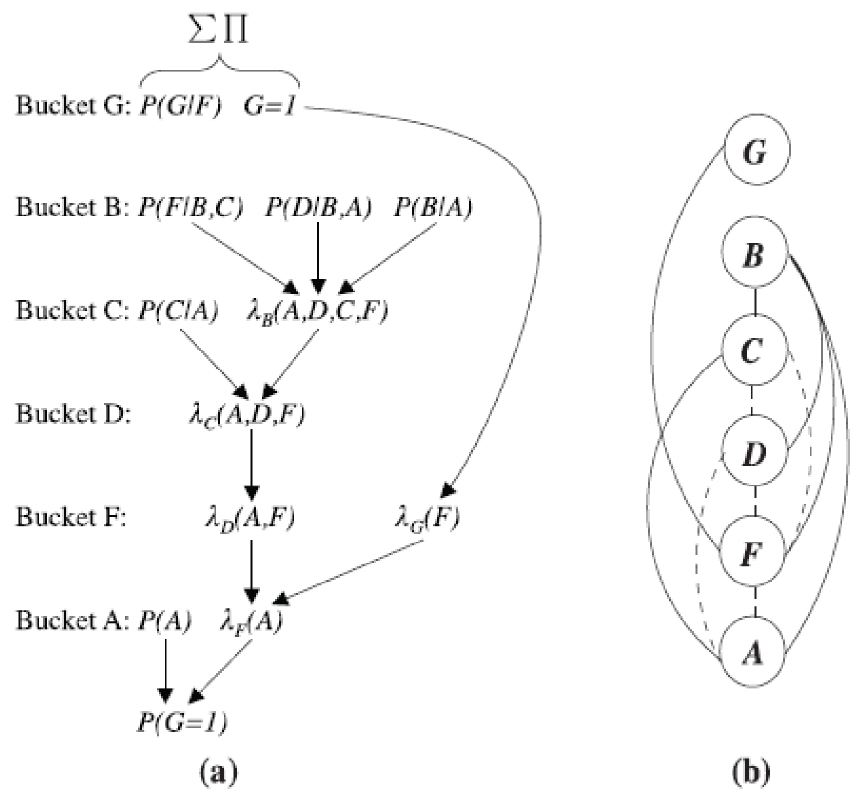


Figure 4.4: The bucket's output when processing along $d_2 = A, F, D, C, B, G$.

Factors: Sum-Out Operation

The sum-out operation is commutative

$$\sum_Y \sum_X f = \sum_X \sum_Y f$$

No need to specify the order in which variables are summed out.

If a factor f is defined over disjoint variables $\mathbf{X}$ and $\mathbf{Y}$

then $\sum_{\mathbf{X}} f$ is said to marginalize variables $\mathbf{X}$

If a factor f is defined over disjoint variables $\mathbf{X}$ and $\mathbf{Y}$

then $\sum_{\mathbf{X}} f$ is called the result of projecting f on variables $\mathbf{Y}$

Factors: Multiplication Operation

B	C	D	f_1
true	true	true	.95
true	true	false	.05
true	false	true	.9
true	false	false	.1
false	true	true	.8
false	true	false	.2
false	false	true	0
false	false	false	1

D	E	f_2
true	true	0.448
true	false	0.192
false	true	0.112
false	false	0.248

The result of multiplying the above factors:

B	C	D	E	$f_1(B, C, D)f_2(D, E)$
true	true	true	true	$0.4256 = (.95)(.448)$
true	true	true	false	$0.1824 = (.95)(.192)$
true	true	false	true	$0.0056 = (.05)(.112)$
⋮	⋮	⋮	⋮	⋮
false	false	false	false	$0.2480 = (1)(.248)$

Factors: Multiplication Operation

The result of multiplying factors $f_1(\mathbf{X})$ and $f_2(\mathbf{Y})$

is another factor over variables $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$:

$$(f_1 f_2)(\mathbf{z}) \stackrel{def}{=} f_1(\mathbf{x}) f_2(\mathbf{y}),$$

where $\mathbf{x}$ and $\mathbf{y}$ are compatible with $\mathbf{z}$; that is, $\mathbf{x} \sim \mathbf{z}$ and $\mathbf{y} \sim \mathbf{z}$

Factor multiplication is commutative and associative

It is meaningful to talk about multiplying a number of factors without specifying the order of this multiplication process.

ALGORITHM BE-BEL

Input: A belief network $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}_G, \prod \rangle$, an ordering $d = (X_1, \ldots, X_n)$; evidence e
output: The belief $P(X_1|\mathrm{e})$ and probability of evidence $P(\mathrm{e})$

1. Partition the input functions (CPTs) into $bucket_1$, …, $bucket_n$ as follows: **for** $i \leftarrow n$ **downto** 1, put in $bucket_i$ all unplaced functions mentioning X_i. Put each observed variable in its bucket. Denote by ψ_i the product of input functions in $bucket_i$.
2. **backward: for** $p \leftarrow n$ **downto** 1 **do**
3. **for** all the functions $\psi_{S_0}, \lambda_{S_1}, \ldots, \lambda_{S_j}$ in $bucket_p$ **do**

    If (observed variable) $X_p = x_p$ appears in $bucket_p$,

    assign $X_p = x_p$ to each function in $bucket_p$ and then

    put each resulting function in the bucket of the *closest* variable in its scope.

    else,

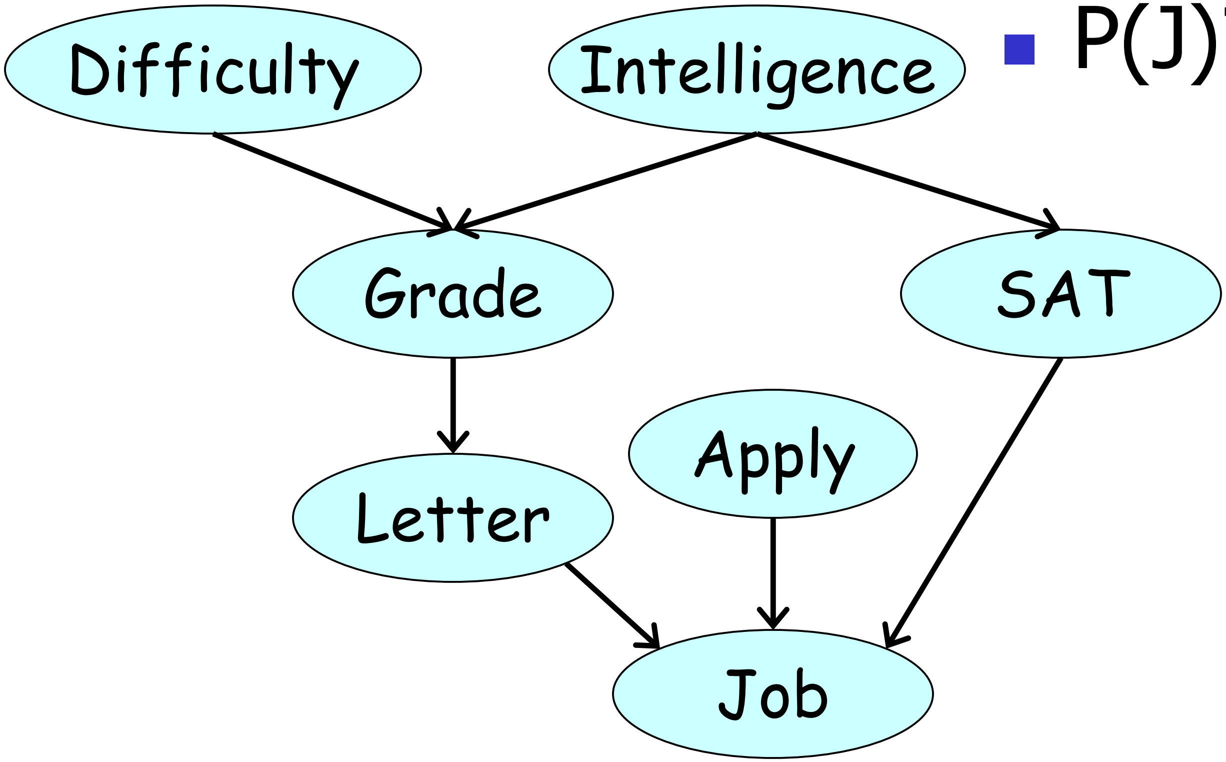
4. $\lambda_p \leftarrow \sum_{X_p} \psi_p \cdot \prod_{i=1}^{j} \lambda_{S_i}$
5. place λ_p in bucket of the latest variable in scope(λ_p),
6. **return** (as a result of processing $bucket_1$):

    $P(\mathrm{e}) = \alpha = \sum_{X_1} \psi_1 \cdot \prod_{\lambda \in bucket_1} \lambda$

    $P(X_1|\mathrm{e}) = \frac{1}{\alpha}\psi_1 \cdot \prod_{\lambda \in bucket_1} \lambda$

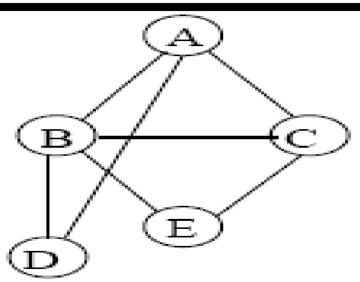
Figure 4.5: BE-bel: a sum-product bucket-elimination algorithm.

Student Network example



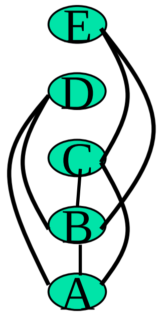
- P(J)?

Bucket Elimination and Induced Width



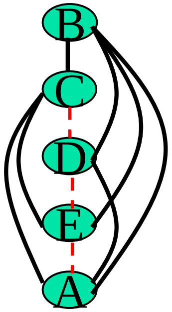
Ordering: a, b, c, d, e

$bucket(E) = \quad P(e|b,c), \;\; e = 0$
$bucket(D) = \quad P(d|a,b)$
$bucket(C) = \quad P(c|a) \;\; || \;\; P(e=0|b,c)$
$bucket(B) = \quad P(b|a) \;\; || \;\; \lambda_D(a,b), \lambda_C(b,c)$
$bucket(A) = \quad P(a) \;\; || \;\; \lambda_B(a)$

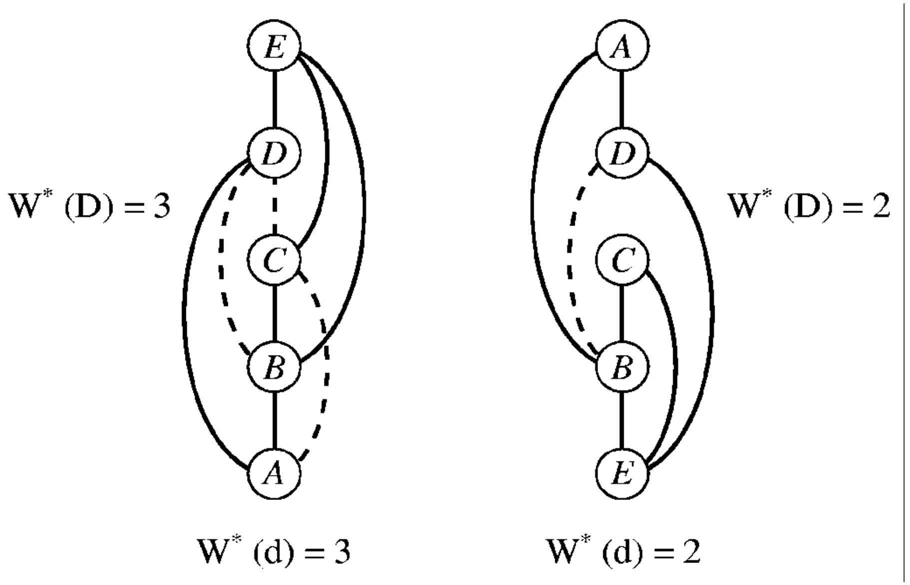


Ordering: a, e, d, c, b

$bucket(B) = \quad P(e|b,c), P(d|a,b), P(b|a)$
$bucket(C) = \quad P(c|a) \;\; || \;\; \lambda_B(a,c,d,e)$
$bucket(D) = \quad || \; \lambda_C(a,d,e)$
$bucket(E) = \quad e = 0 \;\; || \; \lambda_D(a,c)$
$bucket(A) = \quad P(a) \;\; || \;\; \lambda_E(a)$



The Induced-Width



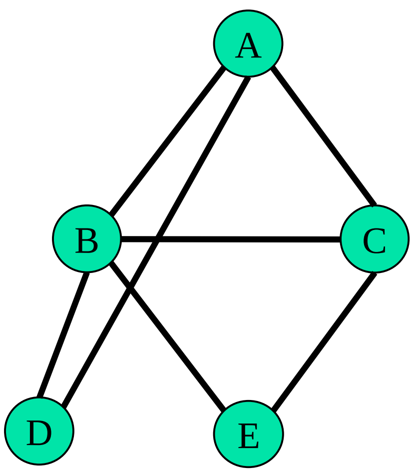
- Width is the max number of parents in the ordered graph
- Induced-width is the width of the induced orderedgraph: recursively connecting parents going from last node to first.
- Induced-width w*(d) is the max induced-width over all nodes in ordering d
- Induced-width of a graph, w* is the min w*(d) over all orderings d

Complexity of Elimination

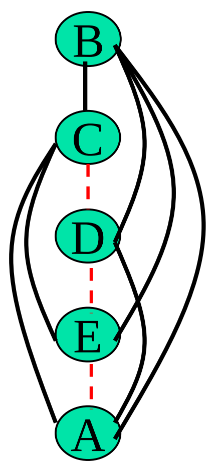
$$O(n \cdot k^{(w^*(d))})$$

$w^*(d)$ – the induced width of moral graph along ordering d

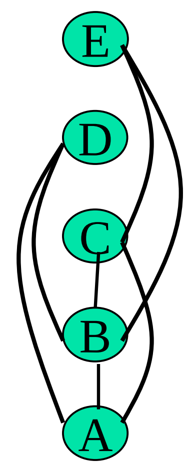
The effect of the ordering:



"Moral" graph



$$w^*(d_1) = 4$$



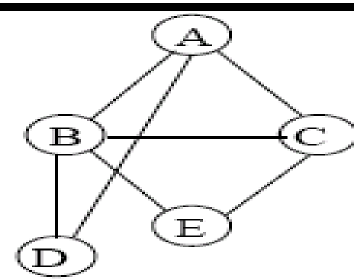
$$w^*(d_2) = 2$$

Complexity of BE-bel

Theorem 4.6 Complexity of BE-bel. *Given a Byaesian network whose moral graph is G, let $w^*(d)$ be its induced width of G along ordering d, k the maximum domain size, and r be the number of input $CPTs$. The time complexity of BE-bel is $O(r \cdot k^{w^*(d)+1})$ and its space complexity is $O(n \cdot k^{w^*(d)})$ (see Appendix for a proof).*

More accurately: O(r exp(w*(d)) where r is the number of cpts.
For Bayesian networks r=n. For Markov networks?

Handling Observations



Observing $b = 1$

Ordering: a, e, d, c, b

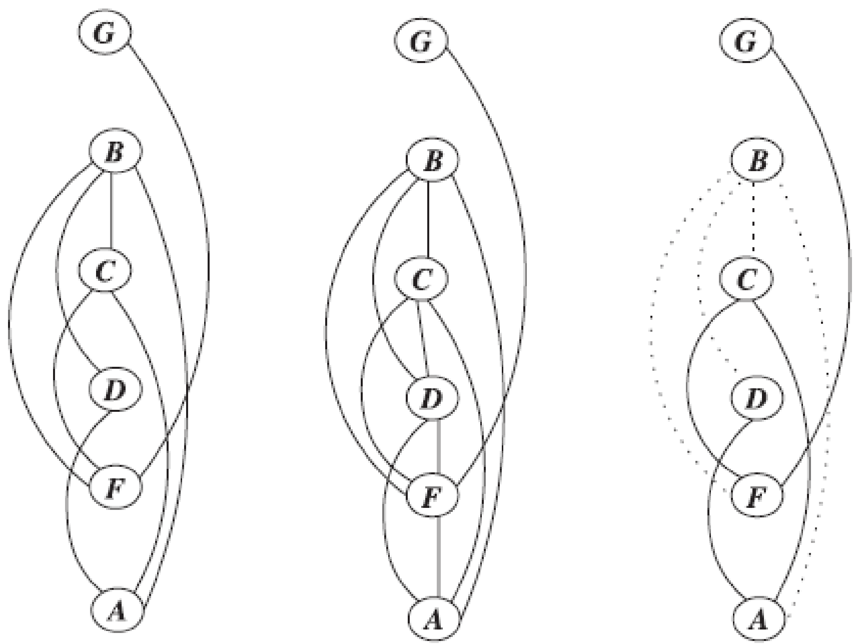
$bucket(B) = \quad P(e|b,c), P(d|a,b), P(b|a), b = 1$
$bucket(C) = \quad P(c|a), \quad || \quad P(e|b = 1, c)$
$bucket(D) = \quad\quad || \quad P(d|a, b = 1)$
$bucket(E) = \quad e = 0 \quad || \quad \lambda_C(e,a)$
$bucket(A) = \quad P(a), \quad || \quad P(b = 1|a) \;\; \lambda_D(a), \lambda_E(e,a)$

Ordering: a, b, c, d, e

$bucket(E) = \quad P(e|b,c), \;\; e = 0$
$bucket(D) = \quad P(d|a,b)$
$bucket(C) = \quad P(c|a) \quad || \quad \lambda_E(b,c)$
$bucket(B) = \quad P(b|a), b = 1 \quad || \quad \lambda_D(a,b), \lambda_C(a,b)$
$bucket(A) = \quad P(a) \quad || \quad \lambda_B(a)$

The impact of observations

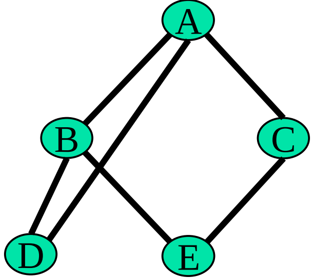
(a) Directed acyclic graph (b) Moral graph



(a) (b) (c)

Figure 4.9: Adjusted induced graph relative to observing B.

Ordered graph Induced graph Ordered conditioned graph



"Moral" graph

Irrelevant buckets for BE-BEL

Buckets that sum to 1 are **irrelevant**.
Identification: no evidence, no new functions.

Recursive recognition : ($bel(a|e)$)

$bucket(E) =$ $P(e|b,c),\ e = 0$
$bucket(D) =$ $P(d|a,b)$,...skipable bucket
$bucket(C) =$ $P(c|a)$
$bucket(B) =$ $P(b|a)$
$bucket(A) =$ $P(a)$

Complexity: Use induced width in moral graph without irrelevant nodes, then update for evidence arcs.

Use the ancestral graph only

Pruning Nodes

Given a Bayesian network $\mathcal{N}$ and query $(\mathbf{Q}, \mathbf{e})$

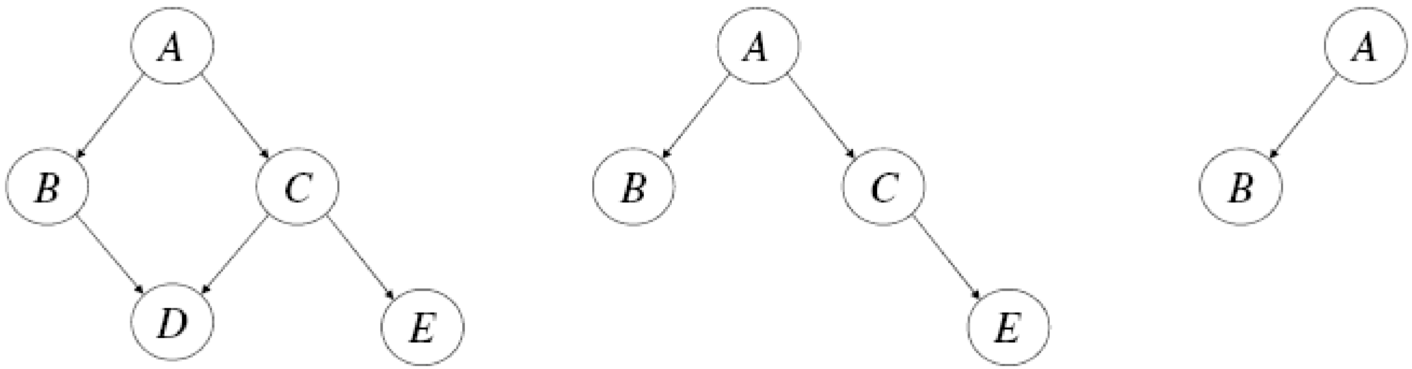
one can remove any leaf node (with its CPT) from the network as long as it does not belong to variables $\mathbf{Q} \cup \mathbf{E}$, yet not affect the ability of the network to answer the query correctly.

If $\mathcal{N}' = \text{pruneNodes}(\mathcal{N}, \mathbf{Q} \cup \mathbf{E})$

then $\Pr(\mathbf{Q}, \mathbf{e}) = \Pr'(\mathbf{Q}, \mathbf{e})$, where $\Pr$ and $\Pr'$ are the probability distributions induced by networks $\mathcal{N}$ and $\mathcal{N}'$, respectively.

Pruning Nodes: Example

Example of pruning irrelevant subnetworks

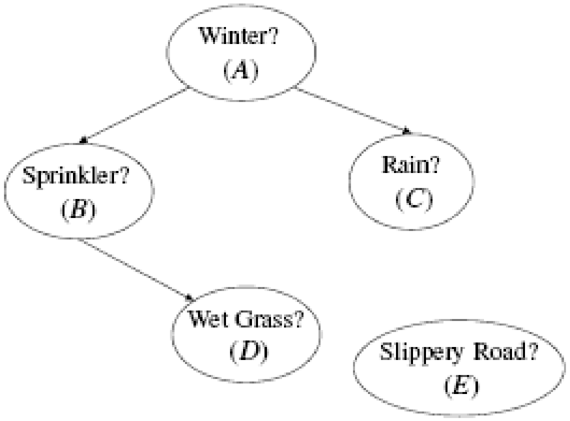


network structure — joint on B, E — joint on B

Pruning Edges: Example

Example of pruning edges due to evidence or conditioning

A	B	$\Theta_{B\|A}$
true	true	.2
true	false	.8
false	true	.75
false	false	.25



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

A	Θ_A
true	.6
false	.4

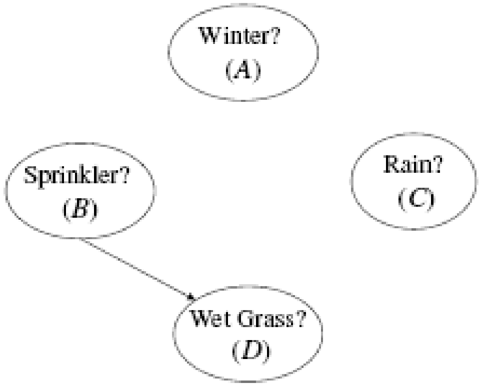
B	D	$\sum_C \Theta_{D\|BC}^{C=\text{false}}$
true	true	.9
true	false	.1
false	true	0
false	false	1

E	$\sum_C \Theta_{E\|C}^{C=\text{false}}$
true	0
false	1

Evidence **e** : $C = \text{false}$

Pruning Nodes and Edges: Example

B	$\Theta'_B = \sum_A \Theta_{B\|A}^{A=\text{true}}$
true	.2
false	.8



C	$\Theta'_C = \sum_A \Theta_{C\|A}^{A=\text{true}}$
true	.8
false	.2

A	Θ_A
true	.6
false	.4

B	D	$\Theta'_{D\|B} = \sum_C \Theta_{D\|BC}^{C=\text{false}}$
true	true	.9
true	false	.1
false	true	0
false	false	1

Query $\mathbf{Q} = \{D\}$ and $\mathbf{e} : A = \text{true}, C = \text{false}$

Probabilistic Inference Tasks

- Belief updating:

$$BEL(X_i) = P(X_i = x_i \mid evidence)$$

- Finding most probable explanation (MPE)

$$\bar{x}^* = \arg\max_{\bar{x}} P(\bar{x}, e)$$

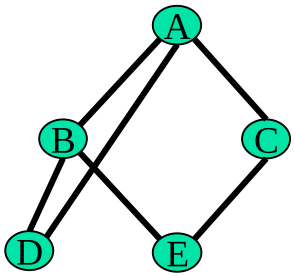
- Finding maximum a-posteriory hypothesis

$$(a_1^*, \ldots, a_k^*) = \arg\max_{\bar{a}} \sum_{X/A} P(\bar{x}, e)$$

$A \subseteq X$:
hypothesis variables

Finding $MPE = \max_{\bar{x}} P(\bar{x})$

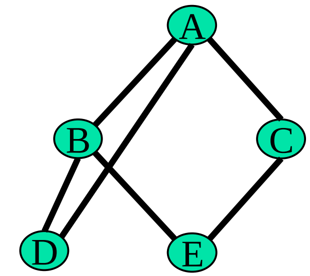
Algorithm *BE-mpe*



$\sum$ is replaced by ***max*** :

$$MPE = \max_{a,e,d,c,b} P(a)P(c|a)P(b|a)P(d|a,b)P(e|b,c)$$

Finding $MPE = \max_{\bar{x}} P(\bar{x})$



Algorithm *elim-mpe* (Dechter 1996)

$\sum$ is replaced by ***max*** **:**

$$MPE = \max_{a,e,d,c,b} P(a)P(c \mid a)P(b \mid a)P(d \mid a,b)P(e \mid b,c)$$

$\max_{b} \prod$ ← Elimination operator

bucket B: P(b|a) P(d|b,a) P(e|b,c)

bucket C: P(c|a) $h^{B}(a,d,c,e)$

bucket D: $h^{C}(a,d,e)$

bucket E: e=0 $h^{D}(a,e)$

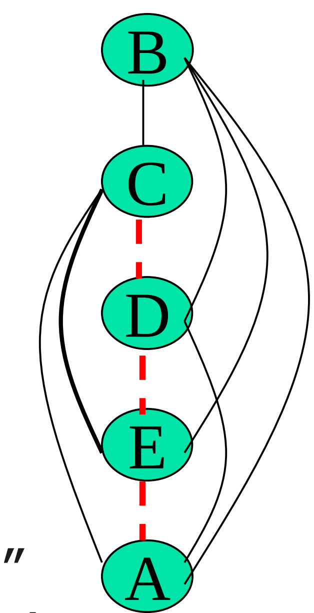
bucket A: P(a) $h^{E}(a)$

MPE

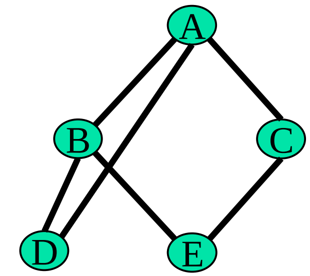
W*=4

"induced width"

(max clique size)



Finding $MPE = \max_{\bar{x}} P(\bar{x})$



Algorithm *elim-mpe* (Dechter 1996)

$\sum$ is replaced by ***max*** :

$$MPE = \max_{a,e,d,c,b} P(a)P(c\,|\,a)P(b\,|\,a)P(d\,|\,a,b)P(e\,|\,b,c)$$

$\max_{b} \prod$ ← Elimination operator

bucket B: P(b|a) P(d|b,a) P(e|b,c)

bucket C: P(c|a) $h^B(a,d,c,e)$

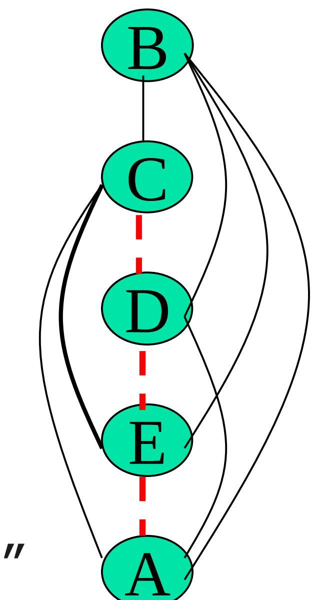
bucket D: $h^C(a,d,e)$

bucket E: e=0 $h^D(a,e)$

bucket A: P(a) $h^E(a)$

MPE

W*=4
"induced width"
(max clique size)



Generating the MPE-tuple

5. $b' = \arg\max_b P(b \mid a') \times P(d' \mid b, a') \times P(e' \mid b, c')$

4. $c' = \arg\max_c P(c \mid a') \times h^B(a', d', c, e')$

3. $d' = \arg\max_d h^C(a', d, e')$

2. $e' = 0$

1. $a' = \arg\max_a P(a) \cdot h^E(a)$

B: P(b|a) P(d|b,a) P(e|b,c)

C: P(c|a) $h^B(a,d,c,e)$

D: $h^C(a,d,e)$

E: e=0 $h^D(a,e)$

A: P(a) $h^E(a)$

Return (a', b', c', d', e')

Algorithm BE-mpe

Input: A belief network $\mathcal{B} =< X, D, G, \mathcal{P} >$, where $\mathcal{P} = \{P_1, ..., P_n\}$; an ordering of the variables, $d = X_1, ..., X_n$; observations e.

Output: The most probable assignment given the evidence.

1. **Initialize:** Generate an ordered partition of the conditional probability function, $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all functions whose highest variable is X_i. Put each observed variable in its bucket. Let ψ_i be the input function in a bucket and let h_i be the messages in the bucket.

2. **Backward:** For $p \leftarrow n$ downto 1, do

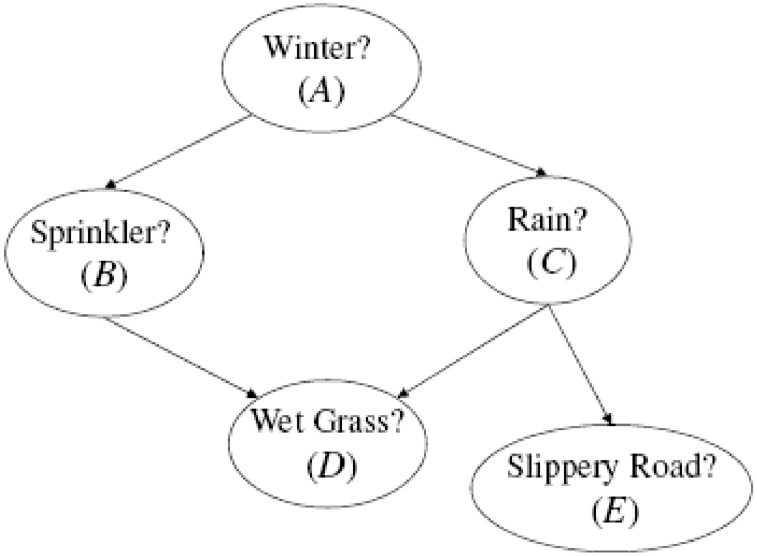
for all the functions $h_1, h_2, ..., h_j$ in $bucket_p$, do

- **If** (observed variable) $bucket_p$ contains $X_p = x_p$, assign $X_p = x_p$ to each function and put each in appropriate bucket.
- **else,** $S_p \leftarrow \bigcup_{i=1}^{j} scope(h_i) \cup scope(\psi_p) - \{X_p\}$. Generate functions $h_p \Leftarrow \max_{X_p} \psi_p \cdot \Pi_{i=1}^{j} h_i$ Add h_p to the bucket of the largest-index variable in S_p.

3. **Forward:**

- Generate the mpe cost by maximizing over X_1, the product in $bucket_1$.
- (generate an mpe tuple)
  For $i = 1$ to n along d do: Given $\overline{x}_{i-1} = (x_1, ..., x_{i-1})$ Choose $x_i = argmax_{X_i} \psi_i \cdot \Pi_{\{h_j \in \ bucket_i\}} h_j(\overline{x}_{i-1})$

Try to compute MPE when E=0



A	Θ_A
true	.6
false	.4

A	B	$\Theta_{B\|A}$
true	true	.2
true	false	.8
false	true	.75
false	false	.25

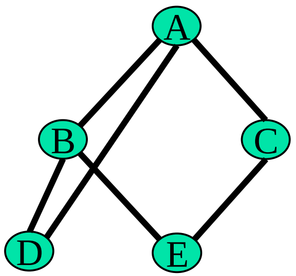
A	C	$\Theta_{C\|A}$
true	true	.8
true	false	.2
false	true	.1
false	false	.9

B	C	D	$\Theta_{D\|BC}$
true	true	true	.95
true	true	false	.05
true	false	true	.9
true	false	false	.1
false	true	true	.8
false	true	false	.2
false	false	true	0
false	false	false	1

C	E	$\Theta_{E\|C}$
true	true	.7
true	false	.3
false	true	0
false	false	1

Finding MAP

Algorithm *BE-map*

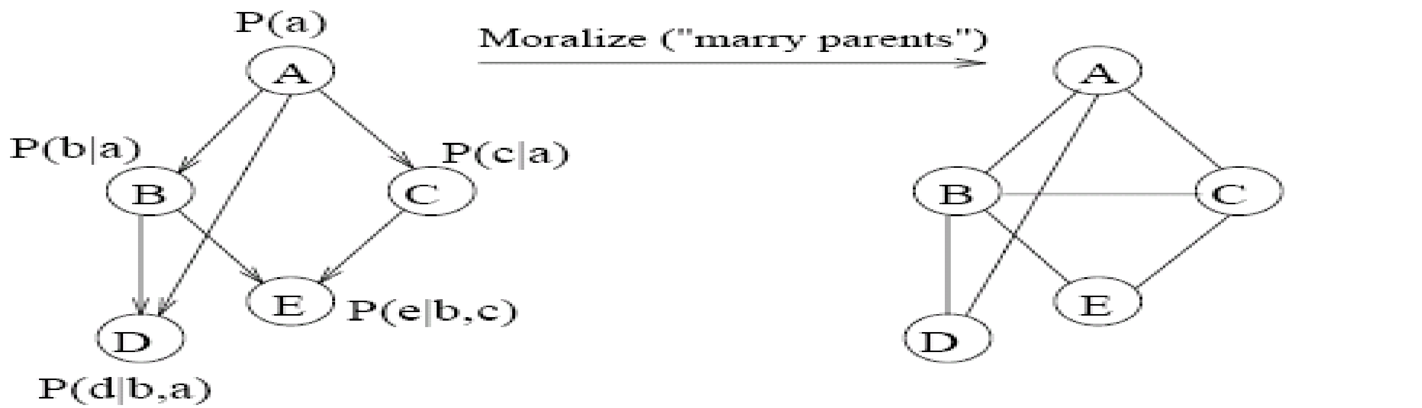


$\sum$ ***and max :***

$$MPE = \max_{a,c} P(a)P(c \mid a)\sum_{e,d,b} P(b \mid a)P(d \mid a,b)P(e \mid b,c)$$

Finding the MAP

(An optimization task)



Variables A and B are the hypothesis variables.
Ordering: $a,\ b,\ c,\ d,\ e$
$max_{a,b} P(a,b,e=0) = \max_{a,b} \sum_{c,d,e=0} P(a,b,c,d,e)$
$= \max_a P(a) \max_b P(b|a) \sum_c P(c|a) \sum_d P(d|b,a)$
$\sum_{e=0} P(e|b,c)$

Ordering: $a,\ e,\ d,\ c,\ b$ .... illegal ordering
$\max_{a,b} P(a,e,e=0) = \max_{a,b} \sum_P (a,b,c,d,e)$
$\max_{a,b} P(a,b,e=0) = \max_a P(a) \max_b P(b|a) \sum_d \cdot$
$\max_c P(c|a) P(d|a,b) P(e=0|b,c)$

Algorithm BE-map

Variable ordering:
Restricted: Max buckets should
Be processed after sum buckets

Algorithm BE-map

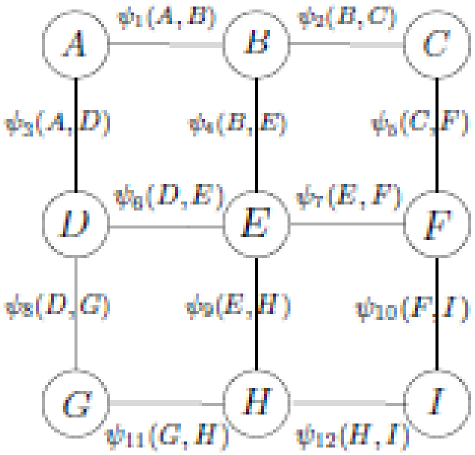
Input: A Bayesian network $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}_G, \prod \rangle$, $P = \{P_1, ..., P_n\}$; a subset of hypothesis variables $A = \{A_1, ..., A_k\}$; an ordering of the variables, d, in which the A's are first in the ordering; observations e. ψ_i is the product of input function in the bucket of X_i.

Output: A most probable assignment $A = a$.

1. **Initialize:** Generate an ordered partition of the conditional probability functions, $bucket_1$, ..., $bucket_n$, where $bucket_i$ contains all functions whose highest variable is X_i.
2. **Backwards** For $p \leftarrow n$ downto 1, do
for all the message functions $\beta_1, \beta_2, ..., \beta_j$ in $bucket_p$ and for ψ_p do
   - **If** (observed variable) $bucket_p$ contains the observation $X_p = x_p$, assign $X_p = x_p$ to each β_i and ψ_p and put each in appropriate bucket.
   - **else**, If X_p is not in A, then $\beta_p \Leftarrow \sum_{X_p} \psi_p \cdot \Pi_{i=1}^{j} \beta_i$;
   else, $(X_p \in A)$, $\beta_p \Leftarrow \max_{X_p} \psi_p \cdot \prod_{i=1}^{j} \beta_i$
   Place β_p in the bucket of the largest-index variable in $scope(\beta_p)$.
3. **Forward:** Assign values, in the ordering $d = A_1, ..., A_k$, using the information recorded in each bucket in a similar way to the forward pass in BE-mpe.
4. **Output:** Map and the corresponding configuration over A.

Theorem 4.16 *Algorithm BE-map is complete for the map task for orderings started by the hypothesis variables. Its time and space complexity are $O(r \cdot k^{w^*_{\mathbf{E}}(d)+1})$ and $O(n \cdot k^{w^*_{\mathbf{E}}(d)})$, respectively, where n is the number of variables in graph, k bounds the domain size and $w^*_{\mathbf{E}}(d)$ is the conditioned induced width of its moral graph along d, relative to evidence variables $\mathbf{E}$. (Prove as an exercise.)* □

BE for Markov networks queries



(a)

D	E	$\psi_6(D,E)$
0	0	20.2
0	1	12
1	0	23.4
1	1	11.7

(b)

Complexity of bucket elimination

Theorem

Given a belief network having n variables, observations e, the complexity of elim-mpe, elim-bel, elim-map along d, is time and space

O(nexp(w*+1)) and O(n exp(w*)), respectively

where $w*(d)$ is the induced width of the moral graph whose edges connecting evidence to earlier nodes, were deleted.

More accurately: O(r exp(w*(d)) where r is the number of cpts.
For Bayesian networks r=n. For Markov networks?

Finding Small Induced-Width
(Dechter 3.4-3.5)

- NP-complete
- A tree has induced-width of ?
- Greedy algorithms:
  - Min width
  - Min induced-width
  - Max-cardinality and chordal graphs
  - Fill-in (thought as the best)
  - See anytime min-width (Gogate and Dechter)

Type of graphs

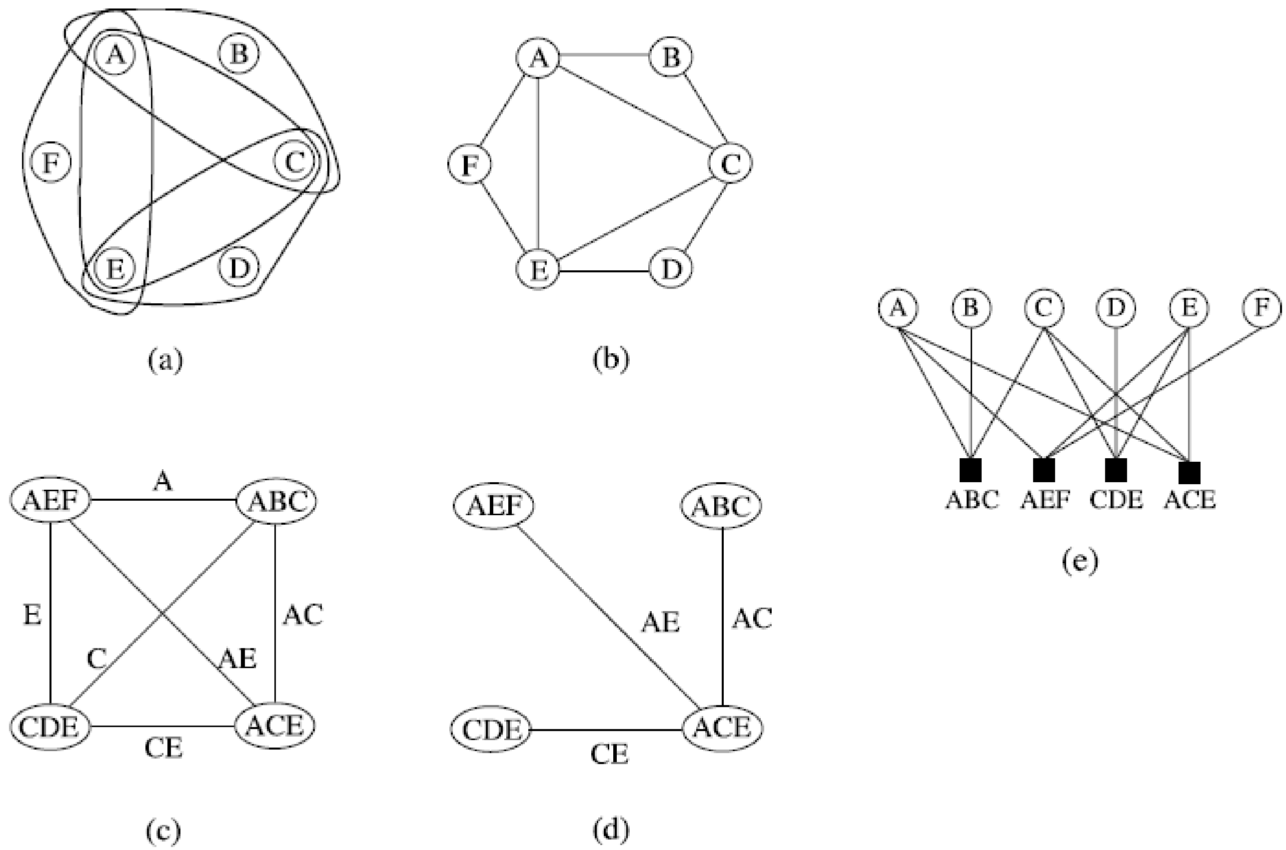


Figure 5.1: (a)Hyper, (b)Primal, (c)Dual and (d)Join-tree of a graphical model having scopes ABC, AEF, CDE and ACE. (e) the factor graph

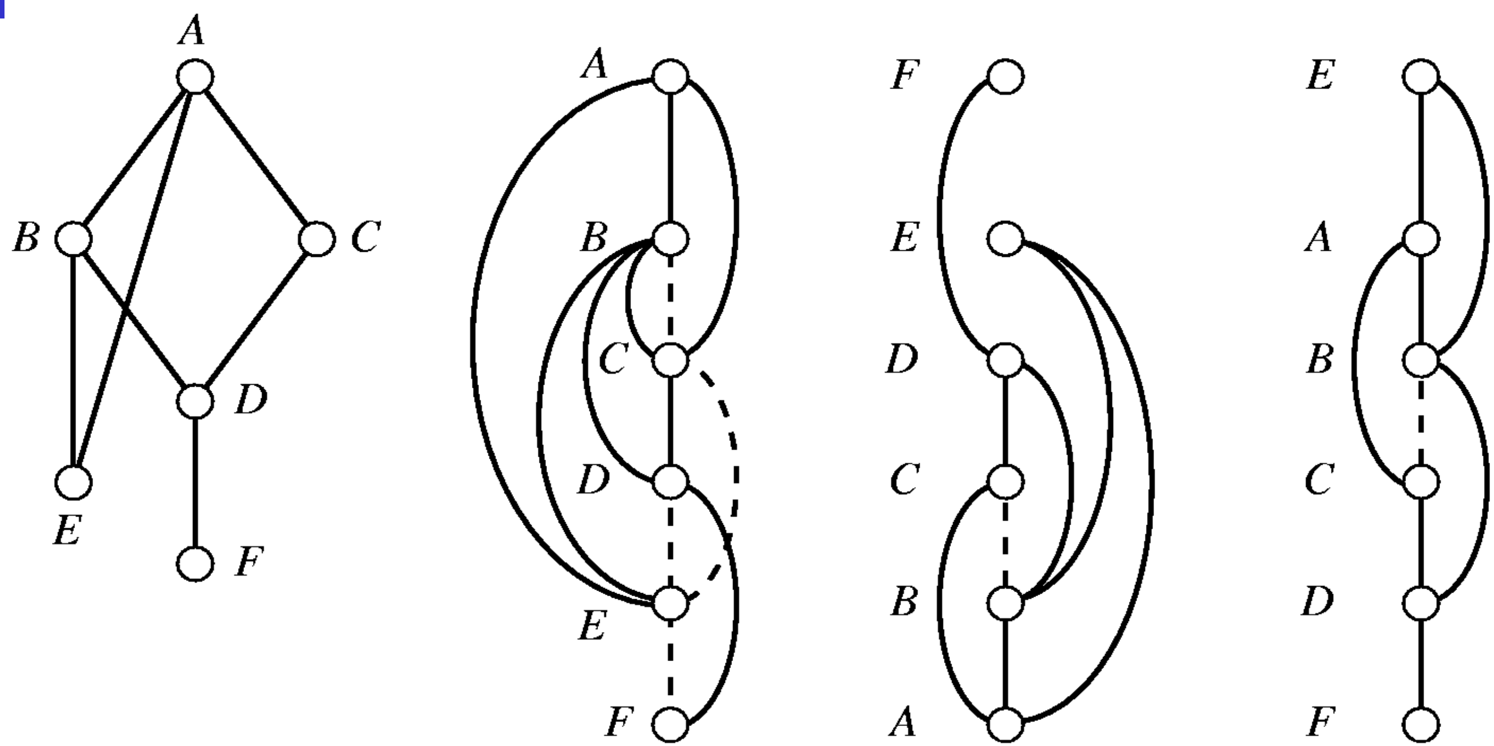
The induced width

Definition 5.2.1 (width) *Given an undirected graph* $G = (V, E)$, *an* ordered graph *is a pair* (G, d), *where* $V = \{v_1, ..., v_n\}$ *is the set of nodes,* E *is a set of arcs over* V, *and* $d = (v_1, ..., v_n)$ *is an ordering of the nodes. The nodes adjacent to* v *that precede it in the ordering are called its* parents. *The* width of a node *in an ordered graph is its number of parents. The* width of an ordering d *of* G, *denoted* $w_d(G)$ *(or* w_d *for short) is the maximum width over all nodes. The* width of a graph *is the minimum width over all the orderings of the graph.*

Definition 5.2.3 (induced width) *The* induced width of an ordered graph (G, d), *denoted* $w^*{}_d$), *is the width of the* induced ordered graph *along* d *obtained as follows: nodes are processed from last to first; when node* v *is processed, all its parents are connected. The* **induced width** *of a graph, denoted by* w^*, *is the minimal induced width over all its orderings. Formally*

$$w^*(G) = \min_{d \in orderings} w^*{}_d(G)$$

Different Induced-graphs



(a) (b) (c) (d)

Min-Width Ordering

MIN-WIDTH (MW)

input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$
output: A min-width ordering of the nodes $d = (v_1, ..., v_n)$.
1. **for** $j = n$ to 1 by -1 do
2. $r \leftarrow$ a node in G with smallest degree.
3. put r in position j and $G \leftarrow G - r$.
   (Delete from V node r and from E all its adjacent edges)
4. **endfor**

Proposition: (Freuder 1982) algorithm min-width finds a min-width ordering of a graph. Complexity O(|E|)

Greedy Orderings Heuristics

MIN-INDUCED-WIDTH (MIW)

input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$

output: An ordering of the nodes $d = (v_1, ..., v_n)$.

1. **for** $j = n$ to 1 by -1 do
2. $r \leftarrow$ a node in V with smallest degree.
3. put r in position j.
4. connect r's neighbors: $E \leftarrow E \cup \{(v_i, v_j) | (v_i, r) \in E, (v_j, r) \in E\}$,
5. remove r from the resulting graph: $V \leftarrow V - \{r\}$.

Theorem: A graph is a tree iff it has both width and induced-width of 1.

MIN-FILL (MIN-FILL)

input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$

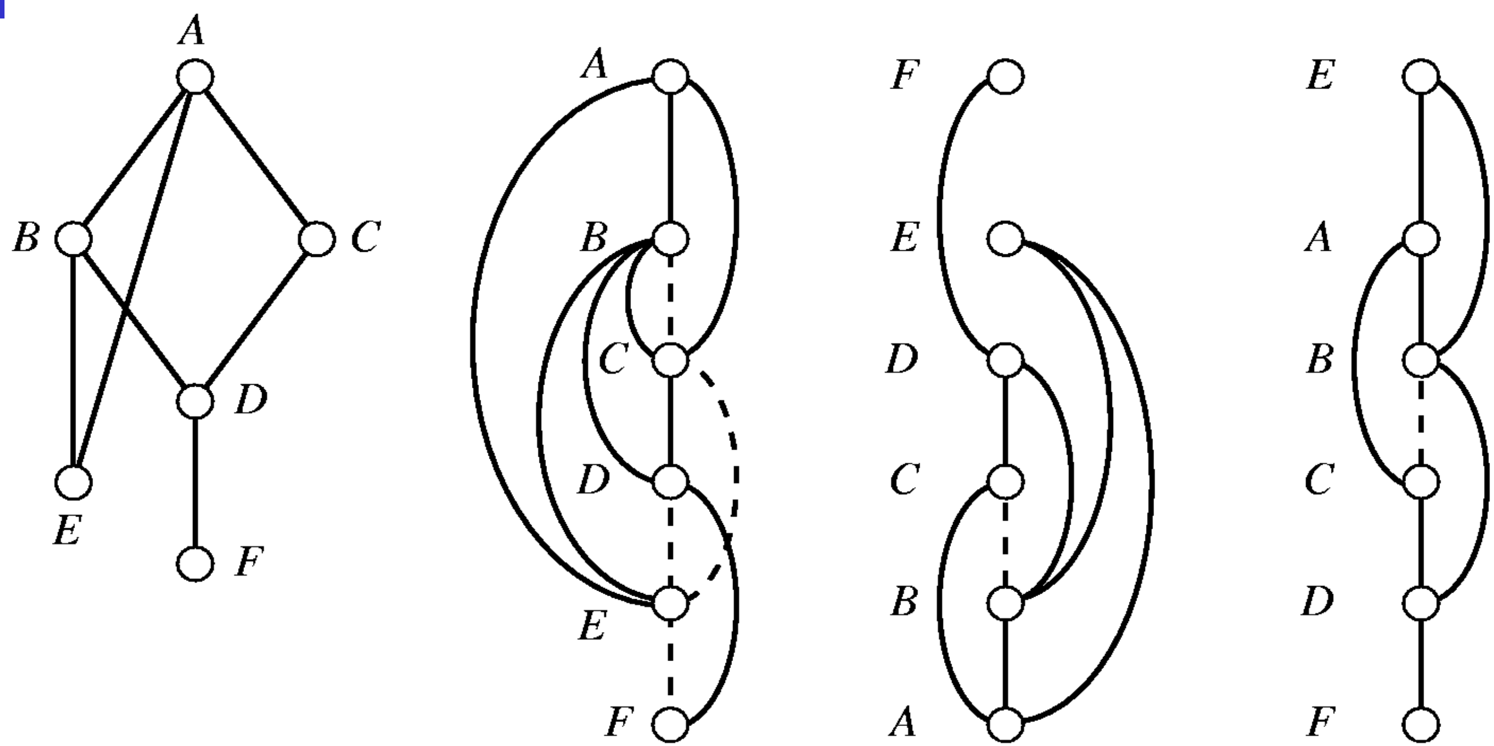
output: An ordering of the nodes $d = (v_1, ..., v_n)$.

1. **for** $j = n$ to 1 by -1 do
2. $r \leftarrow$ a node in V with smallest fill edges for his parents.
3. put r in position j.
4. connect r's neighbors: $E \leftarrow E \cup \{(v_i, v_j) | (v_i, r) \in E, (v_j, r) \in E\}$,
5. remove r from the resulting graph: $V \leftarrow V - \{r\}$.

Complexity?

O(n^3)

Different Induced-Graphs



(a) (b) (c) (d)

Induced-width for chordal graphs

- **Definition:** A graph is chordal if every cycle of length at least 4 has a chord
- Finding w* over chordal graph is easy using the **max-cardinality ordering:** order vertices from 1 to n, always assigning the next number to the node connected to a largest set of previously numbered nodes. Lets d be such an ordering
- A graph along max-cardinality order has no fill-in edges iff it is chordal.
- On chordal graphs width=induced-width.

Max-cardinality ordering

MAX-CARDINALITY (MC)

input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$
output: An ordering of the nodes $d = (v_1, ..., v_n)$.
1. Place an arbitrary node in position 0.
2. **for** $j = 1$ to n do
3. $r \leftarrow$ a node in G that is connected to a largest subset of nodes in positions 1 to $j - 1$, breaking ties arbitrarily.
4. **endfor**

Proposition 5.3.3 *[56] Given a graph $G = (V, E)$ the complexity of max-cardinality search is $O(n + m)$ when $|V| = n$ and $|E| = m$.*

What is the complexity of min-fill? Min-induced-width? $O(n^3)$

K-trees

Definition 5.3.4 (k-trees) *A subclass of chordal graphs are k-trees. A k-tree is a chordal graph whose maximal cliques are of size $k+1$, and it can be defined recursively as follows: (1) A complete graph with k vertices is a k-tree. (2) A k-tree with r vertices can be extended to $r+1$ vertices by connecting the new vertex to all the vertices in any clique of size k. A partial k-tree is a k-tree having some of its arcs removed. Namely it will clique of size smaller than k.*

Which greedy algorithm is best?

- MinFill, prefers a node who add the least number of fill-in arcs.
- Empirically, fill-in is the best among the greedy algorithms (MW,MIW,MF,MC)
- Complexity of greedy orderings?
  - MW is $O(n^2)$…maybe $O(nlogn + m\ \ )$?
  - MIW: O($O(n^3)$,
  - MF ($O(n^3)$,
  - MC is O(m+n), m edges.

Recent work in my group

- **Vibhav Gogate and Rina Dechter.** "A Complete Anytime Algorithm for Treewidth". *In UAI 2004.*
- **Andrew E. Gelfand, Kalev Kask, and Rina Dechter.** "Stopping Rules for Randomized Greedy Triangulation Schemes" in *Proceedings of AAAI 2011.*
- Kask, Gelfand and Dechter, BEEM: Bucket Elimination with External memory, AAAI 2011 or UAI 2011
- Potential project

Mixed Networks

- Augmenting Probabilistic networks with constraints because:
  - Some information in the world is deterministic and undirected ($X \neq Y$).
  - Some queries are complex or evidence are complex (cnf formulas)
- Queries are probabilistic queries

Mixed Beliefs and Constraints

$$\varphi = (G \vee D) \wedge (\neg D \vee B)$$
$$P(\varphi) = ?$$

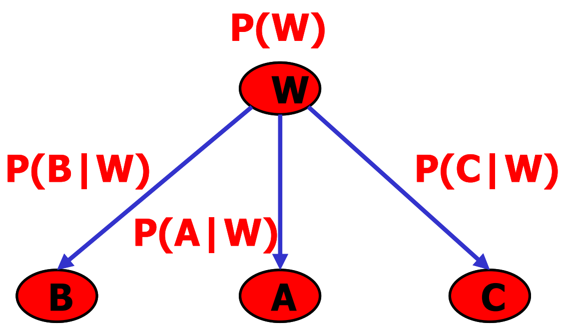
- If the constraint is a cnf formula
- Queries over hybrid network:
- Complex evidence structure

$$P(\bar{x} \mid \varphi) = ?$$
$$P(x_1 \mid \varphi) = ?$$

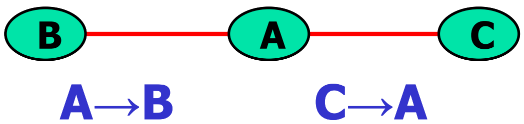
- All reduce to cnf queries over a Belief network:
  - CPE (CNF probability evaluation): Given a belief network, and a cnf formula, find its probability.

Party example again

PN



CN



Semantics?

Algorithms?

Query:
Is it likely that Chris goes to the party if Becky does not but the weather is bad?

$$P(C, \neg B \mid w = bad, A \rightarrow B, C \rightarrow A)$$

Bucket Elimination for Mixed networks

The CPE query

$$P_{\mathcal{B}}(\varphi) = \sum_{\mathbf{x}_\varphi \in Mod(\varphi)} P(\mathbf{x}_\varphi)$$

Using the belief network product form we get:

$$P_{\mathcal{B}}(\varphi) = \sum_{\{\mathbf{x} | \mathbf{x}_\varphi \in Mod(\varphi)\}} \prod_{i=1}^{n} P(x_i | \mathbf{x}_{pa_i}).$$

P((C → B) and P(A → C))

Algorithm 1: BE-CPE

Input: A belief network $\mathcal{M} = (\mathcal{B}, \simeq)$, $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}_G, \prod \rangle$, where $\mathcal{B} = \{P_1, ..., P_n\}$; a CNF formula on k propositions $\varphi = \{\alpha_1, ...\alpha_m\}$ defined over k propositions; an ordering of the variables, $d = \{X_1, \ldots, X_n\}$.

Output: The belief $P(\varphi)$.

1 Place buckets with unit clauses last in the ordering (to be processed first).

`// Initialize`

Partition $\mathcal{B}$ and φ into $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all the CPTs and clauses whose highest variable is X_i.

Put each observed variable into its appropriate bucket. (We denote probabilistic functions by λs and clauses by αs).

2 **for** $p \leftarrow n$ **downto** 1 **do** `// Backward`

  Let $\lambda_1, \ldots, \lambda_j$ be the functions and $\alpha_1, \ldots, \alpha_r$ be the clauses in $bucket_p$

  Process-bucket$_p(\sum, (\lambda_1, \ldots, \lambda_j), (\alpha_1, \ldots, \alpha_r))$

3 **return** $P(\varphi)$ as the result of processing $bucket_1$.

Procedure `Process-bucket`$_p$ $(\sum, (\lambda_1, \ldots, \lambda_j), (\alpha_1, \ldots, \alpha_r))$.

if *bucket*$_p$ *contains evidence* $X_p = x_p$ **then**

1. Assign $X_p = x_p$ to each λ_i and put each resulting function in the bucket of its latest variable
2. Resolve each α_i with the unit clause, put non-tautology resolvents in the buckets of their latest variable and **move any bucket with unit clause to top of processing**

else

$\lambda_p \leftarrow \sum_{\{x_p | \mathbf{x}_{U_p} \in Mod(\alpha_1,\ldots,\alpha_r)\}} \prod_{i=1}^{j} \lambda_i$

Add λ_p to the bucket of the latest variable in S_p, where

$S_p = scope(\lambda_1, \ldots, \lambda_j, \alpha_1, \ldots, \alpha_r)$, $U_p = scope(\alpha_1, \ldots, \alpha_r)$.

(a) Directed acyclic graph

(b) Moral graph

Processing Mixed Buckets

We compute:

In Bucket G: $\lambda_G(f,d) = \sum_{\{g|g \vee d = true\}} P(g|f)$

In $Bucket_F$: $\lambda_F(b,c,d) = \sum_f P(f|b,c)\lambda_G(f,d)$

In $Bucket_D$: $\lambda_D(a,b,c) = \sum_{\{d|\neg d \vee \neg b = true\}} P(d|a,b)\lambda_F(b,c,d)$

In $Bucket_B$: $\lambda_B(a,c) = \sum_{\{b|b \vee c = true\}} P(b|a)\lambda_D(a,b,c)\lambda_F(b,c)$

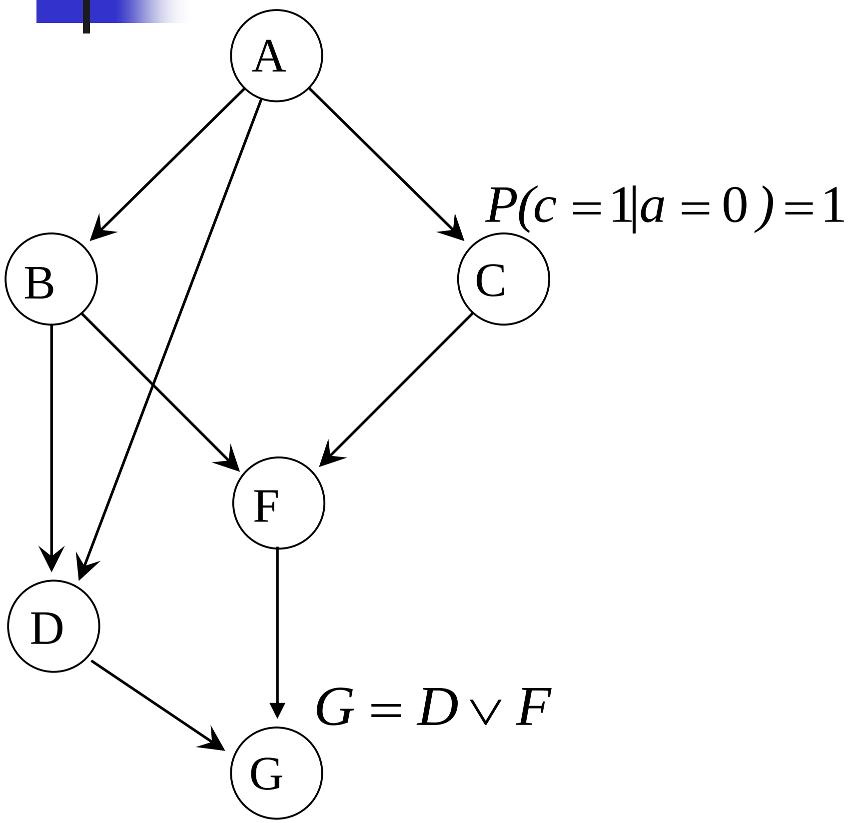
In $Bucket_C$: $\lambda_C(a) = \sum_c P(c|a)\lambda_B(a,c)$

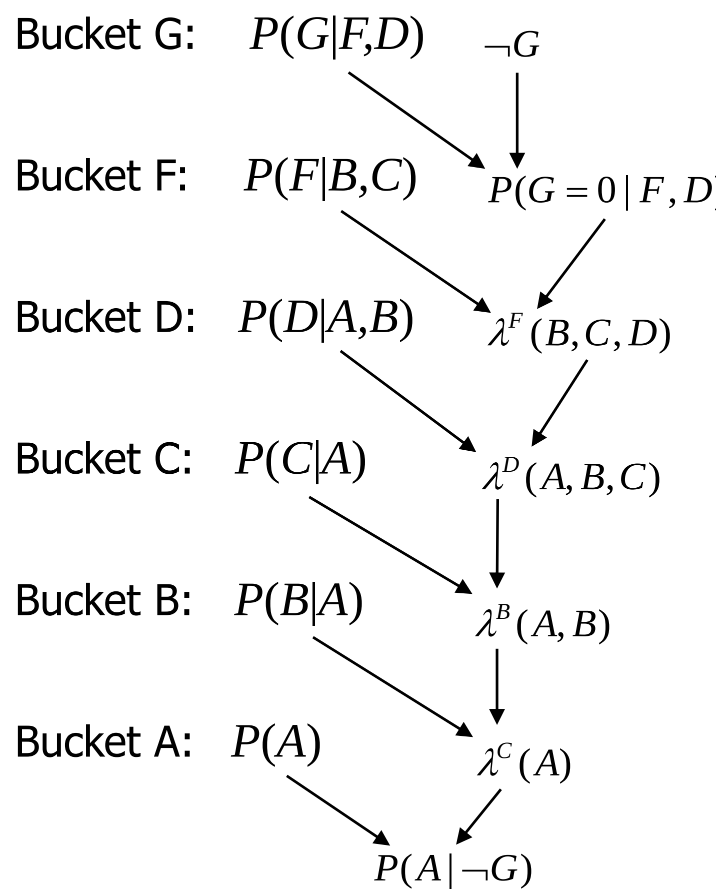
In $Bucket_A$: $\lambda_A = \sum_a P(a)\lambda_C(a)$

$P(\varphi) = \lambda_A$.

For example in $bucket_G$, $\lambda_G(f, d=0) = P(g=1|f)$, because if $D=0$ g must get the value “1”, while $\lambda_G(f, d=1) = P(g=0|f) + P(g=1|f)$. In summary, we have the following.

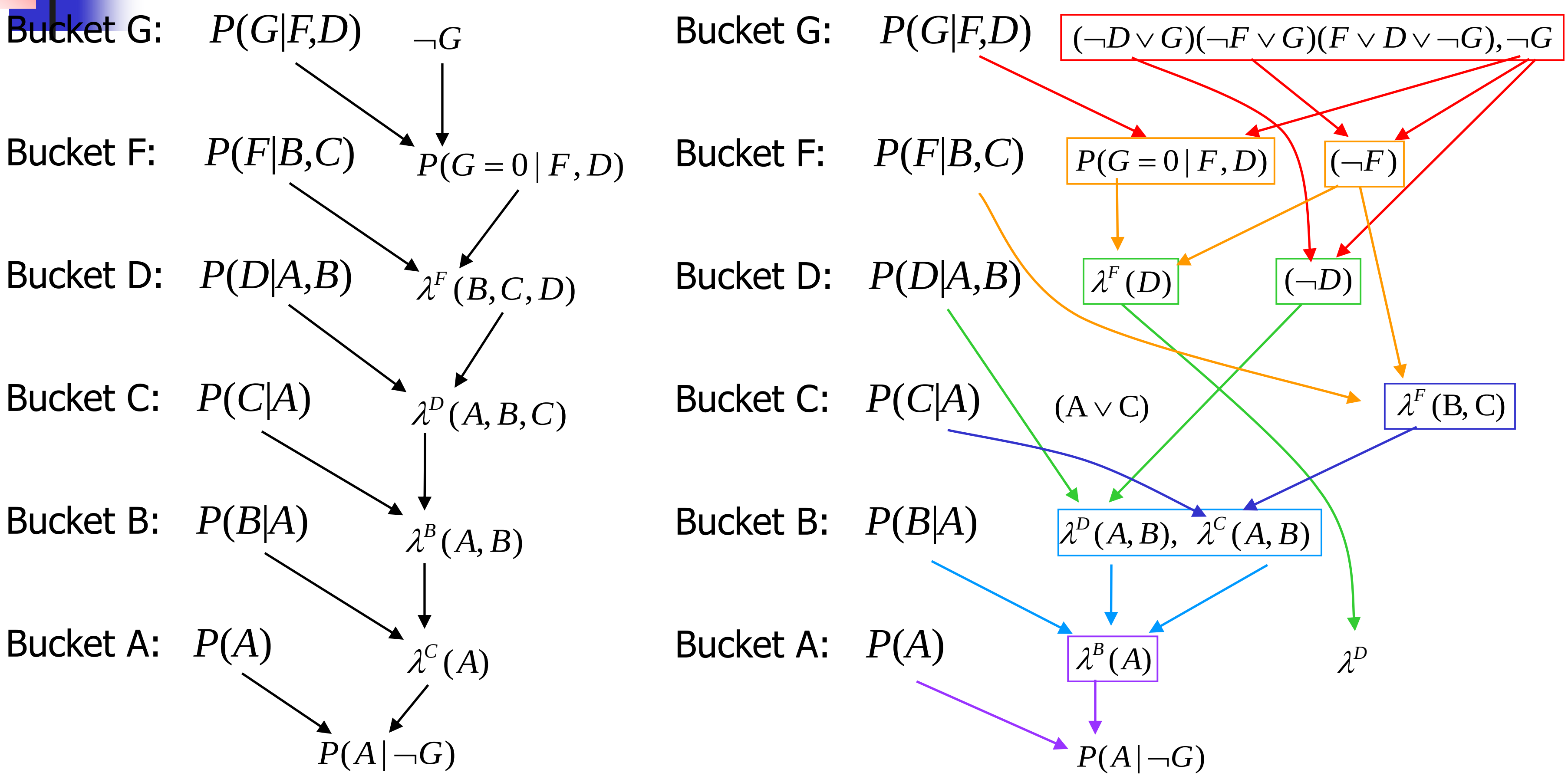
A Hybrid Belief Network





Belief network P(g,f,d,c,b,a)
=P(g|f,d)P(f|c,b)P(d|b,a)P(b|a)P(c|a)P(a)

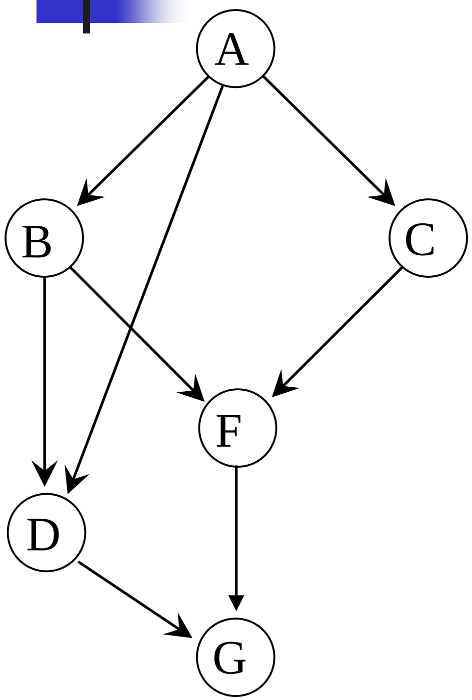
Variable elimination for a mixed network:

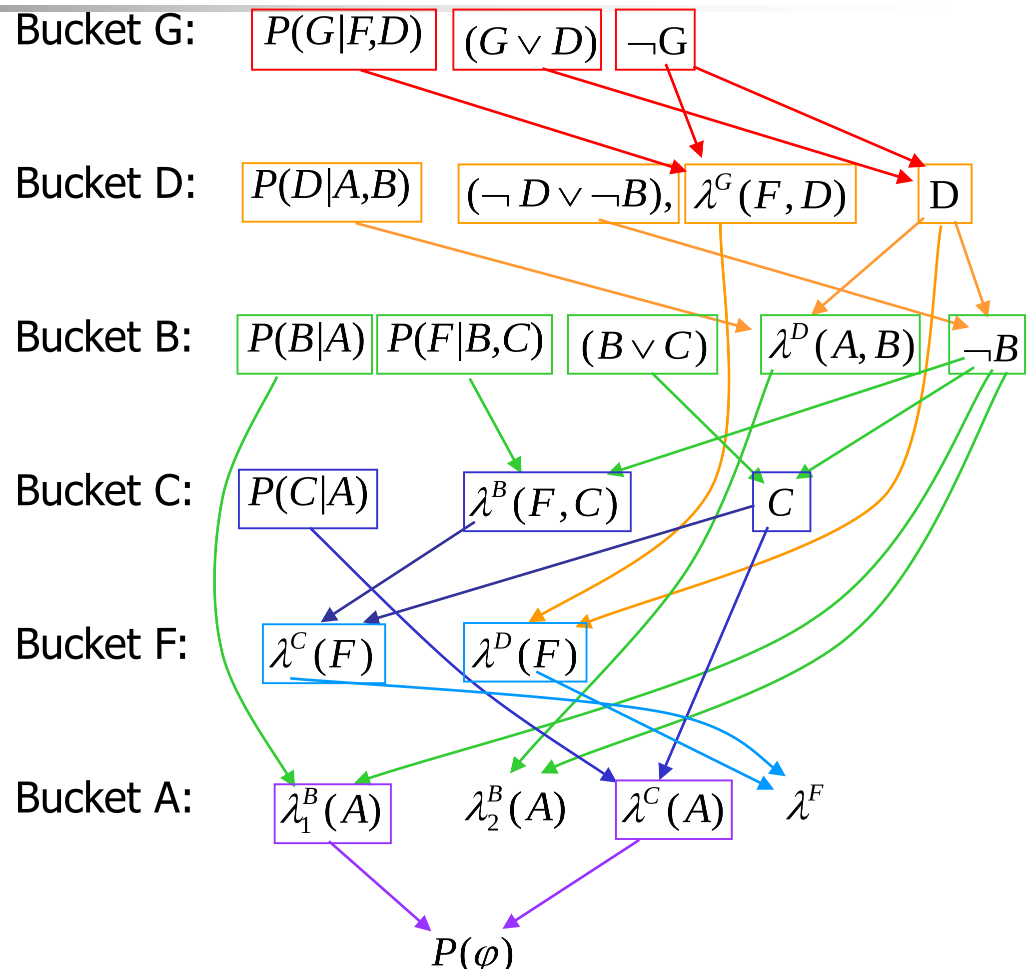


(a) regular Elim-CPE

(b) Elim-CPE-D with clause extraction

Trace of Elim-CPE





Belief network P(g,f,d,c,b,a)
=P(g|f,d)P(f|c,b)P(d|b,a)P(b|a)P(c|a)P(a)

Bucket-elimination example for a mixed network

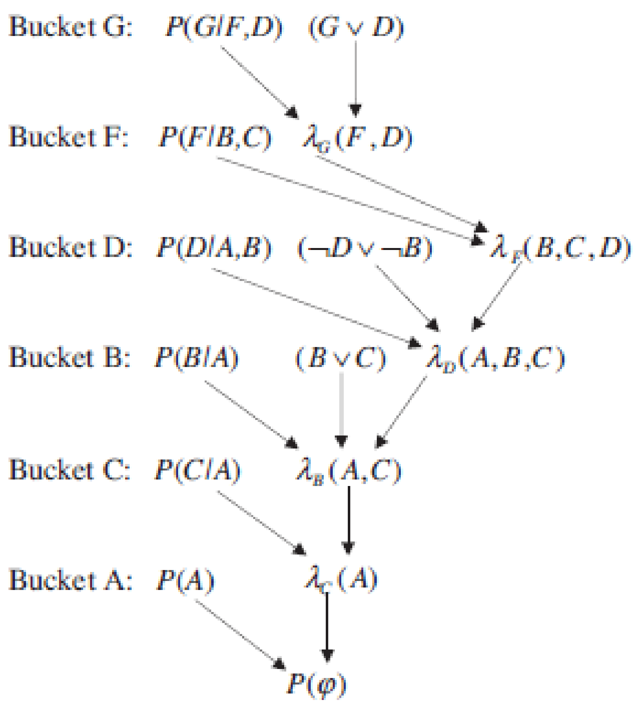


Figure 4.15: Execution of BE-CPE.

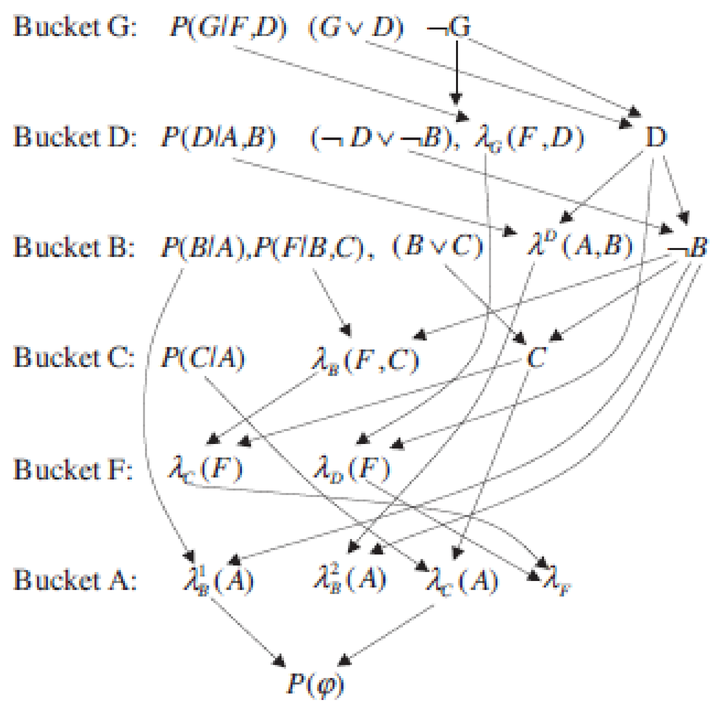


Figure 4.16: Execution of BE-CPE (evidence $\neg G$).

Markov Networks

Definition 2.23 Markov networks. A Markov network is a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{H}, \prod \rangle$ where $\mathbf{H} = \{\psi_1, \ldots, \psi_m\}$ is a set of potential functions where each potential ψ_i is a non-negative real-valued function defined over a scope of variables $\mathcal{S} = \{\mathbf{S}_1, ..., \mathbf{S}_m\}$. $\mathbf{S}_i$. The Markov network represents a global joint distribution over the variables $\mathbf{X}$ given by:

$$P_{\mathcal{M}} = \frac{1}{Z} \prod_{i=1}^{m} \psi_i \quad , \quad Z = \sum_{\mathbf{X}} \prod_{i=1}^{m} \psi_i$$

where the normalizing constant Z is called the partition function.

Dechter, chapter 2

Complexity

Theorem 4.21 Complexity of BE-cpe. *Given a mixed network $M_{\mathcal{B},\varphi}$ having mixed graph is G, with $w^*(d)$ its induced width along ordering d, k the maximum domain size and r be the number of input functions. The time complexity of BE-cpe is $O(r \cdot k^{w^*(d)+1})$ and its space complexity is $O(n \cdot k^{w^*(d)})$. (Prove as an exercise.)*

DEFINITION: An undirected graph $G = (V, E)$ is said to be *chordal* if every cycle of length four or more has a chord, i.e., an edge joining two nonconsecutive vertices.

THEOREM 7: Let G be an undirected graph $G = (V, E)$. The following four conditions are equivalent:

1. G is chordal.

2. The edges of G can be directed acyclically so that every pair of converging arrows emanates from two adjacent vertices.

3. All vertices of G can be deleted by arranging them in separate piles, one for each clique, and then repeatedly applying the following two operations:

   - Delete a vertex that occurs in only one pile.
   - Delete a pile if all its vertices appear in another pile.

4. There is a tree T (called a *join tree*) with the cliques of G as vertices, such that for every vertex v of G, if we remove from T all cliques not containing v, the remaining subtree stays connected. In other words, any two cliques containing v are either adjacent in T or connected by a path made entirely of cliques that contain v.

The running intersection property