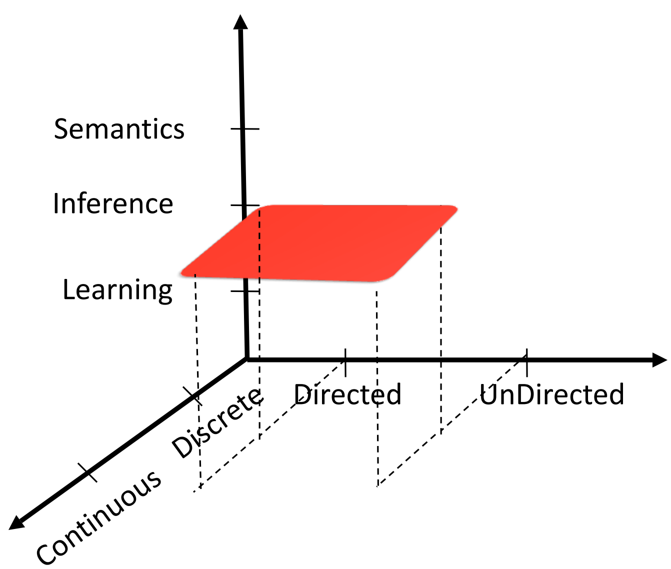
Probabilistic Graphical Models (Cmput 651): Clique Trees

Matthew Brown

{20,24}/10/2008

Reading: Koller-Friedman Ch. 9

Space of topics



Outline

Clique trees and chordal graphs

Variable elimination -> clique tree

Clique tree -> variable elimination]

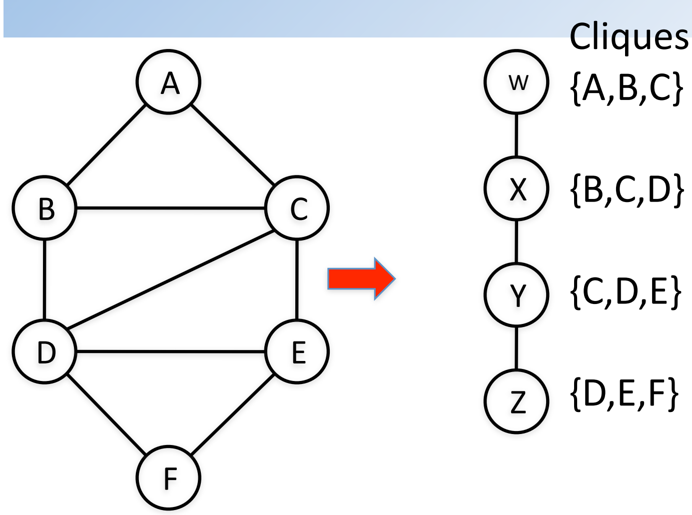
Calibration

Belief update message passing

General queries

Building clique trees

Clique trees



Each edge represents a sepset. The sepset $S_{X,Y}$ between two cliques C_X and C_Y is the intersection of C_X and C_Y ({C,D} in the example above).

Clique trees (Also see KF Definition 4.5.15.)

1st Definition: A tree T is a **clique tree** for a graph H if

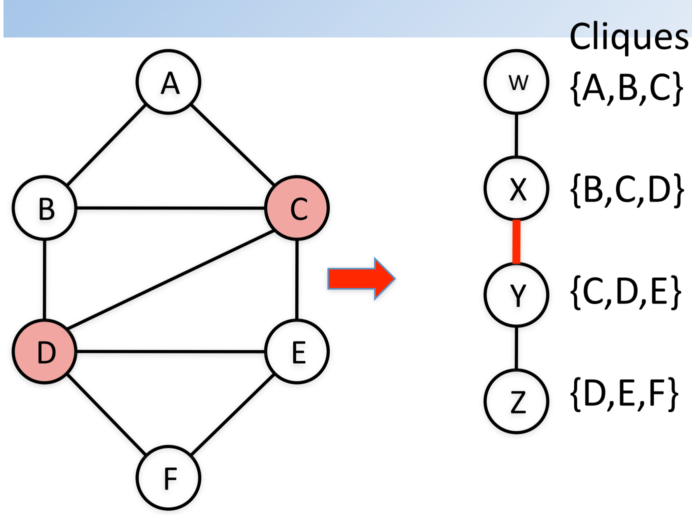
i) each node in T corresponds to a clique in H

ii) each maximal clique in H corresponds to a node in T

iii) each sepset $S_{i,j}$ cuts H into two pieces

(variables on different sides of cut are not connected by a path)

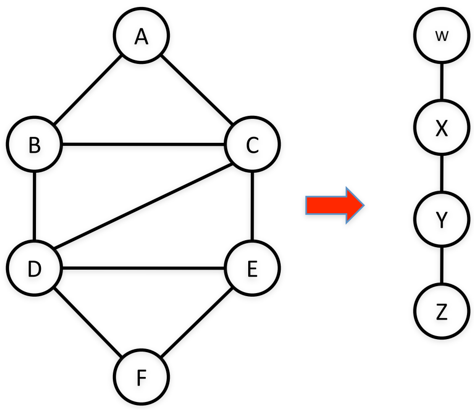
Example of cut by sepset



sepset $S_{X,Y}$ between cliques C_X and C_Y cuts the graph

Chordal graphs and clique trees

Theorem: Every undirected chordal graph has a clique tree. (See KF Theorem 4.5.16)



Outline

Clique trees and chordal graphs

Variable elimination -> clique tree

Clique tree -> variable elimination

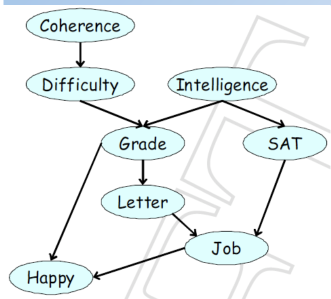
Calibration

Belief update message passing

General queries

Building clique trees

Variable elimination example (See KF Example 8.3.3)



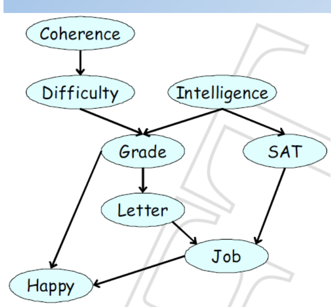
$$\begin{aligned} &P(C,D,I,G,S,L,J,H) \\ =\ & P(C)P(D \mid C)P(I)P(G \mid I,D)P(S \mid I) \\ & P(L \mid G)P(J \mid L,S)P(H \mid G,J) \\ =\ & \phi_C(C)\phi_D(D,C)\phi_I(I)\phi_G(G,I,D)\phi_S(S,I) \\ & \phi_L(L,G)\phi_J(J,L,S)\phi_H(H,G,J). \end{aligned}$$

Summary of variable elimination steps to marginalize out J:

Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D,C)$	C,D	$\tau_1(D)$
2	D	$\phi_G(G,I,D)$, $\tau_1(D)$	G,I,D	$\tau_2(G,I)$
3	I	$\phi_I(I)$, $\phi_S(S,I)$, $\tau_2(G,I)$	G,S,I	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	H,G,J	$\tau_4(G,J)$
5	G	$\tau_4(G,J)$, $\tau_3(G,S)$, $\phi_L(L,G)$	G,J,L,S	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S)$, $\phi_J(J,L,S)$	J,L,S	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	J,L	$\tau_7(J)$

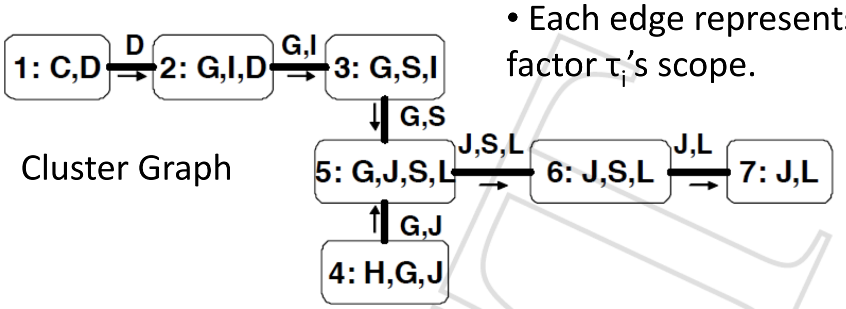
9

Cluster graph for variable elimination (See KF 9.1.1)



Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D,C)$	C,D	$\tau_1(D)$
2	D	$\phi_G(G,I,D)$, $\tau_1(D)$	G,I,D	$\tau_2(G,I)$
3	I	$\phi_I(I)$, $\phi_S(S,I)$, $\tau_2(G,I)$	G,S,I	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	H,G,J	$\tau_4(G,J)$
5	G	$\tau_4(G,J)$, $\tau_3(G,S)$, $\phi_L(L,G)$	G,J,L,S	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S)$, $\phi_J(J,L,S)$	J,L,S	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	J,L	$\tau_7(J)$

- Each node in cluster graph represents a factor ϕ_X's scope.
- Each edge represents a message factor τ_i's scope.



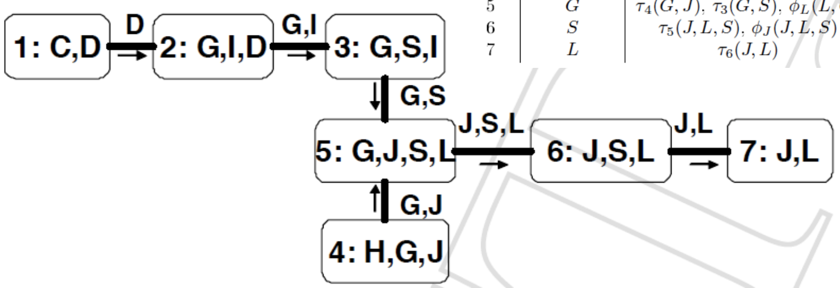
Cluster Graph

10

Cluster graph for variable elimination (See KF 9.1.1)

Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D,C)$	C,D	$\tau_1(D)$
2	D	$\phi_G(G,I,D)$, $\tau_1(D)$	G,I,D	$\tau_2(G,I)$
3	I	$\phi_I(I)$, $\phi_S(S,I)$, $\tau_2(G,I)$	G,S,I	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	H,G,J	$\tau_4(G,J)$
5	G	$\tau_4(G,J)$, $\tau_3(G,S)$, $\phi_L(L,G)$	G,J,L,S	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S)$, $\phi_J(J,L,S)$	J,L,S	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	J,L	$\tau_7(J)$

Cluster Graph



In variable elimination, each node generates a new factor τ (called a message) which is passed to the next node. The next node takes in all τ messages, multiplies them with its own ϕ factor, eliminates ≥1 variable to produce a new τ message, and passes τ on.

Cluster graph (See KF Definition 9.1.1)

Definition: Given set F of factors over *X*,

a **cluster graph** K is an undirected graph such that:

- node i represents a cluster (set) $C_i \subseteq \chi$
- family-preserving: for each factor ϕ in F, exists ≥ 1 node(s) i with $scope(\phi) \subseteq C_i$
- edge C_i-C_j represents sepset $C_i \cap C_j$

Running intersection property

Definition:

Given cluster tree T over factors F,

T has the **running intersection property** means:

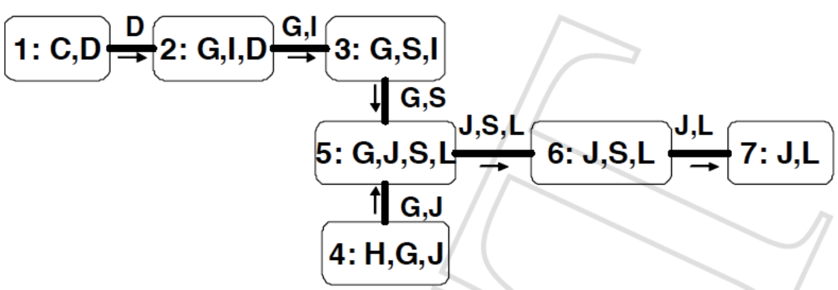
If variable X is in two clusters C_i and C_j,

then X is in every cluster in (unique) path from C_i to C_j.

(See KF Definition 9.1.3.)

Running intersection property

Cluster Graph



Cluster tree satisfies running intersection property.
eg: G is in clusters 2 and 4, as well as the intervening clusters 3 and 5.

Running intersection property and variable elimination

Theorem:

If T is a cluster tree induced by variable elimination, then T satisfies the running intersection property.

(and therefore T is a clique tree, see next slide).

(See KF Theorem 9.1.15 for proof.)

Clique trees (2^{nd} definition) (also see KF 9.1.2)

2^{nd} Definition:

A **clique tree** is

i) a cluster tree

ii) that satisfies running intersection property

Def'n 2 equivalent to earlier def'n 1:

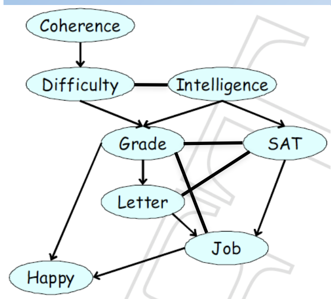
A tree T is a **clique tree** for a graph H if

i) each node in T corresponds to a clique in H

ii) each maximal clique in H corresponds to a node in T

iii) each sepset $S_{i,j}$ cuts H into two pieces

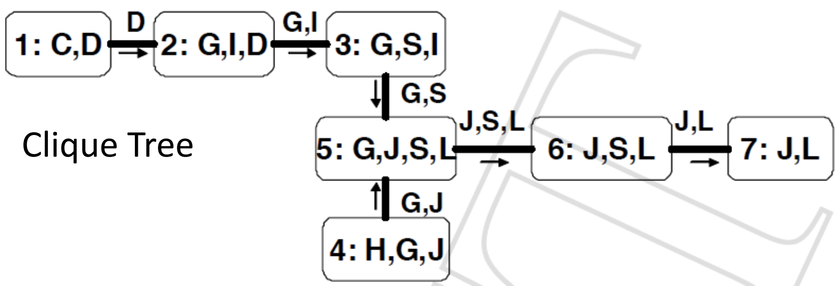
Clique trees: Two equivalent definitions



Note extra edges to make this chordal.

Clique tree satisfies:
def’n 1) based on chordal
def’n 2) based on variable elimination

Clique Tree



Outline

Clique trees and chordal graphs

Variable elimination -> clique tree

Clique tree -> variable elimination

Calibration

Belief update message passing

General queries

Building clique trees

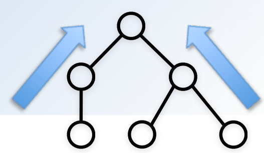
Variable elimination based on clique trees (See KF 9.2.1)

Use clique trees to perform variable elimination via sum product message passing

substantial performance benefits

Clique tree message passing

(see KF 9.2.1.2)



Clique tree sum-product upward pass algorithm:

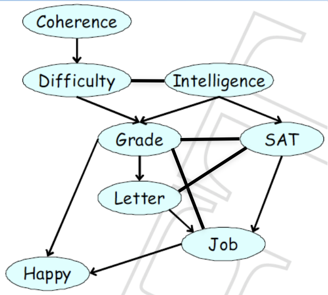
Pass messages up to root clique C_r.

Final clique potential $\beta_r[C_r]$ represents $P_{\mathcal{F}}(C_r) = \sum_{\mathcal{X}-C_r} \prod_{\phi} \phi$

(i.e. marginal probability over root clique)

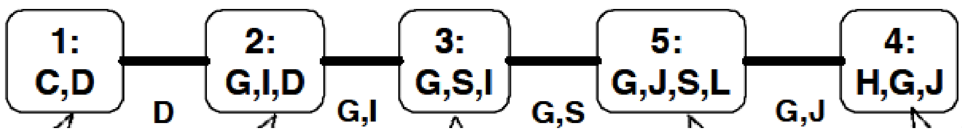
upstream vs. downstream

Clique tree message passing (Also see KF 9.2.1.1, 9.2.1.2)

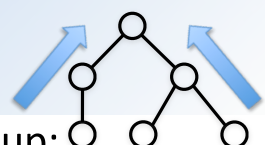


Example task: Given the clique tree below (based on network at left), compute P(J).

Clique Tree



Clique tree message passing
(see KF 9.2.1.1, 9.2.1.2)

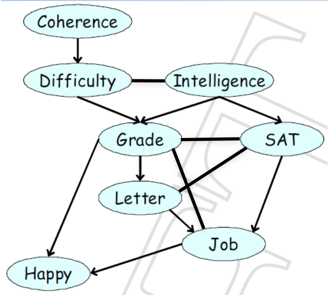


<u>Clique tree sum-product upward pass setup</u>:

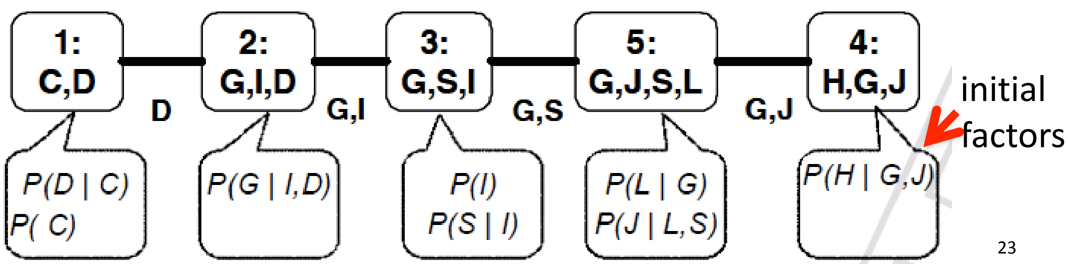
Given clique tree T with cliques $C_1 \ldots C_K$

1. Decides for which variable(s) X you want to compute the marginal (for now, we assume all X are in one clique, but see later slides as well as KF 9.3.4).
2. Select root clique C_r to be a clique containing X.
3. Assign initial factors to cliques such that factor scopes are contained in the cliques.

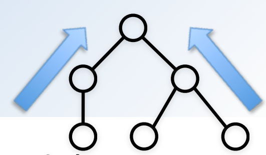
Clique tree message passing (Also see KF 9.2.1.1, 9.2.1.2)



Clique Tree **root clique**



Clique tree message passing
(see KF 9.2.1.2)



<u>Clique tree sum-product upward pass</u> (cont'd):

5. Perform sum-product variable elimination

   start at leaf cliques

   pass messages toward the root.

<u>Message computation</u>:

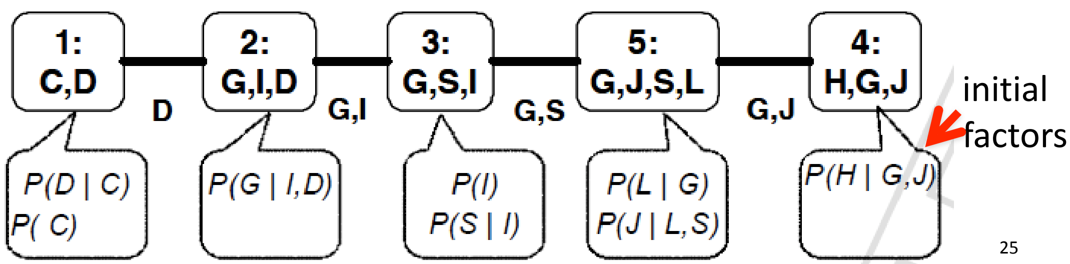
1. factor product of incoming messages (from all neighbours except C_{pr}) and C_i's own initial potential
2. sum out all variables in C_i except sepset (intersection) between C_i and C_{pr}

(C_{pr}=upstream neighbour)

Clique tree message passing (Also see KF 9.2.1.1, 9.2.1.2)

1. In clique C_1, eliminate C: $\sum_C \psi_1[C,D]$; send resulting factor $\delta_{1\to 2}(D)$ as a message to C_2
2. In C_2, define $\beta_2[G,I,D] = \delta_{1\to 2}(D) \cdot \psi_2[G,I,D]$. Eliminate D to produce factor $\delta_{2\to 3}(G,I)$, which is sent to C_3.
3. In C_3, define $\beta_3[G,S,I] = \delta_{2\to 3}(G,I) \cdot \psi_3[G,S,I]$. Eliminate I ro produce $\delta_{3\to 5}(G,S)$, which is sent to 5 for use in Step 5.

Clique Tree

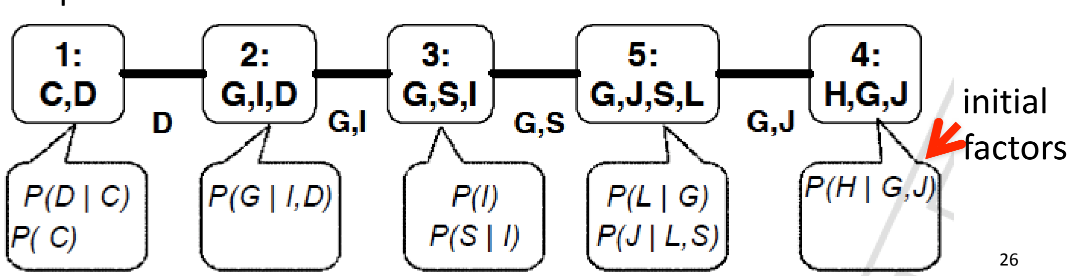


Clique tree message passing (See KF 9.2.1.1, 9.2.1.2)

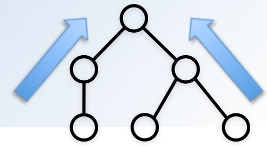
4. In C_4, eliminate H: $\sum_H \psi_4[H,G,J]$, send resulting $\delta_{4\to 5}(G,J)$ to C_5.
5. In C_5, based on messages from Steps 3 and 4, we define

$$\beta_5[G,J,S,L] = \delta_{3\to 5}(G,S) \cdot \delta_{4\to 5}(G,J) \cdot \psi_5[G,J,S,L]$$

Clique Tree



Clique tree message passing
(see KF 9.2.1.1, 9.2.1.2)



Clique tree sum-product upward pass (cont'd):

6. With final clique potential $\beta_r[C_r]$

   if necessary sum out C_r-X

   if necessary normalize

   partition function $Z = \sum_{C_r} \beta_r[C_r]$

   In example, β_5 encodes the joint P(G,J,L,S). To get P(J), sum out G, L, S.

> 1 elimination order (See KF 9.2.1.1)

A clique is **ready** once it's received all of its incoming message ("downstream" messages). Then it can compute its own message and send it "upstream" (toward the root node).

Elimination can proceed in any order that respects readiness requirements.

Outline

Clique trees and chordal graphs

Variable elimination -> clique tree

Clique tree -> variable elimination

Calibration

Belief update message passing

General queries

Building clique trees

Clique tree calibration (Also see KF 9.2.2)

Want all marginals $P(X_j)$:

naive #1: clique tree inference for each X_j

cost: const*num_X_j

naive #2: clique tree inference for each clique C_i

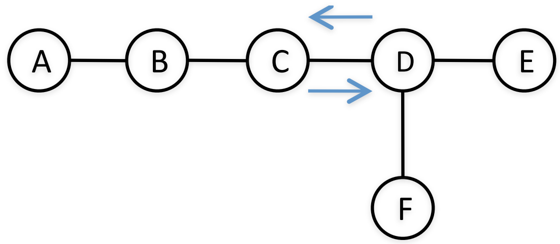
cost: const*num_C_i

better: upward sweep, then backward sweep

2*(num_nodes-1) message computations

cost: 2*const

Only 2 different message per edge



Edge C-D:

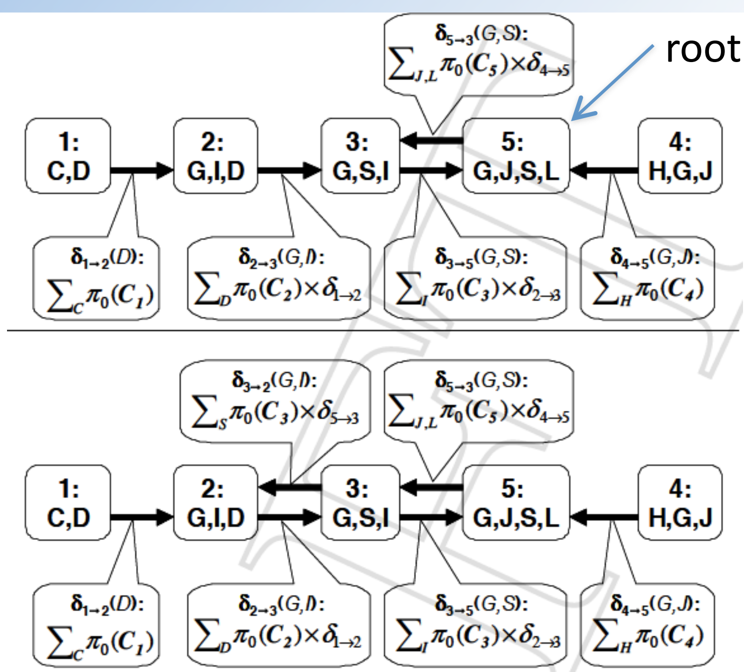
Root is on left or right

Only switch (left<->right) changes incoming messages

=> 2 unique messages possible for each edge

=> 2*(num_nodes – 1) messages total

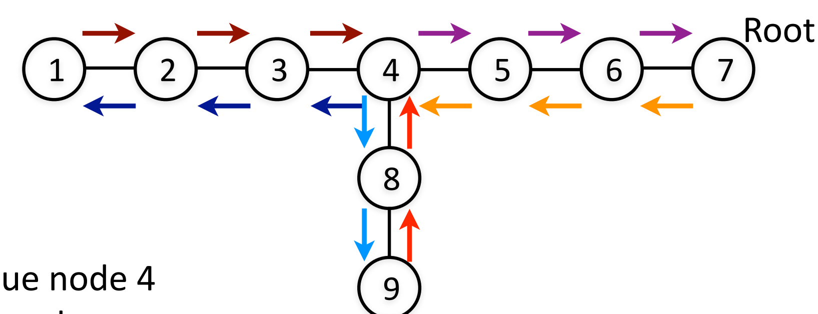
Upward then downward (Also see KF 9.2.2)



Readiness

A clique i is **<u>ready</u>** to compute message to neighbour j after receiving the messages from all it's non-j neighbours.

Messages and message passing



Clique node 4
Upward pass:
- receives 2 (reddish) messages
- sends 1 (purple) message toward root

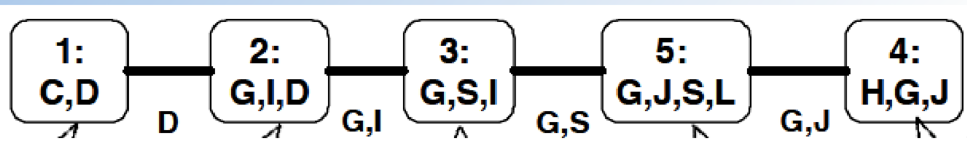
Downward pass:
- receives 1 (orange) message from root branch
- sends 2 <u>different</u> (blue) messages downward

Upward then downward (Also see KF 9.2.2)

After single upward then single downward pass:

- Each clique C_i has all messages
- => final factor $\beta_i[C_i]$ for all i
- => can compute any marginal $P(X_j)$

Calibration



Definition: A clique tree with clique potentials $\beta_i[C_i]$ is **calibrated** if for all neighbouring cliques C_i, C_j

$$\sum_{C_i - S_{i,j}} \beta_i[C_i] = \sum_{C_j - S_{i,j}} \beta_j[C_j]$$

Calibration

<u>Theorem</u>: Sum-product message passing produces a calibrated clique tree.

Proof:

After sum-product (variable elimination!) belief updating

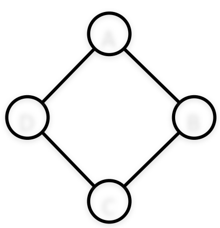
$$\beta_i[C_i] = \sum_{\mathcal{X}-C_i} P_{\mathcal{F}}(\mathcal{X}) = P_{\mathcal{F}}(C_i)$$

(note: $P_{\mathcal{F}}(C_i)$ can be un-normalized)

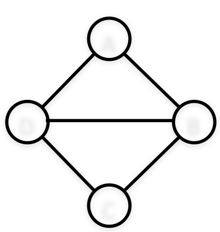
For neighbours C_i and C_j, with $S_{i,j} = C_i \cap C_j$

$$\sum_{C_i - S_{i,j}} \beta_i[C_i] = P_{\mathcal{F}}(S_{i,j}) = \sum_{C_j - S_{i,j}} \beta_j[C_j]$$

Calibrated clique tree = alternative parameterization (also see KF 9.2.3)



$\phi_1[A,B]$			$\phi_2[B,C]$			$\phi_3[C,D]$			$\phi_4[D,A]$		
a^0	b^0	30	b^0	c^0	100	c^0	d^0	1	d^0	a^0	100
a^0	b^1	5	b^0	c^1	1	c^0	d^1	100	d^0	a^1	1
a^1	b^0	1	b^1	c^0	1	c^1	d^0	100	d^1	a^0	1
a^1	b^1	10	b^1	c^1	100	c^1	d^1	1	d^1	a^1	100



Assignment			$\max_C$
a^0	b^0	d^0	600000
a^0	b^0	d^1	300030
a^0	b^1	d^0	5000500
a^0	b^1	d^1	1000
a^1	b^0	d^0	200
a^1	b^0	d^1	1000100
a^1	b^1	d^0	100010
a^1	b^1	d^1	200000

$\beta_1[A,B,D]$

Assignment		$\max_{A,C}$
b^0	d^0	600200
b^0	d^1	1300130
b^1	d^0	5100510
b^1	d^1	201000

$\mu_{1,2}(B,D)$

Assignment			$\max_A$
b^0	c^0	d^0	300100
b^0	c^0	d^1	1300000
b^0	c^1	d^0	300100
b^0	c^1	d^1	130
b^1	c^0	d^0	510
b^1	c^0	d^1	100500
b^1	c^1	d^0	5100000
b^1	c^1	d^1	100500

$\beta_2[B,C,D]$

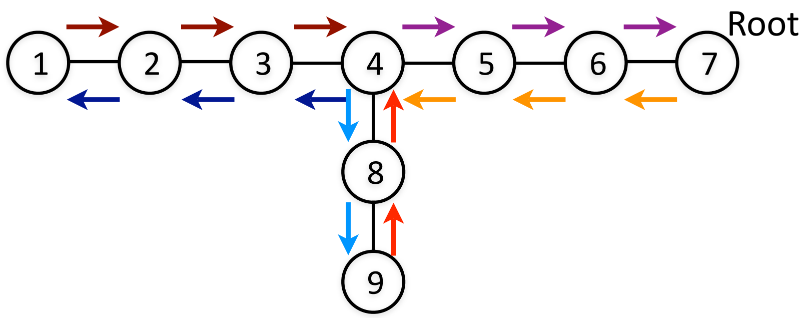
Outline

- Clique trees and chordal graphs
- Variable elimination -> clique tree
- Clique tree -> variable elimination
- Calibration
- **Belief update message passing**
- General queries
- Building clique trees

Belief update message passing (also see KF 9.3.1)

- Sum-product message passing
  - inspired by variable elimination
- Belief update message passing:
  - equivalent to sum-product message passing
  - offers different perspective

Final clique potential



Final clique potential: $\beta_i = \phi_i \cdot \prod_{k \in N_i} \delta_{k \to i}$

ϕ_i = initial potential, $\delta_{k \to i}$ = message, N_i = neighbours of i

Two ways to compute messages (also see KF 9.3.1)

Sum-product $\delta_{i \to j} = \sum_{C_i - S_{i,j}} \phi_i \cdot \underbrace{\prod_{k \in (N_i - \{j\})}}_{} \delta_{k \to i}$

All in-coming messages except $\delta_{j \to i}$

Belief-update $\delta_{i \to j} = \frac{\sum_{C_i - S_{i,j}} \beta_i}{\delta_{j \to i}}$

Final clique potential divided by $\delta_{j \to i}$

Note: $\beta_i = \phi_i \cdot \prod_{k \in N_i} \delta_{k \to i}$

Factor division (also see KF Definition 9.3.1)

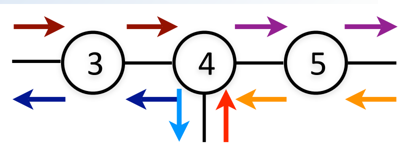
Definition: Given disjoint variable sets X and Y and factors ϕ_1(X,Y) and ϕ_2(Y), **factor division** produces a factor $\psi(\boldsymbol{X},\boldsymbol{Y}) = \frac{\phi_1(\boldsymbol{X},\boldsymbol{Y})}{\phi_2(\boldsymbol{Y})}$

where

0/0 = 0

(x≠0)/0 is undefined

Belief update message passing (also see KF 9.3.1)



Belief update message passing

(aka Lauritzen-Spiegelhalter algorithm):

Iterative process:

Node maintains belief potential

product of initial potential and ALL messages so far

Edge stores previous message (regardless of direction)

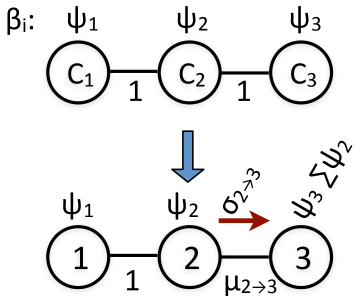
next message is divided by stored message (and then stored)

Keep passing messages until all nodes have all messages

and can compute $\beta_i = \phi_i \cdot \prod_{k \in N_i} \delta_{k \to i}$

Belief update message passing example

(also seek KF example 9.3.3)



$C_1=\{A,B\}$, $C_2=\{B,C\}$, $C_3=\{B,D\}$

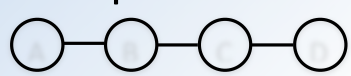
$$\sigma_{2\to 3} = \textstyle\sum_B \psi_2[B,C]$$

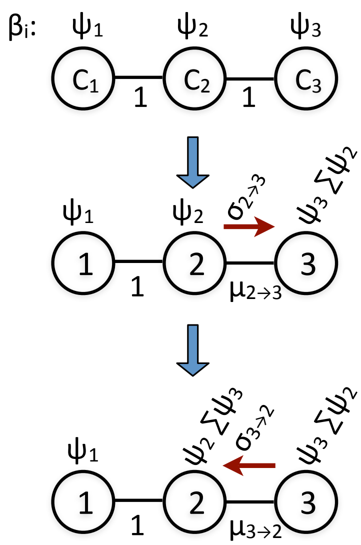
$$\beta_3[C,D] = \psi_3[C,D]\sum_B \psi_2[B,C]$$

$$\mu_{2\to 3} = \sigma_{2\to 3}$$

Belief update message passing example

(also seek KF example 9.3.3)





$C_1=\{A,B\}$, $C_2=\{B,C\}$, $C_3=\{B,D\}$

$$\sigma_{3\to 2}(C) = \sum_D \beta_3[C,D]$$

Update to C_2:

$$\frac{\sigma_{3\to 2}(C)}{\mu_{2,3}(C)} = \frac{\sum_D \beta_3[C,D]}{\mu_{2,3}(C)} = \frac{\sum_D \psi_3[C,D]\mu_{2,3}(C)}{\mu_{2,3}(C)} = \sum_D \psi_3[C,D].$$

$$\mu_{3\to 2} = \sum_D \beta_3[C,D] = \sum_D \left(\psi_3[C,D] \cdot \sum_B \psi_2[B,C]\right)$$

Clique tree measure (also see KF definition 9.2.11)

Definition:

The **clique tree measure** of a calibrated tree T is:

$$\beta_{\mathcal{T}} = \frac{\prod_{C_i \in \mathcal{T}} \beta_i[C_i]}{\prod_{(C_i — C_j) \in \mathcal{T}} \mu_{i,j}(S_{i,j})},$$

where

$$\mu_{i,j} = \sum_{C_i - S_{i,j}} \beta_i[C_i] = \sum_{C_j - S_{i,j}} \beta_j[C_j].$$

Note $\tilde{P}_{\mathcal{F}}(\mathcal{X}) = \beta_{\mathcal{T}}$

Clique tree invariant (also see KF 9.3.2)

Definition:

Clique tree invariance means that

$$\tilde{P}_{\mathcal{F}}(\mathcal{X}) = \frac{\prod_{C_i \in \mathcal{T}} \beta_i[C_i]}{\prod_{(C_i — C_j) \in \mathcal{T}} \mu_{i,j}(S_{i,j})}$$

at each point in belief update calibration.

Proof of clique tree invariance (also see KF Theorem 9.3.4)

$$\tilde{P}_{\mathcal{F}}(\mathcal{X}) = \frac{\prod_{C_i \in \mathcal{T}} \beta_i[C_i]}{\prod_{(C_i - C_j) \in \mathcal{T}} \mu_{i,j}(S_{i,j})}$$

holds during belief update message passing

Proof:

Message $\sigma_{i \to j}$ changes only β_j and $\mu_{i,j}$ to β'_j and $\mu'_{i,j}$.

Want: $\frac{\beta_j}{\mu_{i,j}} = \frac{\beta'_j}{\mu'_{i,j}}$

But belief update message passing was defined so that

$$\beta'_j = \beta_j \frac{\mu'_{i,j}}{\mu_{i,j}}$$

Message passing equivalence (also see KF 9.3.3)

Theorem:

sum-product message passing (variable elimination) is equivalent to

belief-update message passing

(See KF 9.3.3 for proof.)

Outline

Clique trees and chordal graphs

Variable elimination -> clique tree

Clique tree -> variable elimination

Calibration

Belief update message passing

General queries

Building clique trees

General queries on clique trees (also see KF 9.3.4.1)

Conditioning via factor reduction within a clique:

C_1 — C_2 — C_3 C_1 = {A,B} , C_2 = {B,C} , C_3 = {C,D}

Conditioning on D: factor reduction on β_3

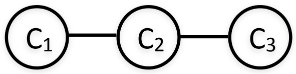
$$\begin{aligned} \tilde{P}_{\mathcal{F}}(C, D = d) &= \beta_3[C, D = d] \\ &= \mathbb{I}\{D = d\} \cdot \beta_3[C, D] \end{aligned}$$

where $\mathbb{I}\{D = d\}$ is an indicator function

(Note: $\tilde{P}_{\mathcal{F}}$ is un-normalized.)

General queries on clique trees (also see KF 9.3.4.1)

Propagating conditioning to entire tree:



$C_1 = \{A,B\}$, $C_2 = \{B,C\}$, $C_3 = \{C,D\}$

W want

$$\tilde{P}_{\mathcal{F}}(A,B,C,D=d) = \mathbb{I}\{D=d\} \cdot \frac{\prod_{C_i \in \mathcal{T}} \beta_i[C_i]}{\prod_{(C_i - C_j) \in \mathcal{T}} \mu_{i,j}(S_{i,j})}$$

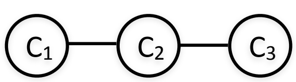
1st step: pass message from C_3 to C_2:

$$\sigma_{3\to 2}(C) = \sum_D \mathbb{I}\{D=d\} \cdot \beta_3[C,D]$$

(remember proper belief updating - go to next slide)

General queries on clique trees (also see KF 9.3.4.1)

Propagating conditioning to entire tree:



$C_1 = \{A,B\}$, $C_2 = \{B,C\}$, $C_3 = \{C,D\}$

Belief updating at C_2 (divide message by sepset potential):

$$\beta_2[B,C] \cdot \frac{\sigma_{3\to 2}(C)}{\mu_{2,3}(C)} = \beta_2[B,C] \cdot \frac{\sum_D \mathbb{I}\{D=d\} \cdot \beta_3[C,D]}{\sum_D \beta_3[C,D]}$$

General queries on clique trees (also see KF 9.3.4.1)

Propagating conditioning to entire tree:

- can propagate conditioning to whole tree
  - or a subset of the tree (if that's all you need)
- cannot easily retract evidence
  - have to store original tree (before conditioning)
- approach not limited just to indicator factors

General queries on clique trees (also see KF 9.3.4.2)

Conditional reasoning across different cliques:

$P(Y|e)$, where $Y \not\subset C_i \; \forall i$

Naive: rebuild clique tree so that $Y \subseteq C_i$ for some i

(Better approach on next slide.)

General queries on clique trees: example
(also see KF 9.3.4.2)

Conditional reasoning across different cliques:

C_1 — C_2 — C_3 $C_1 = \{A,B\}$, $C_2 = \{B,C\}$, $C_3 = \{C,D\}$

Want $\tilde{P}_{\mathcal{F}}(B,D)$

$\tilde{P}_{\mathcal{F}}(B,C,D) = \beta_{\mathcal{T}'}$ $\mathcal{T}' = C_2 — C_3$

$$\begin{aligned}\tilde{P}_{\mathcal{F}}(B,D) &= \sum_C \tilde{P}_{\mathcal{F}}(B,C,D) \\ &= \sum_C \frac{\beta_2[B,C]\beta_3[C,D]}{\mu_{2,3}(C)} \\ &= \sum_C \tilde{P}_{\mathcal{F}}(B \mid C)\tilde{P}_{\mathcal{F}}(C,D)\end{aligned}$$

Do variable elimination on new factors

General queries (also see KF 9.3.4.3)

Multiple queries across different cliques:

Want P(Y|e), where $Y \not\subset C_i$ $\forall i$ for n different Y's

Naive 1: for each Y, rebuild clique tree so that $Y \subseteq C_i$ for some i -> SILLY!

Naive 2: Variable elimination approach (from previous slide) $\binom{n}{2}$ times -> expensive!

Better:

Dynamic programming: start with neighbour cliques, build outward, caching as we go

Outline

Clique trees and chordal graphs

Variable elimination -> clique tree

Clique tree -> variable elimination

Calibration

Belief update message passing

General queries

<u>Building clique trees</u>

Clique trees from variable elimination

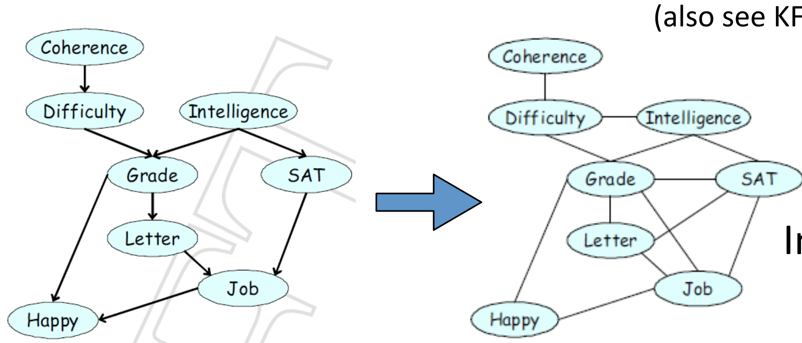
(also see KF 9.4.1, 8.4.2.3)

Definition:

The **<u>induced graph</u>** for a variable elimination

- undirected graph
- edge between all pairs of variables that appear in some intermediate factor ϕ

(also see KF definition 8.4.3)

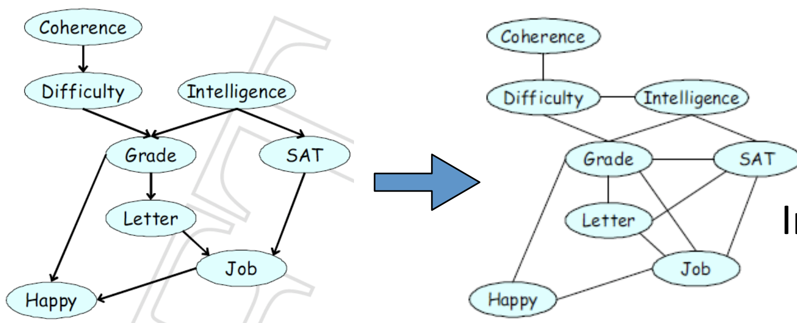


Induced graph

Clique trees from variable elimination
(also see KF 9.4.1, 8.4.2.3)

Theorem: Induced graph is chordal.

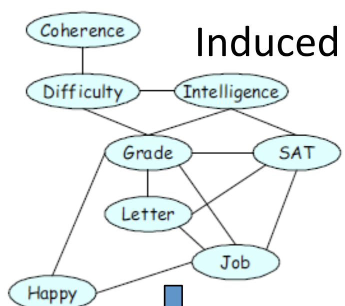
(For proof see KF Theorem 8.4.7).



Induced graph

Clique trees from variable elimination
(also see KF 9.4.1)

Build clique tree from induced graph

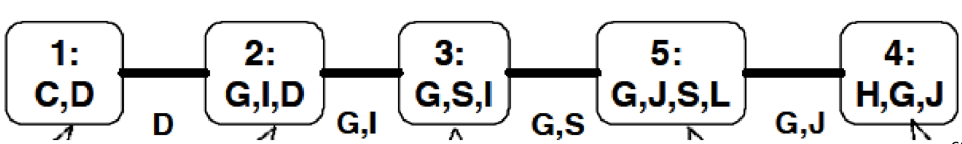


Induced graph

Typically use only maximal cliques in clique tree.

Note: we saw this earlier

Clique Tree



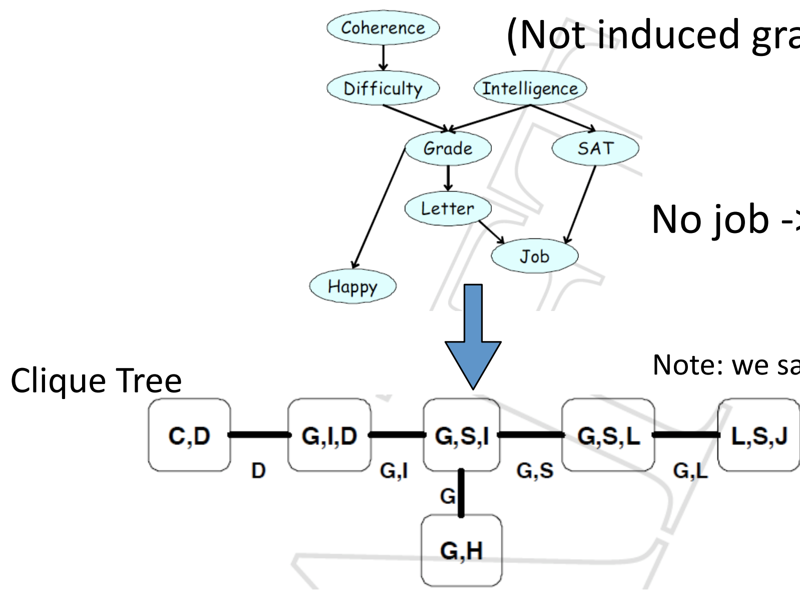
Clique trees from variable elimination
(also see KF 9.4.1)

Build clique tree from induced graph - example 2

(Not induced graph)

No job -> happy edge

Note: we saw this earlier

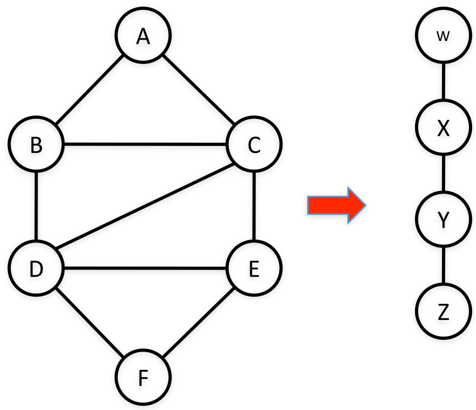


Clique trees from variable elimination
(also see KF 9.4.1)

Building clique trees from induced graphs is formally well-grounded (see KF 9.4.1, 8.4.2.3).

Clique trees from chordal graphs (also see KF 9.4.2)

Theorem: Every chordal graph has a clique tree. (See KF Theorem 4.5.16)



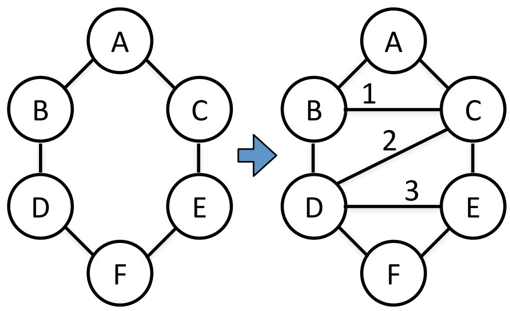
Clique trees from chordal graphs (also see KF 9.4.2)

Given factors $\mathcal{F}$ which factorize based on $\mathcal{H}_{\mathcal{F}}$

1. build chordal graph $\mathcal{H}^*$ from $\mathcal{H}_{\mathcal{F}}$ (triangulation)
2. find maximal cliques in $\mathcal{H}^*$
3. build clique tree $\mathcal{T}$ from $\mathcal{H}^*$'s maximal cliques

Triangulation (also see KF 9.4.2)

- finding minimum triangulation is NP-hard
  - minimum triangulation = one in which largest clique has min size
- exist exact algorithms
  - exponential in size of largest clique
- heuristic algorithms typically used



Heuristic triangulation
(also see KF 9.4.2, 8.4.3.2)

Recall: induced graph from variable elimination is chordal

So, do variable elimination to triangulate

- do not do actual sum and product computations
- just keep track of intermediate factors' scopes
- connect all sets of nodes within a given scope
- see example two slides below

Variable elimination requires an elimination order

- see next slide

Heuristic elimination ordering algorithm
(also see KF 9.4.2, 8.4.3.2)

Want an ordering that produces a small chordal graph.

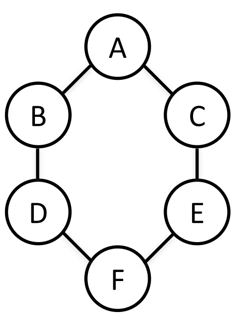
Define cost function (also see KF 8.4.3.2)

- eg: cost(node_X) = #neighbours

Use greedy ordering method (also see KF Figure 8.17)

```
all nodes start unmarked
for k=1...#nodes
    select unmarked node X with min cost
    π(X) = k
    mark X
return ordering π
```


Heuristic triangulation via variable elimination example (also see KF 9.4.2, 8.4.3.2)



Factors:

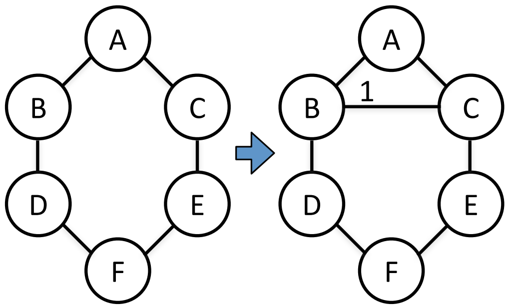
$\phi_1(A,B)$, $\phi_2(A,C)$, $\phi_3(B,D)$, $\phi_4(C,E)$, $\phi_5(D,F)$, $\phi_6(E,F)$

$$P(A,B,C,D,E,F) = \frac{1}{Z} \prod_{i=1\ldots6} \phi_i$$

Ordering: ABCDEF

Example continues over next six slides.

Heuristic triangulation via variable elimination example (cont'd) (also see KF 9.4.2, 8.4.3.2)



Factors:

$\phi_1(A,B)$, $\phi_2(A,C)$, $\phi_3(B,D)$, $\phi_4(C,E)$, $\phi_5(D,F)$, $\phi_6(E,F)$

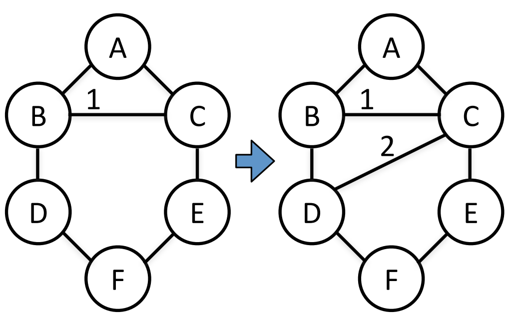
$$P(A,B,C,D,E,F) = \frac{1}{Z} \prod_{i=1\ldots6} \phi_i$$

Ordering: ABCDEF

Eliminate A: A,B,C in ψ_1's scope -> add edge B-C

$$\begin{aligned}\sum_A P(A,B,C,D,E,F) &= \phi_3(B,D)\phi_4(C,E)\phi_5(D,F)\phi_6(E,F)\sum_A \phi_1(A,B)\phi_2(A,C) \\ &= \phi_3(B,D)\phi_4(C,E)\phi_5(D,F)\phi_6(E,F)\sum_A \psi_1(A,B,C) \\ &= \phi_3(B,D)\phi_4(C,E)\phi_5(D,F)\phi_6(E,F)\tau_1(B,C)\end{aligned}$$

Heuristic triangulation via variable elimination example (cont'd) (also see KF 9.4.2, 8.4.3.2)



Factors:

$\phi_1(A,B)$, $\phi_2(A,C)$, $\phi_3(B,D)$, $\phi_4(C,E)$, $\phi_5(D,F)$, $\phi_6(E,F)$

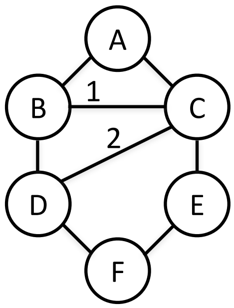
$$P(A,B,C,D,E,F) = \frac{1}{Z} \prod_{i=1\ldots6} \phi_i$$

Ordering: ABCDEF

Eliminate B: B,C,D in ψ_2's scope -> add edge C-D

$$\begin{aligned}\sum_B P(B,C,D,E,F) &= \phi_4(C,E)\phi_5(D,F)\phi_6(E,F)\sum_B \phi_3(B,D)\tau_1(B,C) \\ &= \phi_4(C,E)\phi_5(D,F)\phi_6(E,F)\sum_B \psi_2(B,C,D) \\ &= \phi_4(C,E)\phi_5(D,F)\phi_6(E,F)\tau_2(C,D)\end{aligned}$$

Heuristic triangulation via variable elimination example (cont'd) (also see KF 9.4.2, 8.4.3.2)



Factors:

$\phi_1(A,B)$, $\phi_2(A,C)$, $\phi_3(B,D)$, $\phi_4(C,E)$, $\phi_5(D,F)$, $\phi_6(E,F)$

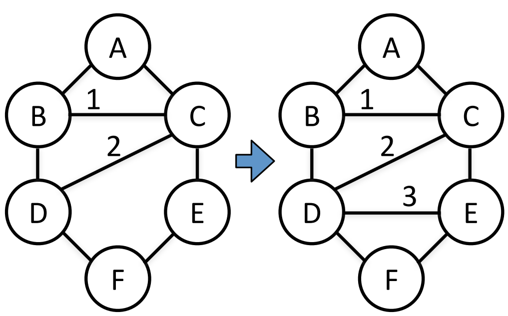
$$P(A,B,C,D,E,F) = \frac{1}{Z}\prod_{i=1\ldots6}\phi_i$$

Ordering: ABCDEF

Eliminate C: C,D,E in ψ_3's scope already connected

$$\sum_C P(C,D,E,F) = \phi_5(D,F)\phi_6(E,F)\sum_C \phi_4(C,E)\tau_2(C,D)$$
$$= \phi_5(D,F)\phi_6(E,F)\sum_C \psi_3(C,D,E)$$
$$= \phi_5(D,F)\phi_6(E,F)\tau_3(D,E)$$

Heuristic triangulation via variable elimination example (cont'd) (also see KF 9.4.2, 8.4.3.2)



Factors:

$\phi_1(A,B)$, $\phi_2(A,C)$, $\phi_3(B,D)$, $\phi_4(C,E)$, $\phi_5(D,F)$, $\phi_6(E,F)$

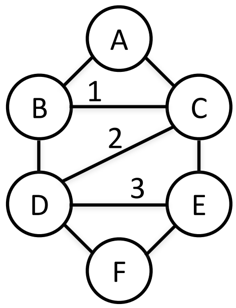
$$P(A,B,C,D,E,F) = \frac{1}{Z}\prod_{i=1\ldots6}\phi_i$$

Ordering: ABCDEF

Eliminate D: D,E,F in ψ_4's scope -> add edge D-E

$$\sum_D P(D,E,F) = \phi_6(E,F)\sum_D \phi_5(D,F)\tau_3(D,E)$$
$$= \phi_6(E,F)\sum_D \psi_4(D,E,F)$$
$$= \phi_6(E,F)\tau_4(E,F)$$

Heuristic triangulation via variable elimination example (cont'd) (also see KF 9.4.2, 8.4.3.2)



Factors:

$\phi_1(A,B)$, $\phi_2(A,C)$, $\phi_3(B,D)$, $\phi_4(C,E)$, $\phi_5(D,F)$, $\phi_6(E,F)$

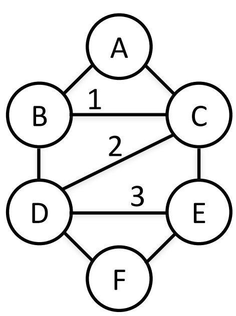
$$P(A,B,C,D,E,F) = \frac{1}{Z} \prod_{i=1\ldots6} \phi_i$$

Ordering: ABCDEF

Eliminate E: E,F in ψ_5's scope already connected

$$\sum_E P(E,F) = \sum_E \phi_6(E,F)\tau_4(E,F)$$
$$= \sum_E \psi_5(E,F)$$
$$= \tau_5(F)$$

Heuristic triangulation via variable elimination example (cont'd) (also see KF 9.4.2, 8.4.3.2)



Factors:

$\phi_1(A,B)$, $\phi_2(A,C)$, $\phi_3(B,D)$, $\phi_4(C,E)$, $\phi_5(D,F)$, $\phi_6(E,F)$

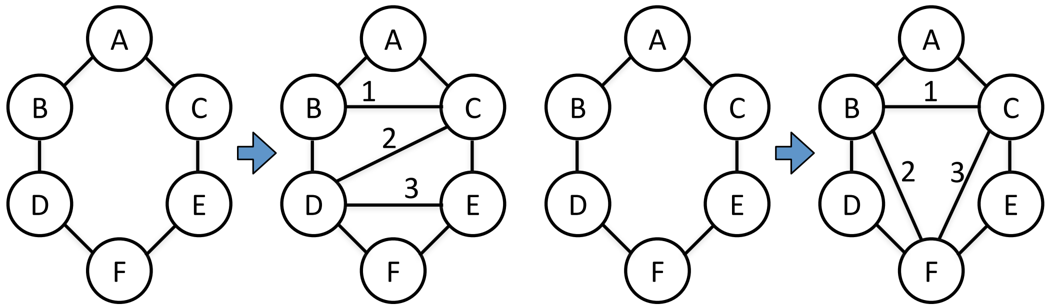
$$P(A,B,C,D,E,F) = \frac{1}{Z} \prod_{i=1\ldots6} \phi_i$$

Ordering: ABCDEF

Eliminate F: F = last variable, so we stop

$$\sum_F P(F) = \sum_F \tau_5(F)$$

Heuristic triangulation via variable elimination
(also see KF 9.4.2, 8.4.3.2)



Ordering: ABCDEF   Ordering: ADEBCF

Find maximal cliques
(also see KF 9.4.2, 8.4.3.2)

Task: given chordal graph, find maximal cliques

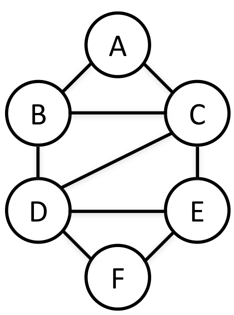
- NP-hard for general graphs
- BUT easy with chordal graphs

Find maximal cliques
(also see KF 9.4.2, 8.4.3.1)

Maximum cardinality search (also see KF Fig 8.16):

```
clique_list = {}, current_clique = {}
while still unmarked nodes:
  select (unmarked) X with max # marked neighbours
  if X fully connected to current_clique:
    add X to current_clique
  else:
    add current_clique to clique_list
    current_clique = {X and X's marked neighbours}
  mark X
return clique_list
```

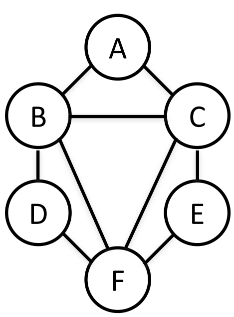

Find maximal cliques
(also see KF 9.4.2, 8.4.3.2)



Nodes: D B C A E F

current_clique: {D} {BD} {BCD} {ABC} {CDE} {DEF}

addition to clique_list:



Nodes: D B F C A E

current_clique: {D} {BD} {BDF} {BCF} {ABC} {CEF}

addition to clique_list:

Find clique tree edges

(also see KF 9.4.2)

Task: given cliques, connect them to build clique tree

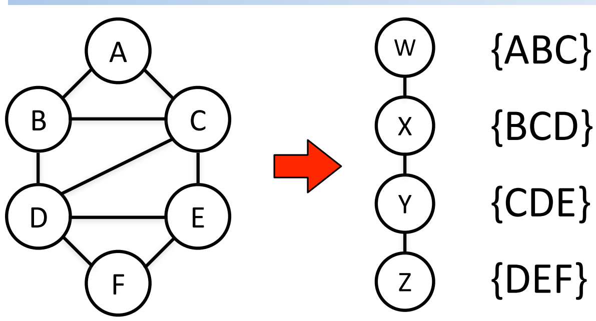
Use maximum spanning tree algorithm:

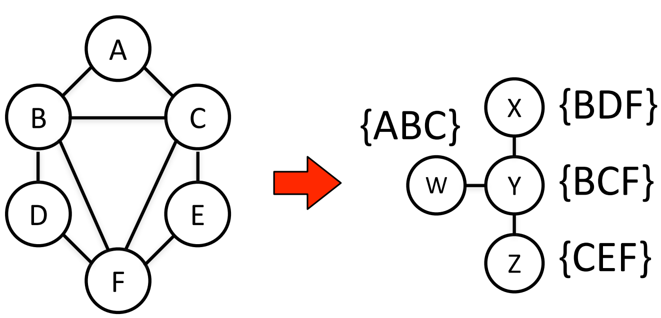
- start with complete clique graph
- assign edge weights: $|C_i \cap C_j|$
- remove edges to produce tree with max sum of edge weights

$\Rightarrow$ edges represent sepsets

Build clique tree

(also see KF 9.4.2, 8.4.3.2)





Clique trees from chordal graphs (also see KF 9.4.2)

Summary:

Given factors $\mathcal{F}$ which factorize based on $\mathcal{H}_{\mathcal{F}}$

1. build chordal graph $\mathcal{H}^*$ from $\mathcal{H}_{\mathcal{F}}$ (triangulation)
2. find maximal cliques in $\mathcal{H}^*$
3. build clique tree $\mathcal{T}$ from $\mathcal{H}^*$'s maximal cliques

Do inference on clique tree.