Lecture 9: February 11, 2015
cs 573: Probabilistic Reasoning
Professor Nevatia
Spring 2015

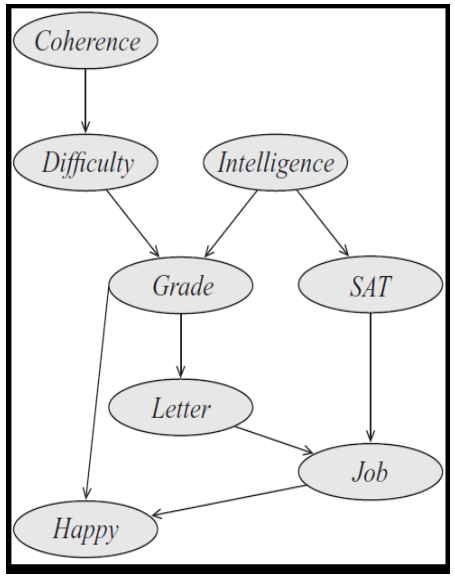
Review

- Assignment #3 will be posted today; due Monday Feb 25
- Last lecture:
  - Sum-Product Variable Elimination Algorithm
    - Applies to arbitrary directed and undirected graphs
- Today's objective
  - Analysis of VE algorithm
  - Graph-Theoretic implications
  - Cluster-tree algorithm

Dealing with Evidence

- Just work with reduced factors, where only terms corresponding to the evidence are used.
- VE will compute P($\boldsymbol{Y}$, $\boldsymbol{e}$) (joint probability)
  - To compute conditional probability, divide by P($\boldsymbol{e}$)
  - P($\boldsymbol{e}$) = $\sum_Y$ P(**y**,**e**)
  - Dividing by P($\boldsymbol{e}$) is equivalent to normalizing P($\boldsymbol{Y}$, $\boldsymbol{e}$)
- Example:
  - Next slide using the "student" network
  - Compute P (J | i^1, h^0)

Example: Inference with Evidence



Compute P (J | i^1, h^0)

Elimination order: C,D, G, S, L

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	$\phi_G[I=i^1](G,D), \phi_I[I=i^1](), \tau'_1(D)$	G,D	$\tau'_2(G)$
5'	G	$\tau'_2(G), \phi_L(L,G), \phi_H[H=h^0](G,J)$	G,L,J	$\tau'_5(L,J)$
6'	S	$\phi_S[I=i^1](S), \phi_J(J,L,S)$	J,L,S	$\tau'_6(J,L)$
7'	L	$\tau'_6(J,L), \tau'_5(J,L)$	J,L	$\tau'_7(J)$

Table 9.3 A run of sum-product variable elimination for $P(J,i^1,h^0)$

VE for Conditional Probabilities (9.2)

Algorithm 9.2 Using Sum-Product-VE **for computing conditional probabilities**

Procedure Cond-Prob-VE (
  $\mathcal{K}$, // A network over $\mathcal{X}$
  $\boldsymbol{Y}$, // Set of query variables
  $\boldsymbol{E} = \boldsymbol{e}$ // Evidence
)

1 $\Phi \leftarrow$ Factors parameterizing $\mathcal{K}$
2 Replace each $\phi \in \Phi$ by $\phi[\boldsymbol{E} = \boldsymbol{e}]$
3 Select an elimination ordering $\prec$
4 $\boldsymbol{Z} \leftarrow = \mathcal{X} - \boldsymbol{Y} - \boldsymbol{E}$
5 $\phi^* \leftarrow$ Sum-Product-VE$(\Phi, \prec, \boldsymbol{Z})$
6 $\alpha \leftarrow \sum_{\boldsymbol{y} \in Val(\boldsymbol{Y})} \phi^*(\boldsymbol{y})$
7 **return** α, ϕ^*

Complexity Analysis

- Let n be number of random variables, m number of initial factors
- At each step, we pick a variable to eliminate, say X_i. We multiply all factors containing X_i, to get a new factor ψ_i. Sum out X_i to get τ_i.
- Let N_i be number of entries in ψ_i , let $N_{max} = \max_i (N_i)$
- Total number of factors in $\mathbf{\Phi}$ is $m + n$ (m initial, n is number of τ_i.)
- Mulitplications: each factor is multiplied exactly once, cost of computing ψ_i is at most N_i .as each entry of ϕ is multiplied into exactly one entry of ψ_i .

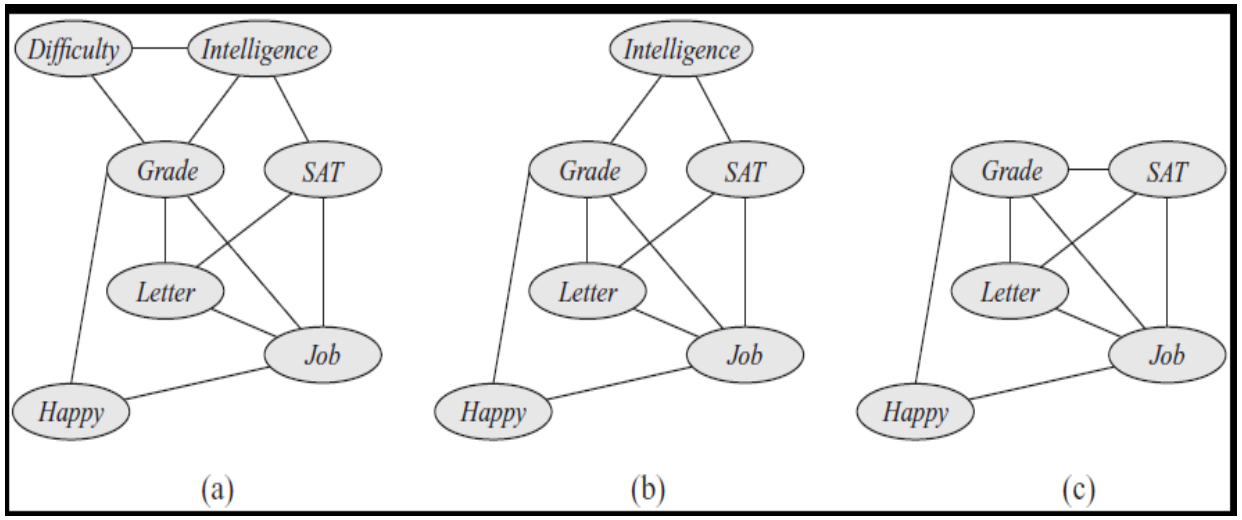
  Total number of multiplications is $\leq (n + m)\, N_{max} = O(mN_{max})$
- Additions: each entry in ψ_i is touched exactly once, so total is $\leq\ nN_{max}$.
- Hence, total complexity is $O(mN_{max})$
- Exponential growth is in size of the factors ψ_i

Graph Theoretic Analysis

- Graph-Theoretic analysis provides some insights
- Definition of scope of a *set* of factors, $\boldsymbol{\Phi}$: union of scope of the factors, ϕ, in the set.
- Let H_{Φ} be an undirected graph whose nodes are in scope of $\boldsymbol{\Phi}$; edges between pairs of nodes that are in scope of one of the factors ϕ in $\boldsymbol{\Phi}$.
  - Moralizes v-structures in a directed graph
- H_{Φ} is a fully connected sub-graph over the scope of each factor in $\boldsymbol{\Phi}$, hence a minimal I-map for distribution, P, defined by $\boldsymbol{\Phi}$.
- Given a context $\boldsymbol{E = e}$, we use the reduced factors and find a correspondingly reduced graph.
- (Formal definition: 9.4 and proposition 9.1 in the book.)

Elimination as Graph Transformation

- To eliminate a variable, X, we first create a single factor ψ that combines all factors containing X ; let $\boldsymbol{Y}$ be the other variables present in ψ. then we eliminate X to create τ that does not contain X. Let $\boldsymbol{\Phi}_X$ be the resulting set of factors.
- Consider new induced graph $H_{\boldsymbol{\Phi}_X}$. It will contain edges between all the variables in $\boldsymbol{Y}$, some may have been already present, new ones are called *fill* edges. X and all edges to it are removed.

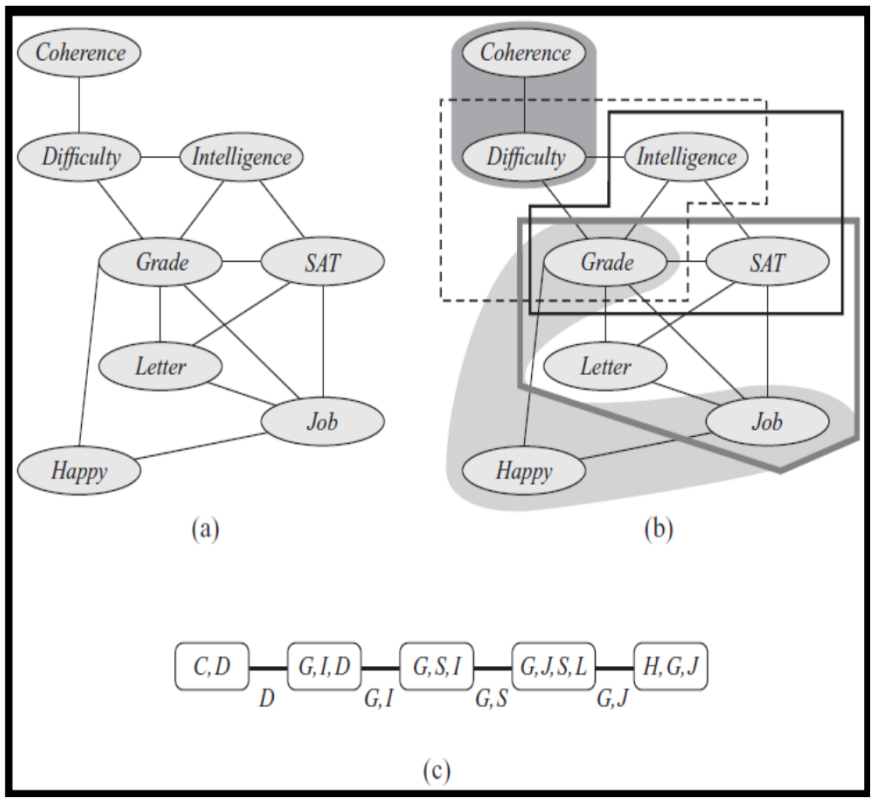


Remove D then I.

Note fill edge G - S

Induced Graph

- Union of all graphs resulting from different steps of the elimination algorithm
- $I_{\Phi,\prec}$ is the induced graph, $\prec$ is the elimination ordering
  - Two variables are connected in this graph if they both appear in some factor ψ.



(b): cliques in induced graph

(c): Clique tree from (b)

- Each node corresponds to a clique in the original graph
- Links if nodes contain common variables; this set is called *sepset*
- Formal def: 4.17

Induced Graph Properties

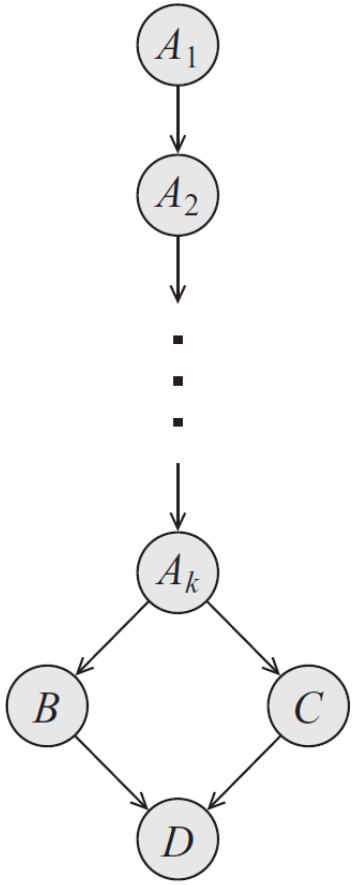
- Scope of every factor generated in elimination process is a clique in $I_{\Phi,\prec}$
- Every maximal clique in $I_{\Phi,\prec}$ is in scope of some intermediate factor.
- For the given example, {G, L, S, J} corresponds to ψ_5 in step 5
- Every induced graph is chordal (triangulated)
- Eliminating edges in a chordal graph does not add *fill* edges
- Sizes of maximal cliques in induced graph depend on elimination ordering
- Define *width* of induced graph to be size of largest clique – 1
- Tree width of a graph (original) is the minimal induced width by any ordering.
  - Minimal induced width gives a bound on best case performance for VE.
- Finding optimal elimination ordering is NP-hard.

Choosing Elimination Order

- Chordal graph:
  - Eliminate nodes that are in a leaf clique first
  - New leaf cliques after elimination of some variables
- Heuristic cost of a variable:
  - **Min-neighbors**: number of neighbors
  - **Min-Weight**: product of weights (cardinality) of its neighbors
  - **Min-Fill**: Number of edges to be added due to elimination
  - **Weighted-min-fill**: Sum of weights of fill edges, weight of an edge is the product of weights of its constituent vertices
  - The last two reported to work best (empirically)
- Search may be conducted deterministically or stochastically (sample from the cost distribution)

Conditioning

- Early algorithms applied only to polytrees (only one path between any pair of nodes) and could not work in graphs with loops.
- Solution was to take one or more nodes in the loop (to constitute a *cutset*) and *condition* them (assign them fixed values), effectively cutting them out of the graph (set of such nodes is also called *cutset*). Repeat the inference for all possible assignments of the nodes in the cutset and sum them.
- Condition on A_k (choose a value)
  - Inference over A_1 thru A_{k-1} would be repeated
- Condition on B
  - Inference over A_1 thru A_k would be repeated



Conditioning

- VE algorithm can work with graphs containing loops
  - However, the scope of intermediate factors can become large
  - Storage and computation requirements can grow
- Conditioning can reduce the size of the resulting factors.
  - However, it can be shown that computation is not reduced; in fact, may be increased as some computations have to be repeated.
- Conditioning may be considered to offer a space-time trade-off
- Naïve algorithm for making inferences, by computing the joint probability distribution entries, is essentially one of conditioning all nodes, except the one whose probability is to be inferred.
- VE and conditioning steps can be mixed and applied alternately
  - Condition a variable, eliminate some, condition some more *etc*
- We omit details of conditioning (9.5) and section 9.6

VE with Structured CPDs

- One approach would be to convert Structured CPDs (tree, rules) to the usual, table based CPDs
- However, as structured CPDs may contain fewer parameters, it is preferable to work with them directly.
- KF book describes a modified version of VE for rule CPDs (sec 9.6)
- We skip this section

Available Packages

- Survey by Kevin Murphy
  - http://www.cs.ubc.ca/~murphyk/Software/bnsoft.html
- MSBNx
  - Interactive BN inference software from Microsoft
  - Has a graphical interface to enter nodes, arcs, CPDs
  - Seems easy to use but no support is available from TA or Professor
  - May use to verify your numerical answers
- GeNIe: from U. of Pittsburg
- LIBDAI: free library for C++
- BNT (Kevin Murphy), widely used, MATLAB only
- HUGIN: powerful and popular commercial software package

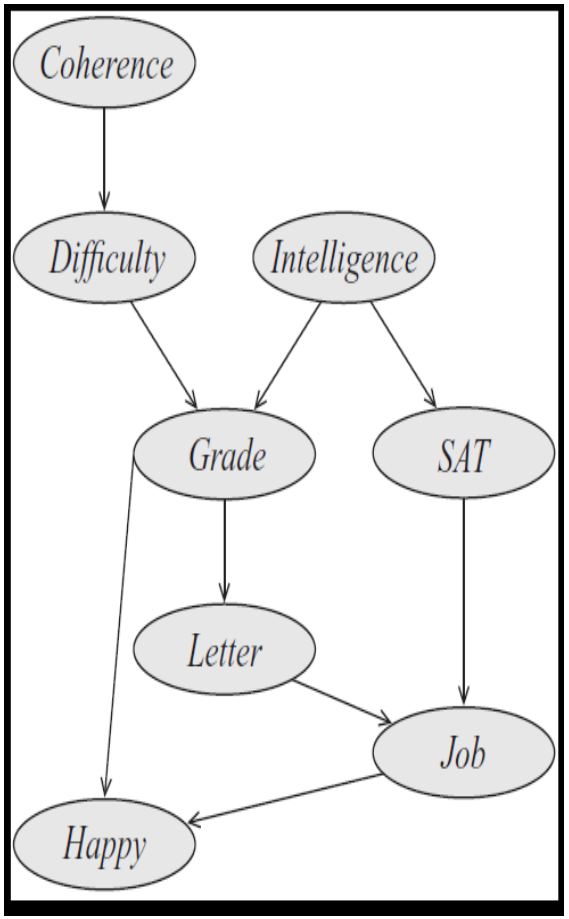
Inference using Clique Trees

- Variable elimination algorithm provides distribution of one variable at a time. We can re-run multiple times to get distribution of all variables but much of the computation would be repeated.
- Working with *clique trees* will allow us to get distribution of all variables efficiently
- Topics:
  - How to convert original networks to clique trees?
  - Inference algorithm for clique trees
  - Generalization to cluster (clique) graphs

Cluster Graph

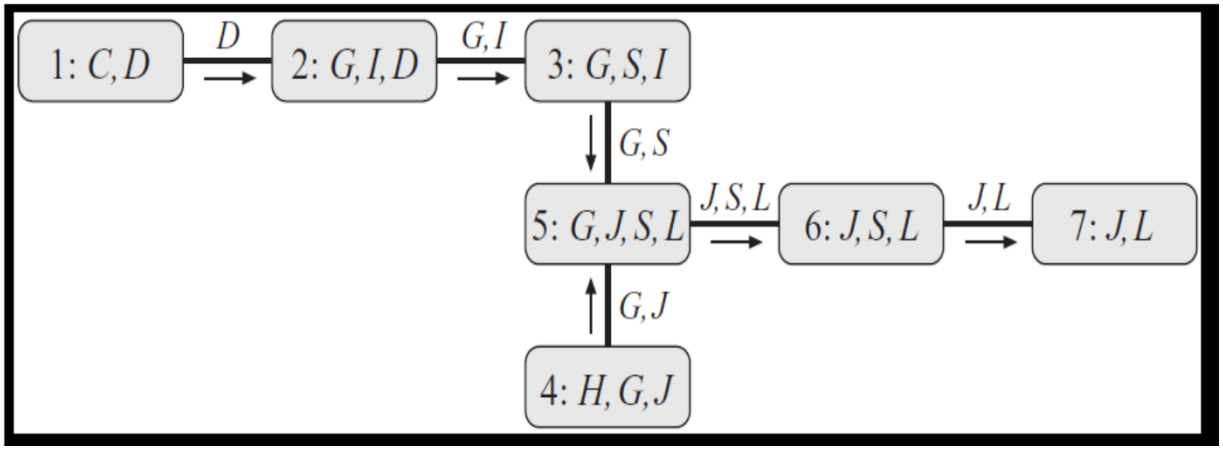
- *U*: graph over a set of factors Φ, ***X*** is set of nodes
- A *node* in this graph is a "cluster" (subset of) variables, say $\mathbf{C}_i$
- **Family preserving**: Each factor ϕ must be associated with some cluster, say $\mathbf{C}_i$, called $\alpha(\phi)$. scope $[\phi] \subseteq \mathbf{C}_i$
- Each *edge* between two nodes, say $\mathbf{C}_i$ and $\mathbf{C}_j$, is associated with a set of nodes, called a *sepset*, $\mathbf{S}_{i,j}$, $\mathbf{S}_{i,j} \subseteq \mathbf{C}_i \cap \mathbf{C}_j$.
- Variable elimination algorithm generates a cluster graph
  - Cluster for each factor ψ_i.
  - Edge between $\mathbf{C}_i$ and $\mathbf{C}_j$ when τ_i is used in computation of τ_j.
- Example Fig 10.1

Cluster Graph from VE Steps

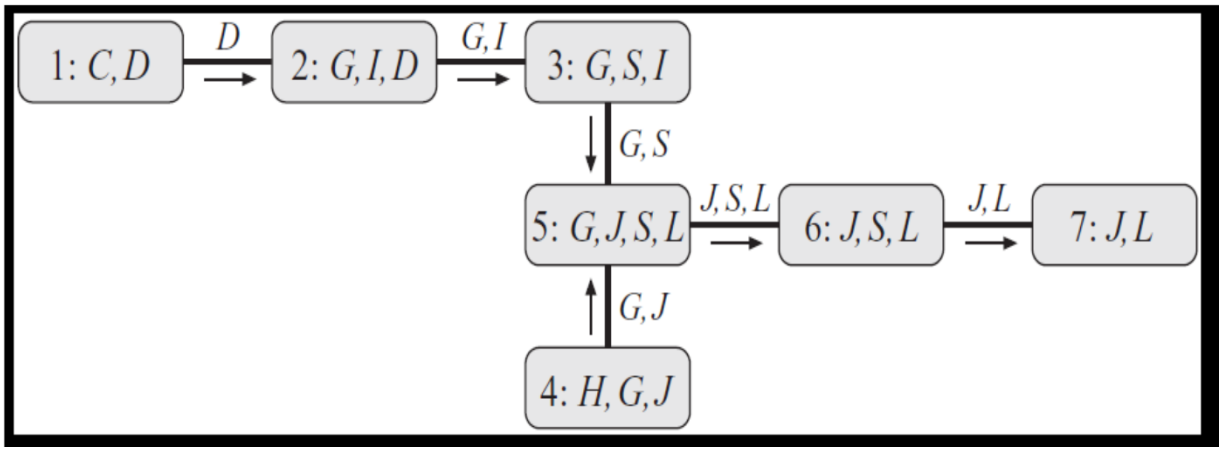


Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C), \phi_D(D,C)$	C,D	$\tau_1(D)$
2	D	$\phi_G(G,I,D), \tau_1(D)$	G,I,D	$\tau_2(G,I)$
3	I	$\phi_I(I), \phi_S(S,I), \tau_2(G,I)$	G,S,I	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	H,G,J	$\tau_4(G,J)$
5	G	$\tau_4(G,J), \tau_3(G,S), \phi_L(L,G)$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

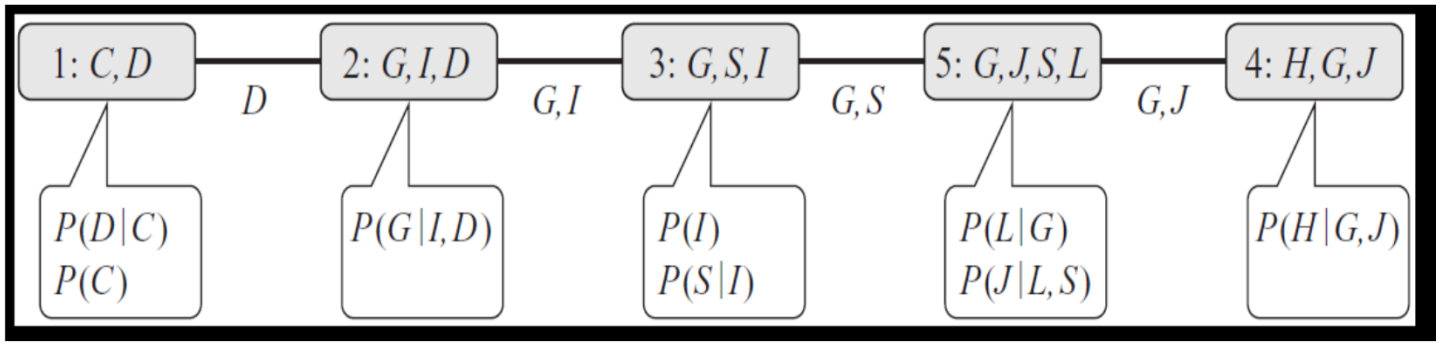
Table 9.1 A run of variable elimination for the query $P(J)$



Simplify the Cluster Graph



Eliminate cliques that are not maximal
(results in a tree for this example)



Next Class

- Read sections 10.2 and 10.3 of the KF book