Inference in graphical models

Graphical model is joint distribution over domain V.

$$V = U \cup H \cup E$$

- E → observed variables (evidence)
- H → hidden variables (unobserved, don't care)
- U → query variables (visible variables) (unobserved, interesting)

Inference = answering queries about U, knowing that $E = e$.

Therefore, if P_V is the graphical model, the distribution

$$P_{U|E} = \frac{\int_{\Omega(H)} P_{U,H,E}}{\int_{\Omega(H \cup U)} P_{U,H,E}}$$

is central to the inference methods that we study.

Types of queries: $P_{U|E=e} = ?$

$P_{X|E=e} = ? \quad X \in U$

$\operatorname{argmax}_{\Omega(U)} P_{U|E=e}$

$\operatorname{argmax}_{\Omega(X)} P_{X|E=e} \quad X \in U$

"$E=e$" is called evidence. This type of evidence is called **categorical.** Sometimes evidence is given in the form of a likelihood, i.e. $\tilde{P}_E$ distribution over $\Omega(E)$.

categorical evidence $\Longleftrightarrow$ $\tilde{P}_E = \begin{cases} 1 & \text{for } E=e \\ 0 & \text{otherwise} \end{cases}$

Brute force computation of $P_{U|E}$ (binary variables)

$$P_{U|E}(u) = \frac{\sum_{h \in \Omega(H)} P_{U,H,E}(u,h,e)}{\sum_{h \in \Omega(h)} \sum_{u \in \Omega(u)} P_{U,H,E}(u,h,e)}$$

$|\Omega(H)| = 2^{|H|}$

$|\Omega(U)| = 2^{|U|}$

$\Rightarrow \mathcal{O}(2^{|H|+|U|})$ operations — exponential in the number of variables that are marginalized over

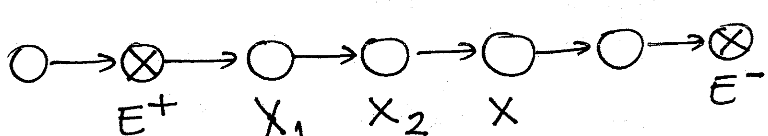
Can we do better by exploiting the structure of graphical models? Yes, if structure is sparse (small no. of parents per variable, small clique size, etc)

Exploiting structure = exploiting independence

Pearl's algorithm

- Bayes nets, singly connected
- computes $P_{X|E}$ for each $X \in V \setminus E$
- linear time (in $|V|$)

Pearl's algorithm in a chain



First idea: $P_{X|E^+E^-} = \frac{P_{X,E^+,E^-}}{P_{E^+E^-}} = \frac{P_{E^+} \cdot P_{X|E^+} \cdot P_{E^-|X,E^+}}{P_{E^+E^-}}$

$$\propto P_{X|E^+} \cdot P_{E^-|X}$$

Computing $P_{X|E^+}$

$\longrightarrow$ propagation of $P_{X_i|E^+}$

$$\underset{E^+}{\otimes} \to \underset{X_1}{\bigcirc} \to \underset{X_2}{\bigcirc} \to \cdots$$

$P_{X_1|E^+}$ known

$$P_{X_2|E^+}(x_2|e^+) = \sum_{x_1} P_{X_1|E^+}(x_1|e^+) \cdot P_{X_2|X_1}(x_2|x_1)$$

.... and so on, recursively until X

Note that: - computation is recursive, propagating $P_{X|E^+}$ down the chain
- every stage involves only 2 variables: X_i and its parent, $P_{X_i|X_{i-1}}$ and $P_{X_{i-1}|E^+}$ (the previous "message") $\Rightarrow$ **LOCALITY**
- total nr. of steps $\leq |V| = n$
- computations / step $|\Omega(X_i)| \cdot |\Omega(X_{i-1})|$

if $|\Omega(X)| \leq r \Rightarrow$ total computation $\sim n r^2$

Computing $P_{E^-|X}$

$\longleftarrow$ propagation of $P_{Y_i|E^-}$

$$\bigcirc \to \underset{Y_2}{\bigcirc} \to \underset{Y_1}{\bigcirc} \to \underset{E^-}{\otimes}$$

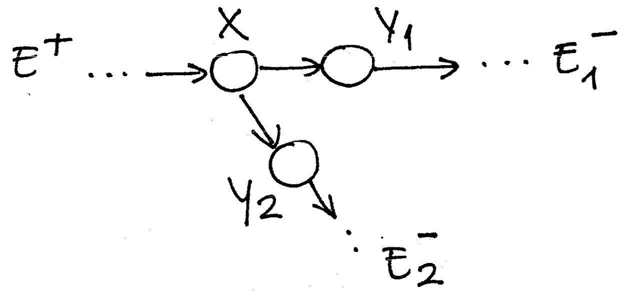
$P_{E^-|Y_1}$ known

$$P_{E^-|Y_2} = \sum_{y_1} P_{E^-|Y_1} \cdot P_{Y_1|Y_2}$$

... and so on

Algorithm
1. Propagate $P_{X|E^+}$ down the chain
2. Propagate $P_{E^-|X}$ up the chain
3. Compute $P_{X|E^+E^-}$ by normalizing $P_{X|E^+} \cdot P_{E^-|X}$

Pearl's algorithm in a tree



<u>up-propagation</u> E^- is now $\{E_1^-, E_2^-\}$

but $E_1^- \perp E_2^- \mid X$

$$P_{E_1^- E_2^- | X} = P_{E_1^-|X} \cdot P_{E_2^-|X}$$

<u>down propagation</u> assume we have $P_{X|E^+}$

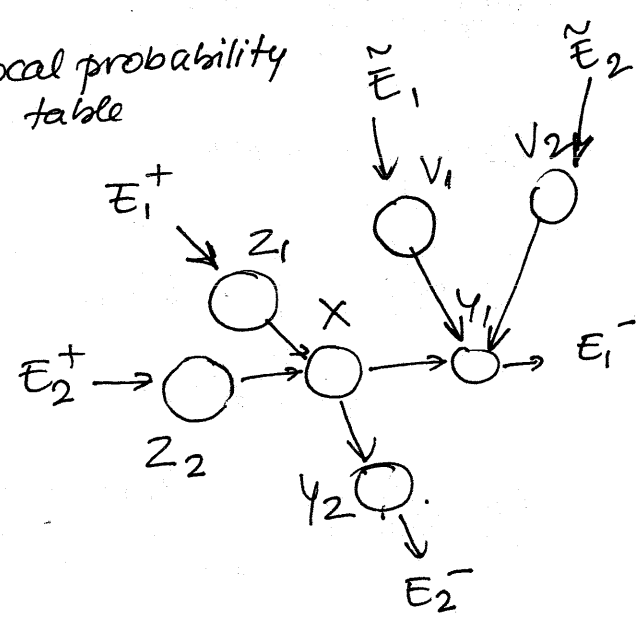
what is $P_{Y_1|\text{"evidence above } y_1\text{"}}$?

$$P_{Y_1|E_1^+ E_2^-} = \sum_x P_{X|E_1^+ E_2^-} \cdot P_{Y_1|X}$$

$$\propto \sum_x P_{X|E^+} \cdot P_{E_2^-|X} \cdot P_{Y_1|X}$$

involves : downward msg. of parent , upward msgs of siblings , local probability table

Pearl's algorithm in polytrees



polytree = singly connected network

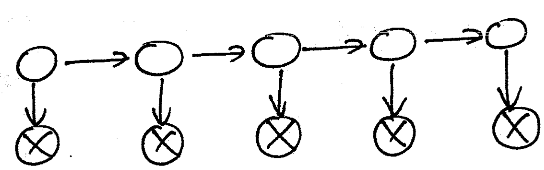
(see [Pearl88] chapter 4, section 5

Running time / step $\sim r^{|pa(X)|}$

Conclusion about Pearl's algorithm

- computes $P_{X/E}$ for all $X \in V \setminus E$
- running time :– linear in $n = |V|$
  - polynomial in $r = \max|\Omega(X)|$ if max # parents bounded
  - exponential in $\underbrace{\text{ie}}_{\max}$ # parents of $X \in V$
- involves only <u>local</u> computations
  - local variables
  - local prob. tables (parameters)
  - messages from parents, children
- makes use of <u>independence</u> relationships
- works exactly on singly connected Bayes nets (polytrees)

<u>Special cases</u>



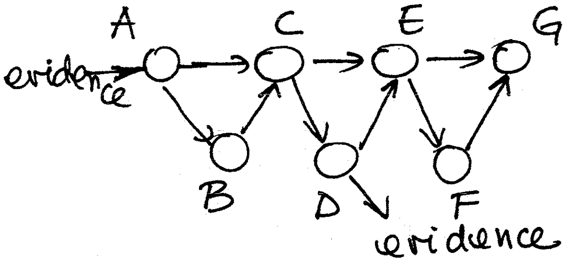
<u>Hidden Markov Model</u>

Pearl's alg $\Leftrightarrow$ Forward - Backward Alg.
(can do max propagation as well $\Leftrightarrow$ Viterbi)

HMM + continuous variables + gaussian distribution + linear dependen
$\doteq$ <u>Linear system</u> (discrete time)
$\Downarrow$
Pearl's algorithm $\Leftrightarrow$ [batch version of] Kalman filter

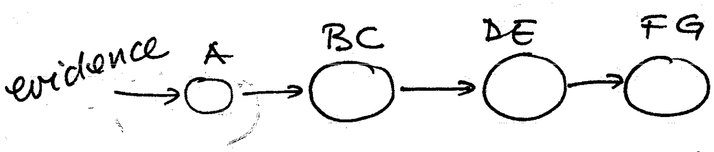
Pearl's alg. itself is a special case of the <u>junction tree</u> alg that will be discussed later on.

What to do on **Bayes nets with loops**?



- Pearl's algorithm — approximation!
- conditioning = breaking the loops
- condense variables
- convert to junction tree —
  — general method

Condensation

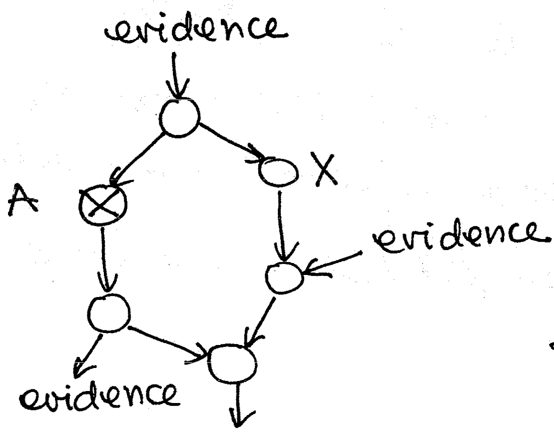


computation cost ~ exponential in the size of largest set of condens. variables

Junction tree alg is a special case of condensation + + propagation

Conditioning

assume A known ⇒ blocks a path and breaks cycle



$$P_{X|\text{evidence}} = P_{X|A=0,\text{evid}} \cdot P(A=0|\text{evidence}) + P_{X|A=1,\text{evid}} \cdot P(A=1|\text{evidence})$$

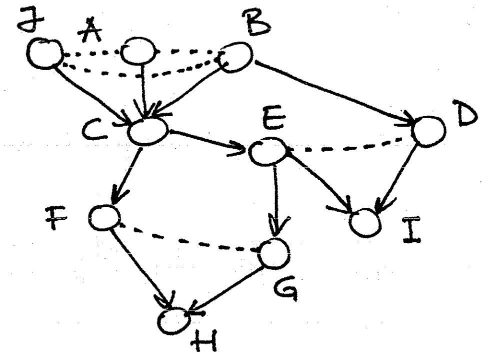
$$P(A=0|\text{evidence}) \propto P(\text{evidence}|A=0) \cdot P_A$$

computation cost ~ exponential in the number of instantiated variables

Converting a Bayes net into a junction tree

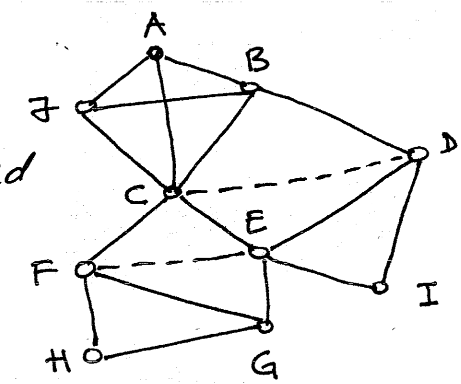
– by adding edges ⇒ hides some independence relationships
(⇒ they won't be exploited in the inference procedure)

1. Moralization



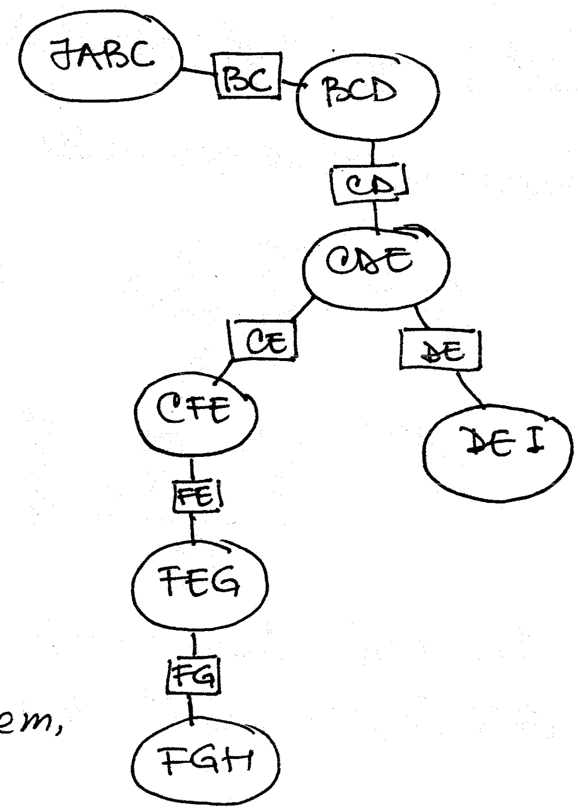
- "marry" parents of common child
- then drop direction of edges

2. Triangulation



- add edges until graph triangulated
- triangulation is not unique!

3. Junction tree construction



- junction tree may not be unique (same cliques, same separators but different topologies)

Total "state space" size =

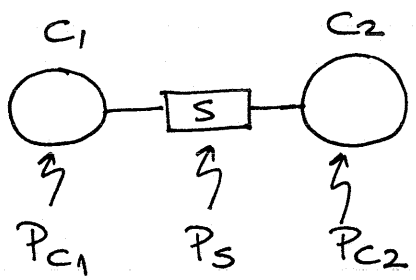
$$= \sum_{C \in \text{cliques}} |\Omega(C)|$$

indicator of total memory, propagation time

Wanted triangulations that give small total state space. Hard problem, only heuristics available.

The Junction tree algorithm

Passing messages between two cliques



before propagation:

$$P_S = \sum_{\Omega(C_1 \setminus S)} P_{C_1} = \sum_{\Omega(C_2 \setminus S)} P_{C_2}$$

we say that junction tree is "balanced"

new info: $P_{C_1} \leftarrow P_{C_1}^*$

Propagating this new info to C_2 is called <u>absorbtion</u>.

1. $P_S^* = \sum_{\Omega(C_1 \setminus S)} P_{C_1}^*$ new separator potential

2. $P_{C_2}^* = P_{C_2} \cdot \dfrac{P_S^*}{P_S}$ replaces P_{C2}; then P_S^* replaces P_S

Why does this operation make sense?

<u>First</u>, note that absorbtion does not change the joint distributio[n]

Supposing there are only two cliques, i.e. $V = C_1 \cup C_2$:

$$P_V = \frac{P_{C_1}^* \cdot P_{C2}}{P_S} \quad \text{before absorbtion}$$

$$P_V^{new} = \frac{P_{C_1}^* \cdot \left(P_{C_2} \cdot \frac{P_S^*}{P_S}\right)}{P_S^*} = P_V \quad \text{after absorbtion}$$

Thus, absorbtion distributes info that is already in the distribution. But how does this info get incorporated?

Second Assume that $P_V = \dfrac{P_{C_1} \cdot P_{C_2}}{P_S}$ and

you observe some [subset of] variable[s] $A \in C_1 \setminus C_2$.

Hence $A = a$. Before the observation we had

$$P_A = \sum_{\Omega(C_1 \setminus A)} P_{C_1} \quad = \text{the marginal of } A \text{ computed in } P_{C_1}$$

After the observation, we have a new "P_A":

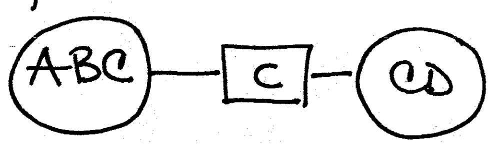
$$P_A^* = \delta_a = \begin{cases} 1 & A = a \\ 0 & \text{otherwise} \end{cases}$$

We incorporate this into C_1 by the law of conditional probabili

(let's denote $C_1 \setminus A = B$)

$$P_{C_1}^* = P_{C_1 / \text{observation}}(b, a' \mid A = a) = \begin{cases} \dfrac{P_{C_1}(b, a)}{P_A} & \text{if } a' = a \\ 0 & \text{if } a' \neq a \end{cases}$$

Example:



A, B, C, D binary variables

$P_{ABC} =$

BC \ A	0	1
00	0.2	0.1
01	0.15	0.05
11	0.2	0.2
10	0.05	0.05

$\Rightarrow P_C =$

0	1
0.4	0.6

$P_{CD} =$

D \ C	0	1
0	0.1	0.5
1	0.3	0.1

Observed $A=0$

$$P_A(a=0) = 0.2+0.15+0.2+0.05 = 0.6$$

$$P^*_{ABC} = \frac{P_{ABC}\cdot\delta_{a,0}}{P_A} = \frac{1}{0.6}\begin{array}{|c|c|}\hline 0.2 & 0\\ \hline 0.15 & 0\\ \hline 0.2 & 0\\ \hline 0.05 & 0\\ \hline\end{array}$$

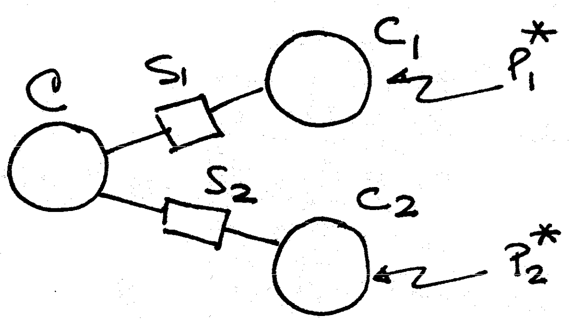
$$P^*_C = \sum_{a,b} P^*_{ABC}(a,b,c) = \begin{array}{|c|c|}\hline \frac{0.25}{0.6} & \frac{0.35}{0.6}\\ \hline\end{array}$$

$$P^*_{CD} = P_{CD}\cdot\frac{P^*_C}{P_C} = \begin{array}{c|c|c|} D\backslash C & 0 & 1\\ \hline 0 & 0.104 & 0.486\\ \hline 1 & 0.312 & 0.097\\ \hline\end{array}$$

for exs $P_{CD}(0,1) = 0.3$

$$P^*_{CD}(0,1) = P_{CD}(0,1)\cdot\frac{P^*_C(0)}{P_C(0)} = 0.3\,\frac{\frac{0.25}{0.6}}{0.4} = 0.312$$

How to absorb from several cliques ?



1. $P^*_{S_i} = \sum_{\Omega(C_i\backslash S_i)} P^*_{C_i} \qquad i=1,2,\cdots$

2. $P^*_C = P_C\cdot\frac{P^*_{S_1}}{P_{S_1}}\cdot\frac{P^*_{S_2}}{P_{S_2}}\cdots$

This is justified by the requirement that P_V remain constant after absorbtion.

The general case

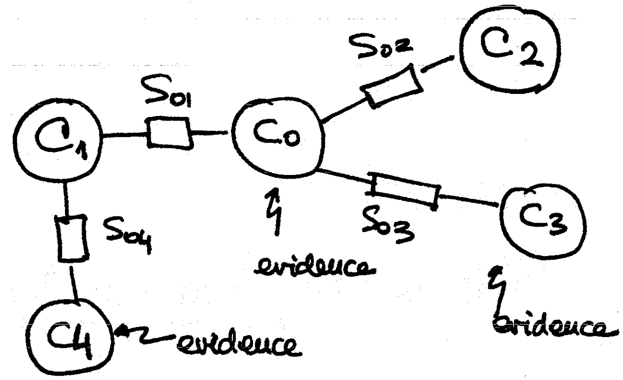
Arbitrary junction tree $\quad P_V = \dfrac{\prod P_{C_i}}{\prod P_{S_j}}$

Arbitrary categorical evidence $\quad X_k = x_k^* \quad k = \overline{1, n_e}$

1. Incorporate evidence:

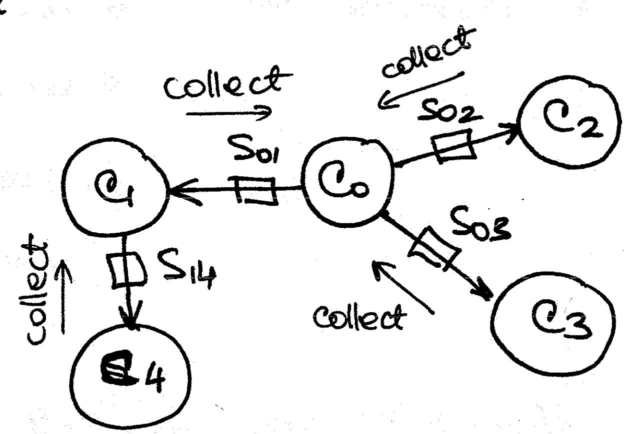
   for each k find C_i clique such that $X_k \in C_i$

$$P^*_{C_i} = \frac{P_{C_i} \cdot \delta_{X_k x_k^*}}{P_{X_k}(x_k^*)}$$



2. "Direct" the junction tree.

   - choose [any clique] C_0 as root
   - direct all edges away from root



3. Collect evidence

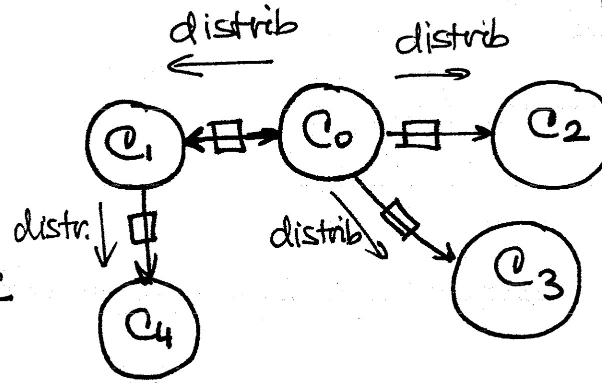
starting from root do recursively:

- call "collect evidence" in all children
- then absorb from all children

(this results in a series of absorbtions from leaves towards the root. For ex: C_1 absorbs from C_4

C_0 — " — C_1, C_2, C_3

4. "Distribute evidence"



starting from root do recursively for each clique C:

- each child of C absorbs from C
- each child calls "distribute evidence"

For ex, here : C_1, C_2, C_3 absorb from C_0

C_4 absorbs from C_1

END

This is the Junction Tree Algorithm.

It ensures that :
- the tree is balanced
- every P_C^*, P_S^* is normalized
- every P_C^* is the marginal of C in P_V^*
- P_V^* represents Prob[V | evidence]

Non-categorical evidence :

- info from external source tells us $P^*_{X_k}$ $(\neq P_{X_k})$ AND $(\neq \delta_{X_k, x_k^*})$
- then $P^*_{C_i} = \dfrac{P_{C_i} \cdot P^*_{X_k}}{P_{X_k}}$ in (1.) and the other steps remain unchanged

Computational complexity of the junction tree algorithm

Absorbtion $C_1 \to C_2$ takes $\sim |\Omega(C_1)|$ additions
$|\Omega(C_2)|$ multiplications

Collect evidence : absorbtions children $\to$ parent
Distribute evidence : — " — parent $\to$ child

$\| \Rightarrow$

$$\text{whole alg} \approx 2 \cdot \underbrace{\sum_{C \in \text{cliques}} |\Omega(C)|}_{2.T} \text{ additions and multiplications}$$

T is called "total state space"; it's an approximate measure of the cost of inference in the junction tree.

Note that:
- $|\Omega(C)|$ is exponential in $|C|$
- therefore $T \ll |\Omega(V)|$ if tree is sparse

  ↑ cost of "brute force" inference
- it is important to minimize T during triangulation. (but this is hard!)
- a good approximation to T is $\max_C |\Omega(C)|$ (minimizing this is also NP-hard)

The JT algorithm
- is exact and general
- any exact, general algorithm for inference has to perform a [possibly hidden] triangulation; hence, it is essentially no better than the JT algorithm
- unfortunately, for many practical problems the JT is intractable : image segmentation, max likelihood decoding, medical diagnosis in QMR-DT

Approximate inference methods

16-732
Oct 13, 1999

- Pearl's (polytree) algorithm ← for Bayes nets
  - comes with no guarantees
  - seems to work in practice
- Markov Chain Monte Carlo methods (MCMC) ← for Markov fields mainly

  i.e. sampling
- variational methods ← for Bayes nets, Markov fields

  approximating P_V/evidence

Graphical Probability Models (aka Belief Nets)

Pearl ('88):
"probability theory is unique in its ability to process context-sensitive beliefs, and what makes the process computationally feasible is that the information needed for specifying context dependencies is represented by graphs and manipulated by local propagation".

Graphical model = graphical representation of (conditional) independence relationships in a joint distribution
= the distribution itself

graphical model
- **structure** (a graph)
- **parametrization** (depends on graph, parameters are "local")

graph = (V , E)

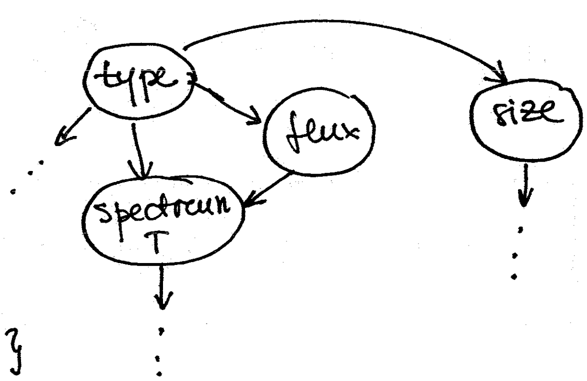
V → nodes, variables

E → edges, "dependencies"

> More precisely: a missing edge represents an independence relationship.

But what is "independence"?

Example: "Redshift" model



V = { type, flux, size, ... }

E = { type → flux, type → size, ... }

Probabilistic independence

$$\boxed{A \perp B \iff P_{AB} = P_A \cdot P_B}$$

"A independent B"

(marginal formulation)

<u>Example</u> 1. A, B binary variables, $A \perp B$

P_{AB}:

B \ A	0	1	P_B
0	0.12	0.28	0.4
1	0.18	0.42	0.6
P_A	0.3	0.7	

<u>Example</u> 2. A, B real, jointly gaussian

$$[A\ B] \sim N([\mu_A\ \mu_B], \Sigma_{AB})$$

$$P_{AB}(a,b) = \frac{1}{|\Sigma|^{1/2} \cdot 2\pi} \exp\left\{-\frac{1}{2} (a-\mu_A\ \ b-\mu_B) \Sigma_{AB}^{-1} \begin{pmatrix} a-\mu_A \\ b-\mu_B \end{pmatrix}\right\}$$

$$A \perp B \iff \Sigma_{AB} = \begin{bmatrix} \sigma_A^2 & 0 \\ 0 & \sigma_B^2 \end{bmatrix} \quad \text{(diagonal)}$$

Ex 1. Prove this!

Ex 2.

$$\begin{matrix} A \perp B \\ A \perp C \end{matrix} \ \overset{?}{\Longrightarrow} \ A \perp B, C$$

$$\begin{matrix} A \perp B \\ A \perp C \end{matrix} \ \underset{?}{\Longleftarrow} \ A \perp B, C$$

Is any of the implications $\rightleftarrows$ true? Prove or give counterexample!

$$\boxed{A \perp B \iff P_{A|B} = P_A} \iff$$

knowing B brings no additional information about A

(conditional formulation)

Conditional independence

$$\boxed{A \perp B \mid C \iff P_{AB|C} = P_{A|C} \cdot P_{B|C} \iff P_{A|BC} = P_{A|C}}$$

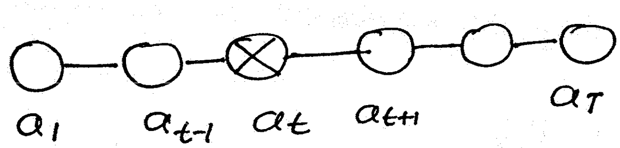
"once C is known, knowing B brings no additional information about A"

↳ has to hold for <u>all</u> values of A, B, C

Representing independences in graphs

Independence (in joint distribution) ⟷ Separation (in graph)

Markov chain

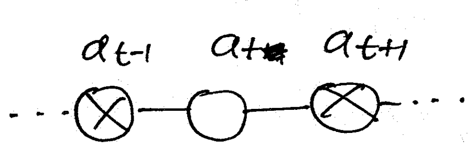


$i < t < j$

$a_i \perp a_j \mid a_t$

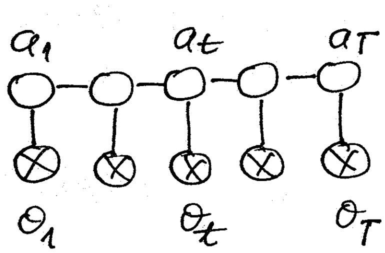
Knowing a state makes future independent from past.

Knowing a_{t+1}, a_{t-1} ⇒ no other state can give additional info about a_t



⇒ a_{t-1}, a_{t+1} <u>separate</u> a_t from the other variables

<u>Variant</u> **Hidden Markov Model**

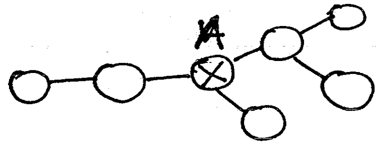


Separation properties allow for efficient algorithms for HMMs (forward-backward, Viterbi, Kalman filter, Baum-Welch)

There are T "hidden" variables. $\Rightarrow$ 2^T hidden states

But, forward-backw, etc, ... are $O(T)$ (i.e. linear!)

Tree



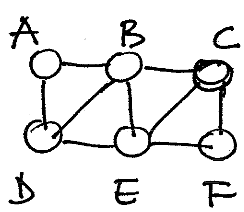
graph with no cycles (or singly connected)

Property – every variable A breaks the tree into 2 or more subtrees

– between two variables there is exactly 1 path

Trees are the only structure for which learning can be done efficie

Markov field



Separation

- $V_1 \perp V_2 \mid V_3 \iff V_3$ "blocks" all the paths from V_1 to V_2

- variable $\perp$ everything else | neighbors

(i.e. all info about a variable X that can be obtained from $V \setminus \{X\}$ is contained in the neighbors of X)

Examples: $A, D \perp C, F \mid B, E$

$B \perp F \mid C E$

$B \not\perp F \mid D E$

$A \perp C, E, F \mid \underbrace{B, D}_{\text{neighbors of } A}$

Parametrization

Clique = fully connected subgraph of a graph

Ex: • all edges are size 2 cliques

• ABD, BDE, BCE, CEF are size 3 cliques. They are maximal.

• BCDE is not a clique

Denote by $\mathcal{C}_G$ the set of maximal cliques of a graph G.

P_V Markov Field over graph G $\iff$ $\boxed{P_V(x) = \prod_{c \in \mathcal{C}_G} \psi_c(x_c)}$

Ex: $P_{ABCDEF} = \underbrace{\psi_{ABD}(abd)}_{\downarrow} \psi_{BDE}(bde) \psi_{BCE}(bce) \psi_{CEF}(cef)$

"local" function/parametrization – depends only on the variables ABD

How much memory is saved?

- assume $A, B, .. F$ binary variables
- assume ψ are [probability] tables

P_{ABCDEF} = prob. table with 2^6 entries, 2^6-1 free parameters

ψ_{ABD} = prob table with 2^3 entries, 2^3-1 " "

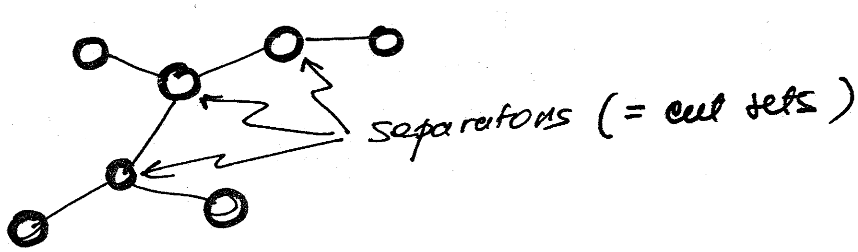
saving: $2^6-1 - 4(2^3-1) = 35$ out of 63 (more than 50%)

Ex: Tree over n variables (binary)

$T \equiv P_V = \prod_{AB \in E} \psi_{AB}$ Memory: $3(n-1)$

For a tree, ψ_{AB} can be related to the corresponding marginals P_{AB}

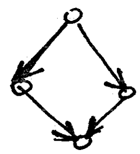
$$\boxed{T = \frac{\prod_{AB \in E} P_{AB}(a,b)}{\prod_{A \in V} P_A(a)^{\deg A - 1}}} = \frac{\prod P_{clique}}{\prod P_{separator}}$$



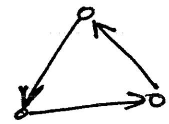
Bayes Nets

Oct 6, 199[illegible]
16-732

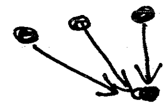
DAG (Directed Acyclic Graph) = directed graph with no directed cycles



a DAG

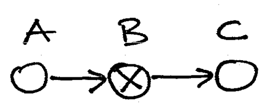


not a DAG

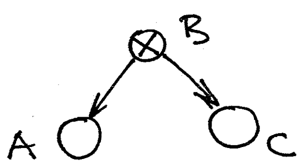


parents } family
child

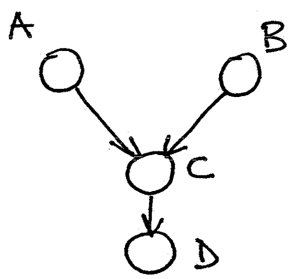
Separation in a DAG (d-separation)



$A \perp C \mid B$ (path $A \to B \to C$ blocked)



$A \perp C \mid B$, but $A \not\perp C$



$A \perp B$ but $A \not\perp B \mid C$
$A \not\perp B \mid D$

(path $A \dashrightarrow B$ blocked by C or D)

In general $A \perp B / C$ in a DAG iff all paths btw. A and B are blocked when C is instantiated.

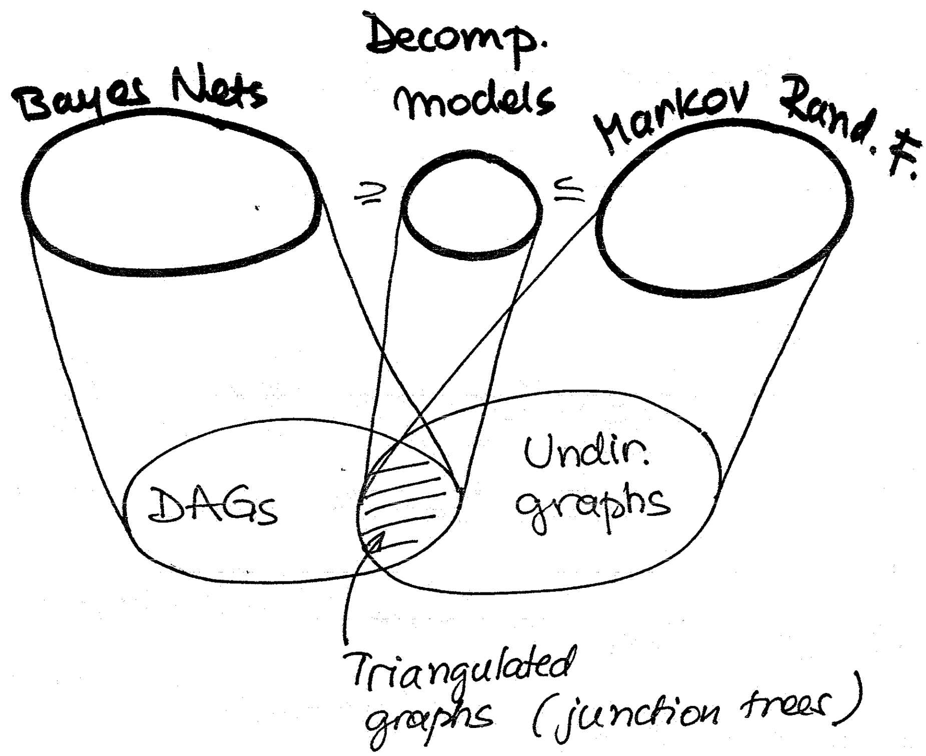
Understanding separation is important for inference: if two variables are separated we can ignore one when reasoning about the other.

Parametrization

$pa(A)$ = parents of variable A

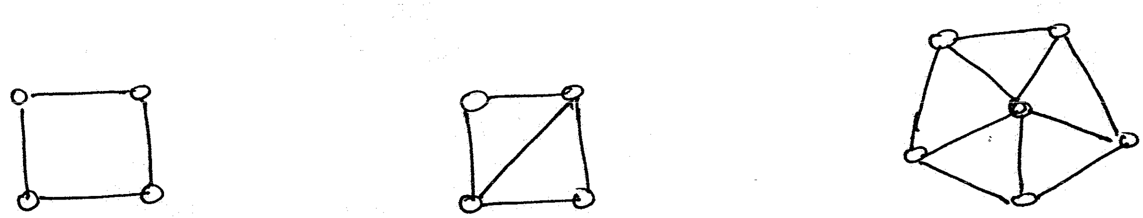
$$P_V = \prod_{A \in V} \underbrace{P_{A|pa(A)}}_{\text{local fctions}}$$

local fctions (depend only on A, $pa(A)$)



Triangulated graphs / decomposable models

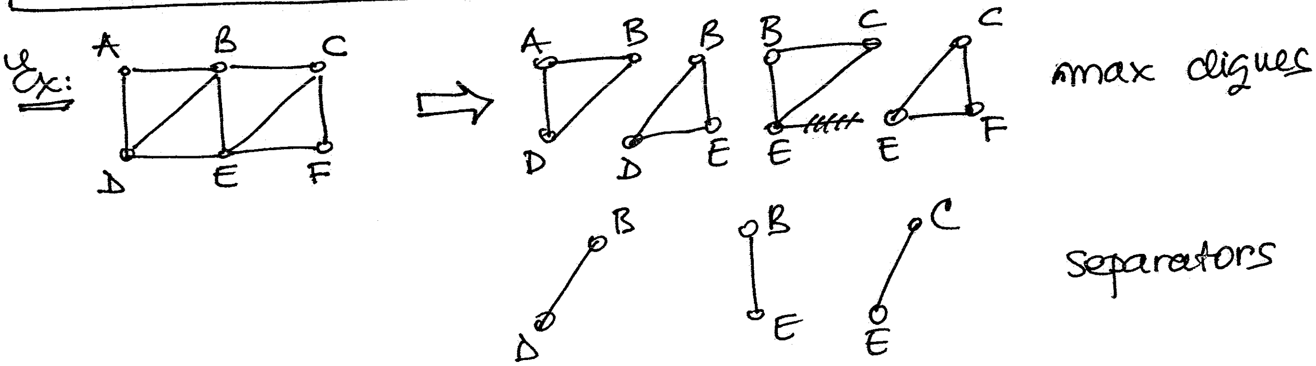
Graph G <u>triangulated</u> (<u>or</u> <u>chordal</u>) ⟺ every cycle ≥4 in G has a chord



not triangulated   triangulated   NOT triangulated

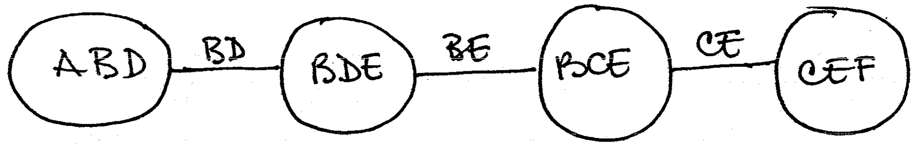
Triangulated graphs can be represented as a tree of max cliques.

Ex:



Junction tree

⇨



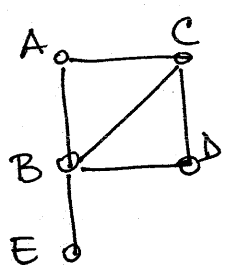
⇨ parametrization:

$$P_{ABCDEF} = \frac{P_{ABD} \cdot P_{BDE} \cdot P_{BCE} \cdot P_{CEF}}{P_{BD} \cdot P_{BE} \cdot P_{CE}} = \frac{\prod_{C} P_{cliques}}{\prod_{S} P_{separators}}$$

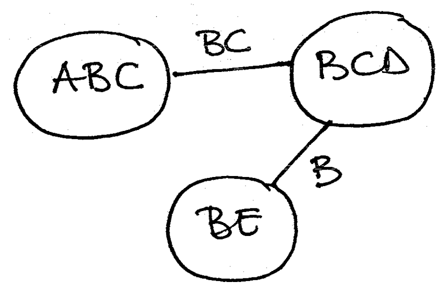
Remarks: • Junction tree structure is not unique

• Junction tree is a <u>max</u> spanning tree w.r.t. separator size

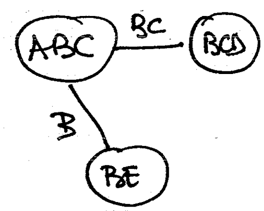
Ex:



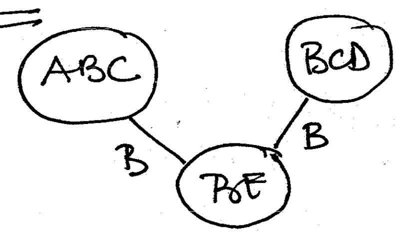
⇨



OR

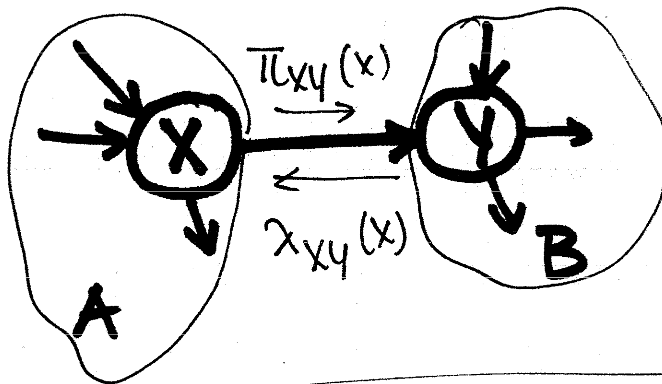


but <u>NOT</u>:



The Modified Pearl Algorithm

(After Peot & Schachter)



At each node: X:

$U = \{U_1 \dots U_m\}$ parents

$Y = \{Y_1 \dots Y_n\}$ children

$$\boxed{\begin{aligned} \pi_{XY}(x) &= P(X=x, e_A) \\ \lambda_{XY}(x) &= P(e_B \mid X=x) \\ &\text{"MESSAGES"} \end{aligned}}$$

$P_{X|U}$ conditional prob. table

$\lambda_X(x)$ = evidence for X

$= \begin{cases} \delta_x & \text{if } X = x \text{ observed} \\ 1 & \text{if } X \text{ unobserved} \end{cases}$

Local data tables : $\underset{P(X, e^+)}{\pi(x)}$, $\underset{P(e^- \mid X=x)}{\lambda(x)}$

Algorithm (Modified Pearl)

Initialize $\pi(x) = \pi_{XY_j}(x) = P(X=x)$ ← obtained by a propagation without evidence

$\lambda = 1$

Update at node X :

$$P(x) \propto \pi(x)\,\lambda(x)$$

$$\pi(x) = \sum_{u \in \Omega_U} P(X=x \mid U=u) \prod_{k=1}^{m} \pi_{U_k X}(u_k)$$

$$\lambda(x) = \lambda_X(x) \prod_{i=1}^{n} \lambda_{XY_j}(x)$$

[Modified Pearl Algorithm]

STAT 593A
Spring 05

2

To parent U_i

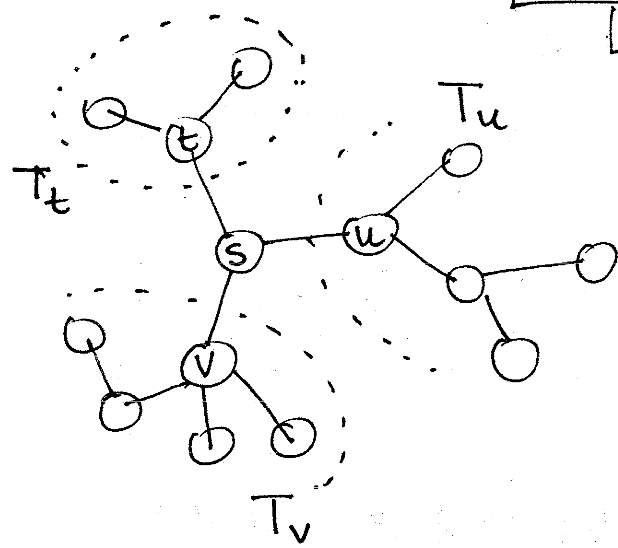
$$\lambda_{U_i X}(u_i) = \sum_x \lambda(x) \sum_{u' \in \Omega_{U \setminus U_i}} P(x \mid u', u_i) \prod_{k \neq i} \pi_{U_k X}(u_k)$$

To child Y_j

$$\pi_{X Y_j}(x) = \pi(x)\, \lambda_X(x) \prod_{k \neq j} \lambda_{X Y_k}(x)$$

Sum-Product Algorithm on Trees

(After Wainwright & Jordan)



Tree distribution

$$P_X = \frac{\prod_{st\in E} P_{st}(x_s,x_t)}{\prod_{s\in V} P_s(x_s)^{\deg s-1}}$$

"potentials" $\psi_{st} = P_{st}$

$\psi_s = 1/P_s^{\deg s-1}$ ...... enter evidence by multiplication

$$\Rightarrow \quad P_X(x) = \prod_{s\in V} \psi_s(x_s) \cdot \prod_{st\in E} \psi_{st}(x_s,x_t)$$

Wanted: $\mu_s(x_s) = \sum_{x':\, x'_s = x_s} P_X(x')$ Marginal of node s

Remarks:
Node s separates tree into s, and disjoint trees $\underbrace{T_u, T_v, T_t}_{n(s)}$

Denote $T_u = (V_u, E_u)$

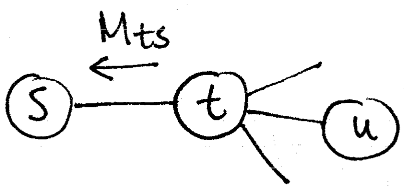
$$\mu_s(x_s) = \sum_{x':\, x'_s = x_s} \psi_s(x_s) \prod_{u\in n(s)} P_{X_{V_u}}(x'_{V_u})$$

$$= \psi_s(x_s) \cdot \prod_{u\in n(s)} \underbrace{\left[\sum_{x'_{V_u}} P_{X_{V_u}}(x'_{V_u}) \cdot \psi_{su}(x_s, x'_u)\right]}_{M^*_{us}(x_s)}$$

$M^*_{us}(x_s)$ ←--- "message" from T_u

Algorithm

Message updating: $M_{ts}(x_s) \leftarrow \sum_{x'_t} \left[\psi_{st}(x_s, x'_t)\, \psi_t(x'_t) \cdot \prod_{u \in N(t) \setminus s} M_{ut}(x'_t) \right]$ (*)



Message propagation:

- Each node s gets messages from all its neighbors, and sends messages to all neighbors (asynchronous, naive)
- OR: - Choose a root s
  - Collect messages recursively from leaf nodes to s
  - Distribute messages recursively from s toward leaf nodes

Sum-Product: in (*), each ~~co~~ message is computed as a product of messages received, followed by a sum.

Max-Product: Replace "Σ" by "max" in (*) $\Rightarrow$ Dynamic programming algorithm to compute the most likely configuration