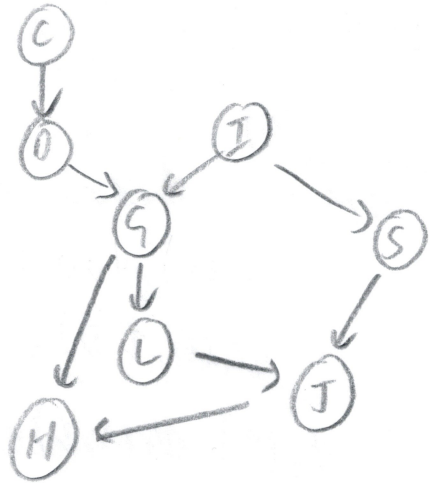
- (*) This is a key example to mastered which forms basis of ch 9 and 10.



Ex 9.1

$P(C,D,I,G,S,J,L,H) =$

$P(C)P(D|C)P(I)P(G|I,D)P(S|I)$
$P(L|G)P(J|L,S)P(H|G,J)$

- Apply VE to compute $P(J)$ using $\zeta = (C,D,I,H,G,S,L)$

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

Elim C: compute:- $\psi_1(C,D) = \phi_C(C)\cdot\phi_D(D,C)$

$\tau_1(D) = \sum_C \psi_1(C,D)$

Elim D: compute:-

$\psi_2(G,I,D) = \phi_G(G,I,D)\tau_1(D)$

$\tau_2(G,I) = \sum_D \psi_2(G,I,D)$

Elim I: $\psi_3(G,I,S) = \phi_I(I)\phi_S(S,I)\tau_2(G,I)$

$\tau_3(G,S) = \sum_I \psi_3(G,I,S)$

Elim H: $\psi_4(H,G,J) = \phi_H(H,G,J)$

$\tau_4(G,J) = \sum_H \psi_4(H,G,J)$

Elim G $\psi_5(G,J,L,S) = \phi_L(L,G)\tau_3(G,S)\tau_4(G,J)$

$\tau_5(J,L,S) = \sum_G \psi_5(G,J,L,S)$

Elim S $\psi_6(J,L,S) = \phi_J(J,L,S)\tau_5(J,L,S)$

$\tau_6(J,L) = \sum_S \psi_6(J,L,S)$

Elim L: $\psi_7(J,L) = \tau_6(J,L)$

$$\tau_7(J) = \sum_L \psi_7(J,L).$$

Algorithmic summary

Step	Var elim.	Factors used	Vars involv.	New fact
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	$\tau_7(J)$

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

C,D → C_1

G,I,D → C_2

G,I,S → C_3

G,J,L,S → C_5

G,H,J → C_4

Assignment

$P(C)P(D|C) = \phi_C(C)\phi_D(D,C) = C_1$

$P(G|I,D) = \phi_G(G,I,D) = C_2$

$P(I)P(S|I) = \phi_I(I)\phi_S(S,I) = C_3$

$P(L|G)P(J|L,S) = \phi_L(L,G)\phi_J(J,L,S) = C_5$

$P(H|G,J) = \phi_H(H,G,J) = C_4$

init. pot.

$\psi_1(C_1) = \phi_C(C)\phi_D(D,C)$

$\psi_5(C_5) = \phi_L(L,G)\phi_J(J,L,S)$ ✓

$P(J)$

- select any root clique, a clique with J

  i.e. C_4 or C_5

1. C_1: eliminate C by $\sum_C \psi_1(C,D)$ $\quad \delta_{1\to 2}(D) = \sum_C \psi_1(C,D)$

   $\delta_{1\to 2}(D)$ to C_2

2. C_2 elim D $\quad \beta_2(G,I,D) = \delta_{1\to 2}\cdot\psi_2(G,I,D)$

   $\delta_{2\to 3}(G,I) = \sum_D \beta_2(G,I,D)$ to C_3

3. C_3 elim I

   $\beta_3(G,S,I) = \delta_{2\to 3}(G,I)\cdot\psi_3(G,S,I)$

   $\delta_{3\to 5}(G,S) = \sum_I \beta_3(G,S,I)$ to C_5

C_4: Elim H:

$$\delta_{4\to 5}(G,J) = \sum_H \psi_4(H,G,J) \quad \text{to } C_5$$

C_5: $\beta_5(G,J,S,L) = \delta_{3\to 5}(G,S)\,\delta_{4\to 5}(G,J)\,\psi_5(G,J,S,L)$

$\beta_5(G,J,S,L)$

$$P(J) = \sum_G \sum_S \sum_L \beta_5(G,J,S,L)$$

$\alpha(\phi) = (C, D)$

Koller & Friedman (2009)

13/06/2020

Ch 10 - Exact inference - Clique trees

- chapter not.

- set of factors Φ
- set of variables X which factors defined over.
- each factor ϕ_i has scope $\underline{X}_i$ (takes arguments over)
- this set of factor defines an unnormalised measure

$$\tilde{P}_{\Phi}(X) = \prod_{\phi_i \in \Phi} \phi_i(\underline{X}_i)$$

- Bayesian nets (no evidence)
- factors - CPDs
- measure $\tilde{P}_{\underline{\Phi}}$ is a normalised distribution.
- Bayesian nets (evidence $\underline{E} = \underline{e}$)
- factors are CPDs restricted to $\underline{e}$
- measure $\hat{P}_{\underline{\Phi}}(X) = P_B(X, \underline{e})$
- Gibbs distn (with/without evidence)
- factors - (restricted) potentials
- unnormalised Gibbs measure $\tilde{P}_{\underline{\Phi}}$

(*) All ops that can be performed on a normalised measure can also (a)
be perf. on an unnormalised measure.

- marginalise $\tilde{P}_{\underline{\Phi}}$ on a subset of variables by summing out
- also can consider condit measure $\tilde{P}_{\underline{\Phi}}(\underline{X} | \underline{Y}) = \tilde{P}_{\underline{\Phi}}(\underline{X}, \underline{Y}) / \tilde{P}_{\underline{\Phi}}(\underline{Y})$
- same as $P_{\underline{\Phi}}(\underline{X} | \underline{Y})$

10.1 - Variable Elim, Clique Trees.

- VE algo - manipulate factors.
- each step:- 1) create a factor ψ_i by multiplying existing factors
  2) Eliminate a variable in ψ_i to generate a new factor τ_i
  (used to create another).

(*) AH view:

- A factor ψ_i is considered as a computational data structure
- It takes messages "τ_j" generated by other factors ψ_j and generates a message "τ_i" that is used by another factor ψ_k.

(*) Fig 10.1 - cluster tree

(*) 10.1.1. - cluste graphs

- cluste graph - data structe - graphical flowchart - factor man process
- node - cluste - graph - subset of variables
- undirected - edges - connect - clusters - scopes - non-empty intesectia

---

D 10.1. - cluster graph, family pres, sepset

- A cluster graph U for a set of factors Φ ove $\mathcal{X}$ is an undirected graph, each of whose nodes i is associated with a subset $C_i \subseteq \mathcal{X}$.
- A cluster graph must be family-preserving -
  - each factor $\phi \in \Phi$ must associated with a cluster C_i, denoted $\alpha(\phi)$, such that $Scope[\phi] \subseteq C_i$
- Each edge between a pair of clusters C_i and C_j is associated with a sepset $S_{i,j} \subseteq (C_i \cap C_j)$

- VE algo → cluster graph
(*) Cluste for each factor ψ_i used in computation; associated with set of variables $C_i = Scope[\psi_i]$
(*) Draw an edge between two clusters C_i and C_j if message τ_i produced by eliminating a variable in ψ_i is used in the computation of τ_j.

(*) 10.1.2. - clique trees

- VE algo - each intem factor used at most once
  - when ϕ_i is ~~use~~ used in S-PE algo to create ψ_j, it is removed from the set of factors Φ, can no longer be used.

(*) cluster graph induced by VE algo → tree.

(*) VE-algo defines directionality via message flow

(*) All messages flow → root (singe cluster - final result - comp)

(*) CS convention:- root of tree is up
- messages sent upward to root.

(*) upstream - downstream

D 10.2 (Running intersection property)

- Let T be a cluster tree over set of factors Φ
- V_T - vertices of T ; E_T - edges of T
- T has running intersection property if; whenever there is a variable such that $X \in C_i$ and $X \in C_j$; then X is also in every cluster ~~(or)~~ in the (unique) path in T between C_i and C_j.

(R): Running intersection property $\Rightarrow S_{i,j} = (C_i \cap C_j)$

- Running intersection property must holds for cluster tree induced by VE algo; a variable appears in every factor from moment it is introduced (by mult. in a factor (2) that mentions it) until it is summed out

---

T. 10.1

- Let T be a cluster tree induced by VE algo over some set of factors Φ. Then T satisfies the running intersection property.

P 10.1

- Let T be a cluster tree induced by a variable elimination algo over some set of factors Φ.
- Let C_i and C_j be two neighbouring clusters such that C_i passes the message τ_i to C_j
- Then the scope of the messages is precisely $C_i \cap C_j$

10.3 (cluster tree / running int / clique tree)

- Let Φ be a set of factors over $\mathcal{X}$
- A cluster tree over Φ that satisfies the running intersection property is called a clique tree / junction tree
- In the case of a clique tree, the clusters are also called cliques.

(R): T is a clique tree for Φ iff T is a clique tree for a chordal graph containing H_Φ

(10.3) (10.4.17)

- properties true iff clique-tree data structure admits VE by passing messages over the tree

(R): Running intersection property $\Rightarrow$ independence statement (10.4.17 clique trees)

(I): Let T be a cluster tree over Φ
H_Φ be an undirected graph over $\mathcal{X}$.
for any sepset $S_{i,j}$, let $W_{<(i,j)}$ be set of all variables in scope of clusters on C_i side of tree (~)

$W_{<(j,i)}$ be ——— || ——— C_j ——— || ———

10.2

- T satisfies running intersection property iff for every sepset $S_{i,j}$ we have (~) that $W_{<(i,j)}$ and $W_{<(j,i)}$ are separated in H_Φ given $S_{i,j}$

(R)(*)

- We can show equivalence of both definitions (10.3 and 4.17)

(*) show that running intes. prop of tree T implies that each node in T corresponds to a clique in a chordal graph H' containing H, and that each maximal clique in H' is represented in T. (?)

(*) Accompanying diagrams (collated) for clarity
- They are crucial for an understanding of what is going on.

Fig 9.8 (Koller)
(BN for Student)

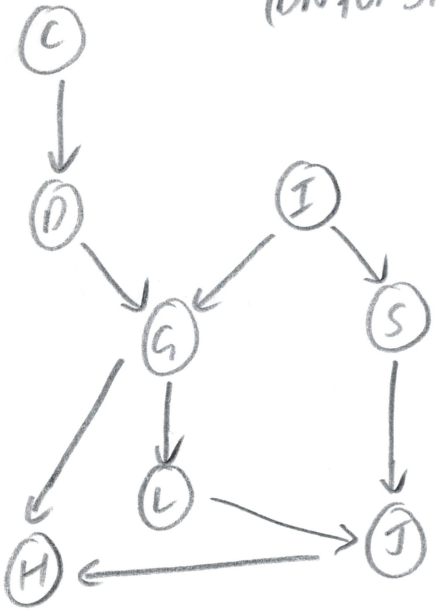
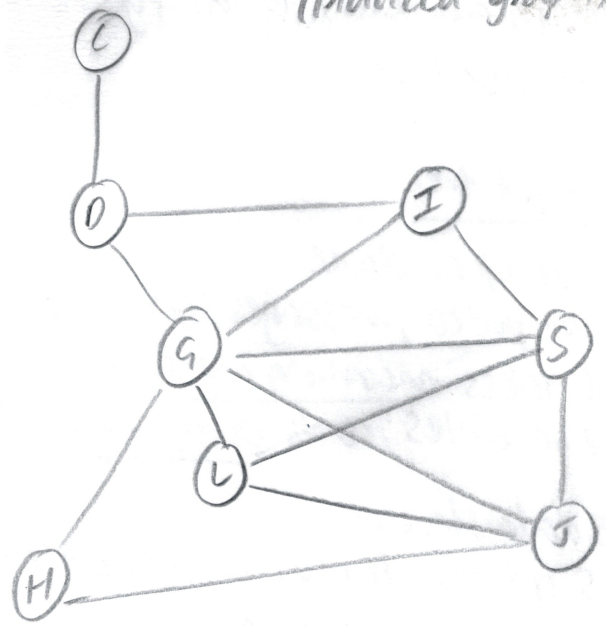


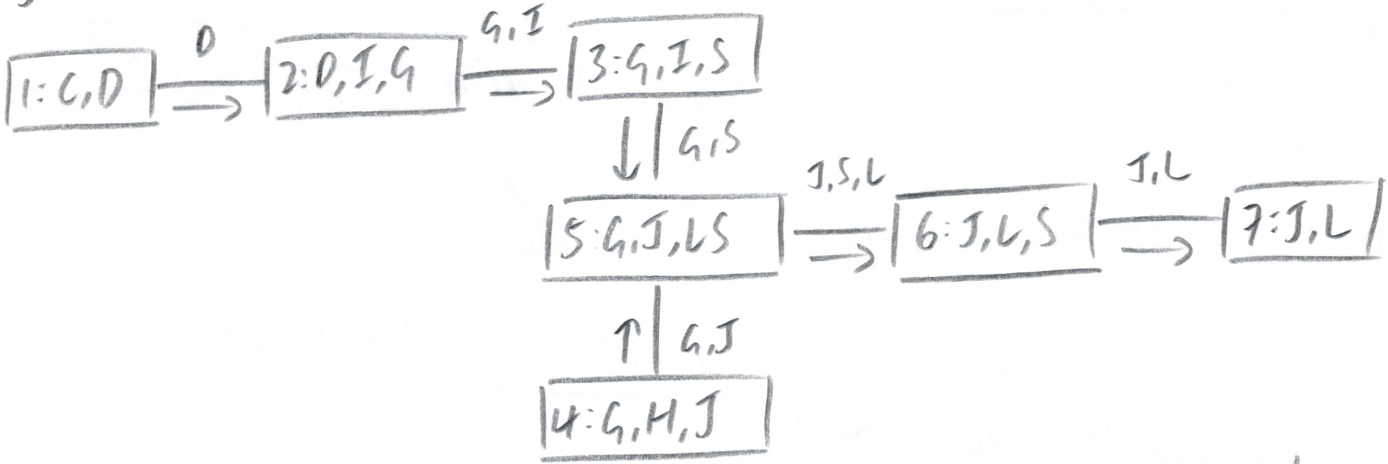
Fig 9.11 (a)
(Induced graph)



under VE $\prec = (C, D, I, H, G, S, L)$

- Fig 9.11(b) - shows maximal cliques (messy to represent) (this used to gen. clique tree)

- Fig 10.1 - cluster tree for VE $\prec = (C, D, I, H, G, S, L)$



(*) Note this is constructed from VE algo under above elim order $\prec$
(*) Note the clusters are not maximal
(*) How we get from Fig 10.1 → Fig 10.2 will be covered later in the section

---

10.2 - Message Passing - Sum Product

- Clique tree and variable elimination.
- Same clique tree - different executions - variable elim.
- Data structure - caching computations - multiple efficient executions

-(*) we have a set of factors Φ over $\mathcal{X}$; and assume we are given a clique tree $\mathcal{T}$ over Φ, as defined in D.4.17 (i.e. with maximal cliques)

(*) $\mathcal{T}$ satisfies (guaranteed) to satisfy the running intesection property and family preservation

---

10.2.1 - VE in clique tree

-clique tree - VE op guidance

(*) factors if computed; messages sent along edges

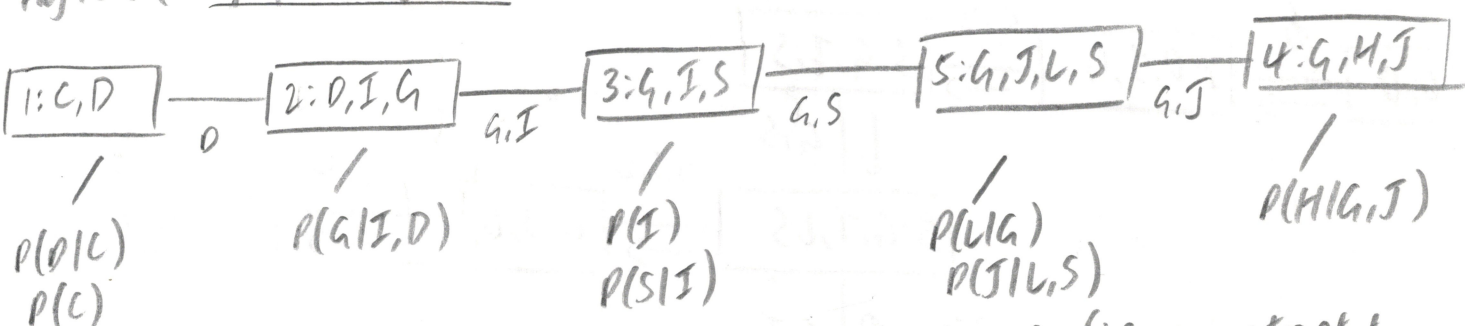
(*) each clique takes incoming messages (factors), multiplies them, sums out one or more variables; sends outgoing message to another clique.

(*) clique tree data structure:-
  i) dictates ops. over factors in clique tree
  ii) partial order over operations.

(*) If clique C' requires a message from C; C' must wait until C performs computation and sends message to C'. comput.

Alarm example 10.2.1.1

-fig 10.2 (simplified clique tree) (based on student network)



(*) note that this is constructed from maximal cliques (i.e. in contrast to fig 10.1); non-maximal cliques absent

- cover this later.

(*) $\mathcal{T}$ satisfies both family preservation; running intesection property

(*) also specifies assignment α of initial factors (CPDs) to cliques.

-going to list steps here:-

1) Generate initial potentials
2) Define query
3) Select root clique
4) Execute message passing upstream to root clique

1) Generate initial potentials for each clique:- $\psi_i^{init.}$

- each $\psi_i(C_i)$ generated by multiplying factors assigned to clique C_i
- showing this all explicitly

$$P(C,D,I,G,S,J,L,H) = P(C)P(D|C)P(I)P(G|I,D)P(S|I)P(L|G)P(J|L,S)P(H|G,J)$$

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

$$\phi_H(H,G,J)$$

- Assignment of initial factors to cliques: (four initial pot.)

Clique C_1: $P(C), P(D|C) \longrightarrow \psi_1(C,D) = \phi_C(C)\phi_D(D,C)$

C_2: $P(G|I,D) \longrightarrow \psi_2(G,I,D) = \phi_G(G,I,D)$

C_3: $P(I), P(S|I) \longrightarrow \psi_3(G,I,S) = \phi_I(I)\phi_S(S,I)$

C_4: $P(H|G,J) \longrightarrow \psi_4(H,G,J) = \phi_H(H,G,J)$

C_5 $P(L|G), P(J|L,S) \rightarrow \psi_5(G,J,L,S) = \phi_L(L,G)\phi_J(J,L,S)$

---

2) Define query - $P(J)$

---

3) select root clique

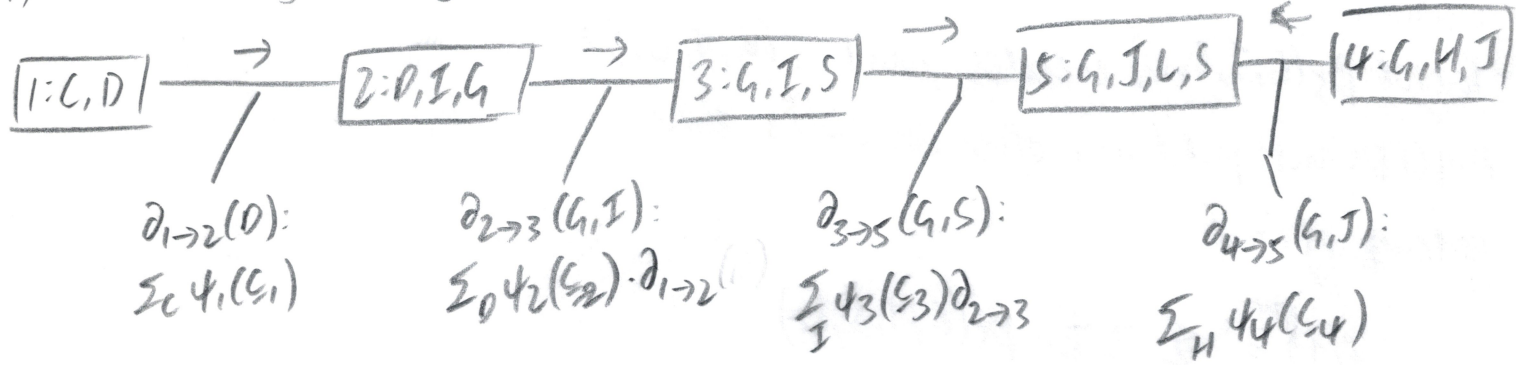
i.e. any of cliques $C_1 \cdots C_5$ which contains J.

e.g. C_4 or C_5 : randomly choose C_5 (makes no diff.) (arbitrarily)

---

4) Execute message passing:- (root clique C_5)

root clique ↓



PTO. ⟶

C_1: Eliminate C:-

$\sum_C \psi_1(C,D)$ generates a factor with scope D. Define this as a message to send to C_2.

$\delta_{1\to 2}(D) = \sum_C \psi_1(C,D)$

C_2: Eliminate D:

- define $\beta_2(G,I,D) = \delta_{1\to 2}(D) \cdot \psi_2(G,I,D)$

$\sum_D \beta_2(G,I,D)$ generates a factor over G,I. Define as message to send to C_3.

$\delta_{2\to 3}(G,I) = \sum_D \psi_2(G,I,D) \cdot \delta_{1\to 2}(D) = \sum_D \beta_2(G,I,D)$

C_3: Eliminate I:

- define $\beta_3(G,S,I) = \delta_{2\to 3}(G,I) \cdot \psi_3(G,S,I)$

$\sum_I \beta_3(G,S,I)$ generates factor over G.S ... to C_5

$\delta_{3\to 5}(G,S) = \sum_I \psi_3(G,S,I)\, \delta_{2\to 3}(G,I)$

C_4 elim. H

$\delta_{4\to 5}(G,J) = \sum_H \psi_4(H,G,J)$ to C_5

C_5: (root clique):-

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

- Factor $\beta_5(G,J,S,L)$ encodes joint $P(G,J,L,S)$
- All CPDs multiplied in; other variables elim.
- Obtain $P(J)$:-

$$P(J) = \sum_G \sum_S \sum_L \beta_5(G,J,S,L).$$

(*) constraint on operder

- only constraint: clique must get all of its incoming messages from downstream neighbours before sending outgoing messages to upstream neighbours.

- ready clique - clique has received all incoming messages

(*) valid execution orders

(*) varying root clique choice; varying query (minor, charges)

10.2.1.2. - clique tree message passing

- a general, formal specification of above.
- component specification; then pseudocode

(*) general variable elim algo - message passing on a clique tree

(*) let T be a clique tree with cliques $C_1, \ldots, C_K$

- multiply factors (assigned to each clique) → initial potentials
- pass messages between neighboring cliques; sending them all to root clique

initial potential
- each factor $\phi \in \Phi$ is assigned to clique $\alpha(\phi)$ (- can be viewed as a function that takes factor, returns clique index)
- initial potential of C_j (clique j)

$$\psi_j(C_j) = \prod_{\phi : \alpha(\phi) = j} \phi$$

- As each factor is assigned to one clique (*):

$$\prod_{\phi} \phi = \prod_{j} \psi_j$$

(i.e. product of all init. potentials is equivalent to product of initial factors (CPDs))

root cliques; neighbours
- C_r - selected root clique
- perform sum-product VE over cliques (start from leaves of clique tree inwards)

- For a clique C_i:
  - NB_i - set of indexes of cliques that are neighbours of C_i

- $pr(i)$ - upstream neighbour of i (one on path to root clique)

(*) Each clique C_i, except for root, performs a message passing comp. and sends a message to upstream neighbour $C_{pr(i)}$

sum-prod messg pass: message from C_i to C_j:-

$$\delta_{i \to j} = \sum_{C_i - S_{i,j}} \psi_i \cdot \prod_{k \in (NB_i - \{j\})} \delta_{k-i}$$

(*) clique C_i multiplies all incoming messages from its other neighbours with its initial clique pot, resulting in a factor ψ whose scope is the clique. (example earlie uses β) (2)

(*) then sums out all variables except those in sepset between C_i and C_j, sends resulting factor as a message to C_j

Message passing process:

(*) Proceeds up tree - root clique culmination

(*) Root clique - received all messages → multiply with its own initial potentials.

(*) result is a factor called the beliefs $\beta_r(C_r)$ (only for root clique does this special term apply (?))

- this represents:-

$$\tilde{P}_\Phi(C_r) = \sum_{\chi - C_r} \prod_{\phi} \phi$$

Algorithm 10.1 - pseudocode (upward pass only)

- no need to write out
- check you understand (*) select root clique

(*) Ready cliques at beginning of algo ⇒ start at leaves (as they do not receive messages

(*) CF at beginning, calls subroutines - initialise clique, SP-message.

(*) searches for ready cliques (implicitly defines an ordering)

- executes sum-product message passing
- compute belief at root clique C_r to get β_r

(*) Note this algorithm can be run only for a selected root clique C_r ; which is informed by the query variable.

(*) E. 10.3 → shows importance of ordering of operations.
- multiple orderings possible

(*) use algo 10.1 to compute magnal prob - query nodes $(\underline{Y}) \rightarrow P(\underline{Y})$

(*) select query nodes fully contained in clique to be root. C_r

(*) Do Algo 1 ; then :-
extract $\tilde{P}_{\underline{\Phi}}(\underline{Y})$ from final pot. at C_r by summing out $(C_r - \underline{Y})$ (non query variables).

10.2.1.3 Correctness

correctness: Algo 10.1 applied to $\mathcal{T}$ (satisfying running intersect, fam pres.) computes desired expressions over messages, cliques

↳ Algo eliminates a variable X only when a message is sent from C_i to neighboring C_j : $X \in C_i$ and $X \notin C_j$ (simple).

Proposition 10.2

- Assume that X is eliminated when message sent from C_i to C_j
- Then X does not appear anywhere in the tree on the C_j side of the edge $(i-j)$.

(R): Semantic interp of messages

- let (i,j) be edge in cliquetree
- $F_{<(i \to j)}$ - set of factors in cliques on C_i-side of the edge
- $V_{<(i \to j)}$ - set of variables on C_i-side but are not in sepset (S_{i-j})

(*) see example (10.2) clique tree

$F_{<(i \to j)} = \{P(C), P(D|C), P(G|I,D), P(I), P(S|I)\}$

$V_{<(i \to j)} = \{C, D, I\}$

(*) message passed between cliques C_i and C_j is product of all factors in $F_{<(i \to j)}$ marginalised over the variables in the sepset S_{i-j}

(that is, summing out all the variables $V_i(i \to j)$)

Theorem 10.3

- $S_{i,j} = S_i \cap S_j = S_i \cap C_j$ (sepset variables)
- let $\delta_{i \to j}(S_{i,j})$ be a message from C_i to C_j
- then $\delta_{i \to j}(S_{i,j}) = \sum_{V_i(i \to j)} \prod_{\phi \in F_i(i \to j)} \phi$.

(R): related to T. 10.2.

- sepset divides graph into conditionally independent pieces
- CI property - allows message (defined over sepset) to summarise completely the info on one side of the clique tree that is necessary for comp. on the other.

Corollary 10.1

- C_r - root clique
- β_r is computed via algo 10.1
- $\beta_r(C_r) = \sum_{\chi - C_r} \tilde{P}_\Phi(\chi)$

(R): BN, $\mathcal{B}$: Φ consists of CPDs in $\mathcal{B}$ reduced with evidence e

$\Rightarrow \beta_r(C_r) = P_{\mathcal{B}}(C_r, e)$

MN $\mathcal{H}$: Φ consists of compt/potentials

$\Rightarrow \beta_r(C_r) = \tilde{P}_\Phi(C_r)$

- obtain probability over variables in C_r, by normalising resulting factor to sum to 1
- MN $\to$ obtain value of partition by summing up all entries in potential of root clique $\beta_r(C_r)$.

10.2.2 clique tree calibration

(*) (CT): concerns computation of probability queries for multiple variables (rather than 1 variable as above through choice of root clique) (or variable scope of root clique)

(*) in particular, partially obs. settings → pd over all latent variables (and parents)

(*) consider computing posterior over every r.v. in network:-

naive:- run clique tree inference (upward pass Algo 10.1) n times for each $X_1, \ldots, X_n$ you want to query - comp. cost - nC

less naive:- run clique tree inference for every clique - comp. cost - KC

where C - comp. cost of one run of Algo 10.1 (upward pass of VE in C.T.)

(*) can do much better

(*) observation:-

- 2 neighboring cliques C_i and C_j in some clique tree.
- via 1.10.3 (?) → value of message sent from C_i to C_j does not depend on specific choice of root clique C_r.

(*) As long as root clique C_r is on the C_j-side, exactly same message is sent from C_i to C_j

(*) same argument for root clique on C_i-side (?) ✓ i.e. edge directionality

(*) essentially, it is not the specific position of root clique C_r, but where it is relative to a particular pair of the adjacent cliques in question (i.e. C_i or C_j-side); that matters for the value of the message sent

(*) any given clique tree - each edge - 2 messages - for each edge direction.

C cliques ⇒ (C-1) edges ⇒ 2(C-1) messages to compute.

W: next move → sum-product belief propagation / sum product message-passing, or Shafer-Shenoy (1990) algorithm applied to PGMs.

(*) extend previous concept of a 'ready clique' to be bidirectional (?)

---

definition 10.4 (ready clique)

- T - clique tree
- C_i is ready to transmit to neighbor C_j when C_i has messages from all of its neighbors except from C_j

---

(*) lays extension → idea of upward/downward passes

(R): when C_i ready to transmit → C_j; can compute $\delta_{i \to j}(S_{i,j})$ by multiplying initial potential with all of its incoming messages except one from C_j and then eliminate variables in $\{C_i - S_{i,j}\}$ ✓

(*) uses another layer of dynamic programming

- Guestrin:- VE operates on factors by caching comput within a single inference op e.g. $P(X_i|e)$

CT enables caching computations across multiple inference ops $P(X_i, X_j)\ \forall i,j$.

(*) Next algo; Algo 2 ⟶ sum-product belief propagation

(*) Pseudocode - asynchronously; each clique sends message - when ready.

(*) message passing process:-

upward pass:- Pick a root clique, send all messages to root. When complete; root - has all messages.

Downward pass:- send the appropriate message to all its children. continues until all leaves of tree reached → no more msgs.

(*) Asynchronous pseudocode is equivalent; except root clique is first clique that happens to obtain all messages from neighbours.

(*) Actual implement - use schedule to prevent double comp of messages.

Algo 10.2 - pseudocode - calibration using sum-product message passing on a clique tree.

(n): This algo requires absolute clarity (some o/s issues). (w): uses an arbitrary root clique; does not matter which one.

- see Jordan (2003): Ch4

Example (10.4) - shows diagrammatically (Fig 10.5) downward pass from a partic. root clique.

---

(01): At the end of the process; compute beliefs for all cliques in the tree

(02): by multiplying the initial potential with each of the incoming messages

(03): Messages used in the computation of beliefs $\beta_i(C_i)$ for clique C_i

(04): are precisely the same messages that would have been used

(05): in a standard upwards pass of the algorithm with C_i as the root

---

Corollary 10.2

- Assume for each clique i; β_i computed as in Algo 2; then

$$\beta_i(C_i) = \sum_{\chi - C_i} \tilde{P}_\Phi(\chi)$$

16/06/2020

(R) C_i computes message to C_j (neighboring clique) based on initial potential ψ_i, not modified potential β_i

(*) β_i already integrates information from j; to do so ⇒ double counting factors assigned to C_j.

(*) Conclusion of algorithm → each clique contains marginal (unnormalised) probability over variables in its scope

(*) Computing marginal prob over query variable ⇒ eliminate other redundant variables in clique.

(*) If X appears in two cliques, marginal must be the same.

Definition 10.3 (calibration / clique beliefs / sepset beliefs)

- Two adjacent cliques C_i and C_j are calibrated if:-

$$\sum_{C_i - S_{i,j}} \beta_i(C_i) = \sum_{C_j - S_{i,j}} \beta_j(C_j)$$

- A clique tree T is calibrated if all pairs of adjacent cliques are calibrated
- For a calibrated clique tree, we use the terms:-

i) clique beliefs:- $\beta_i(C_i)$

ii) sepset beliefs:- $\mu_{i,j}(S_{i,j}) = \sum_{C_i - S_{i,j}} \beta_i(C_i) = \sum_{C_j - S_{i,j}} \beta_j(C_j)$

(*) key advantage of Shafer-Shenoy sum-product b.p → computes posterior probability of all variables in a PGM in 2x comp. of upward pass in same tree.

(*) execution cost of message passing in clique tree to one root: C

cost of Algo 10.2 : 2C

(*) Compare with nC, KC for naive runs of message passing.