Lecture 3: Exact Inference

Probabilistic Graphical Models, Koller and Friedman:

- Chapters 9 and 10:

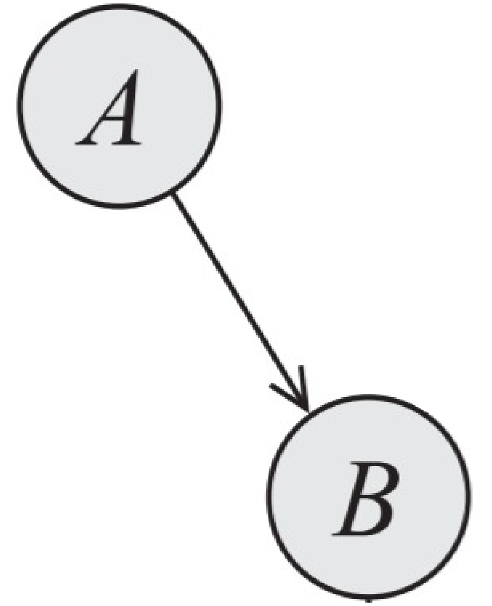
- Variable Elimination, Message Passing, Factor Graphs from DAGs, Sum Product Algorithm, Belief propagation, Clique Graphs/Trees, Inference with evidence, Junction Tree Algorithm

Bayes Net

$P(A)$

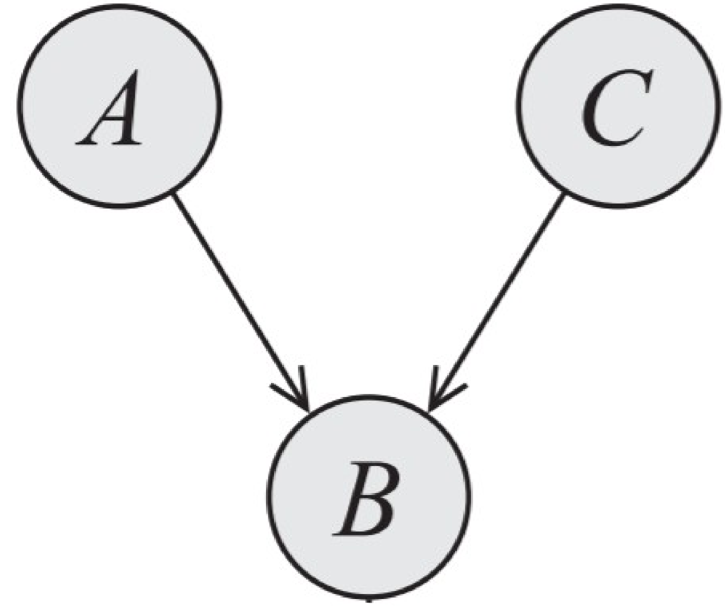
Bayes Net

$P(A,B)=P(A)P(B|A)$



Bayes Net

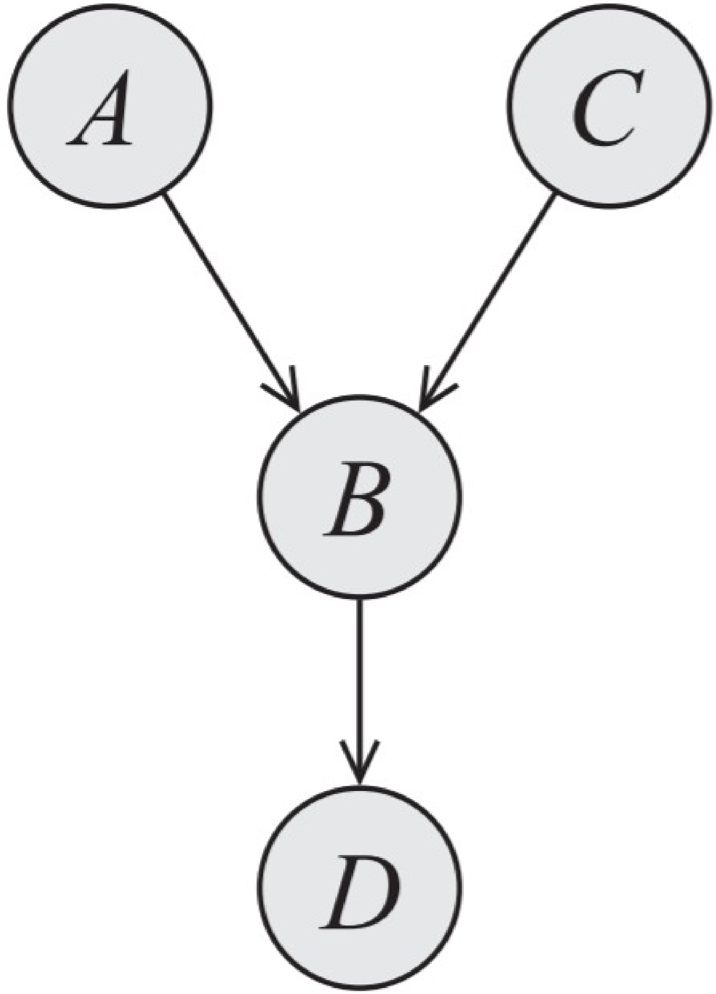
$$P(A,B,C)=P(A)P(C)P(B|A,C)$$



Bayes Net

$$P(A,B,C,D) =$$

$$P(A)P(C)P(B|A,C)P(D|B)$$



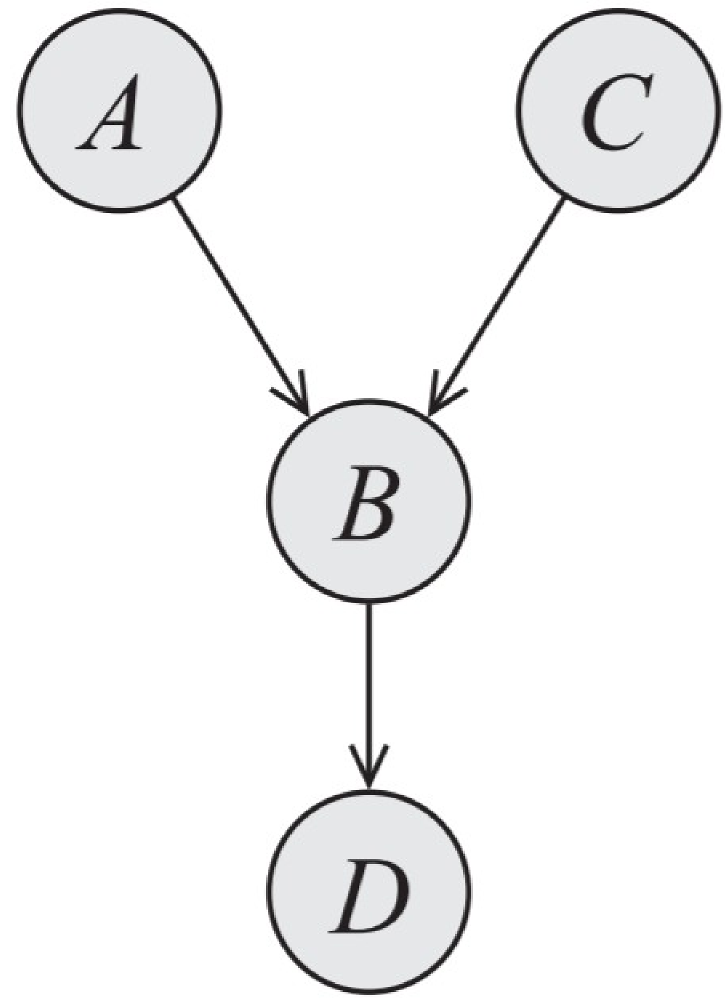
Bayes Net

$P(A,B,C,D) =$

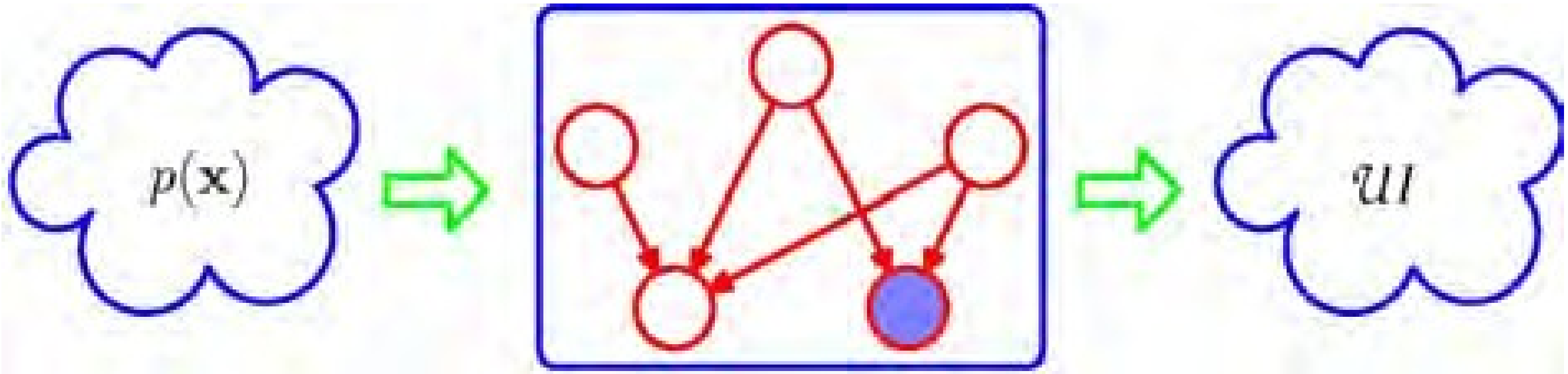
$P(A)P(C)P(B|A,C)P(D|B)$

$\Rightarrow \mathcal{UF}$

- $A \perp C$? *Yes*
- $A \perp B$? *No*
- $A \perp C \mid B$? *No*
- … $\Rightarrow \mathcal{UI}$



Filter View of a PGM



- Let $\mathcal{UI}$ denote the distributions that can pass
- ie. those that satisfy all conditional independence statements
- Let $\mathcal{UF}$ denote the distributions with factorization over cliques (also for undirected graphs, MRF)
- Hammersley-Clifford says for MRF: $\mathcal{UI} = \mathcal{UF}$ (except if some P=0)
- Similar result for DAG, Theorem 3.1

Figure from the lectures of: Bjoern Andres and Bernt Schiele

Directed vs. Undirected Graphs

Picture is in independancy, '$\mathcal{UI}$-space'

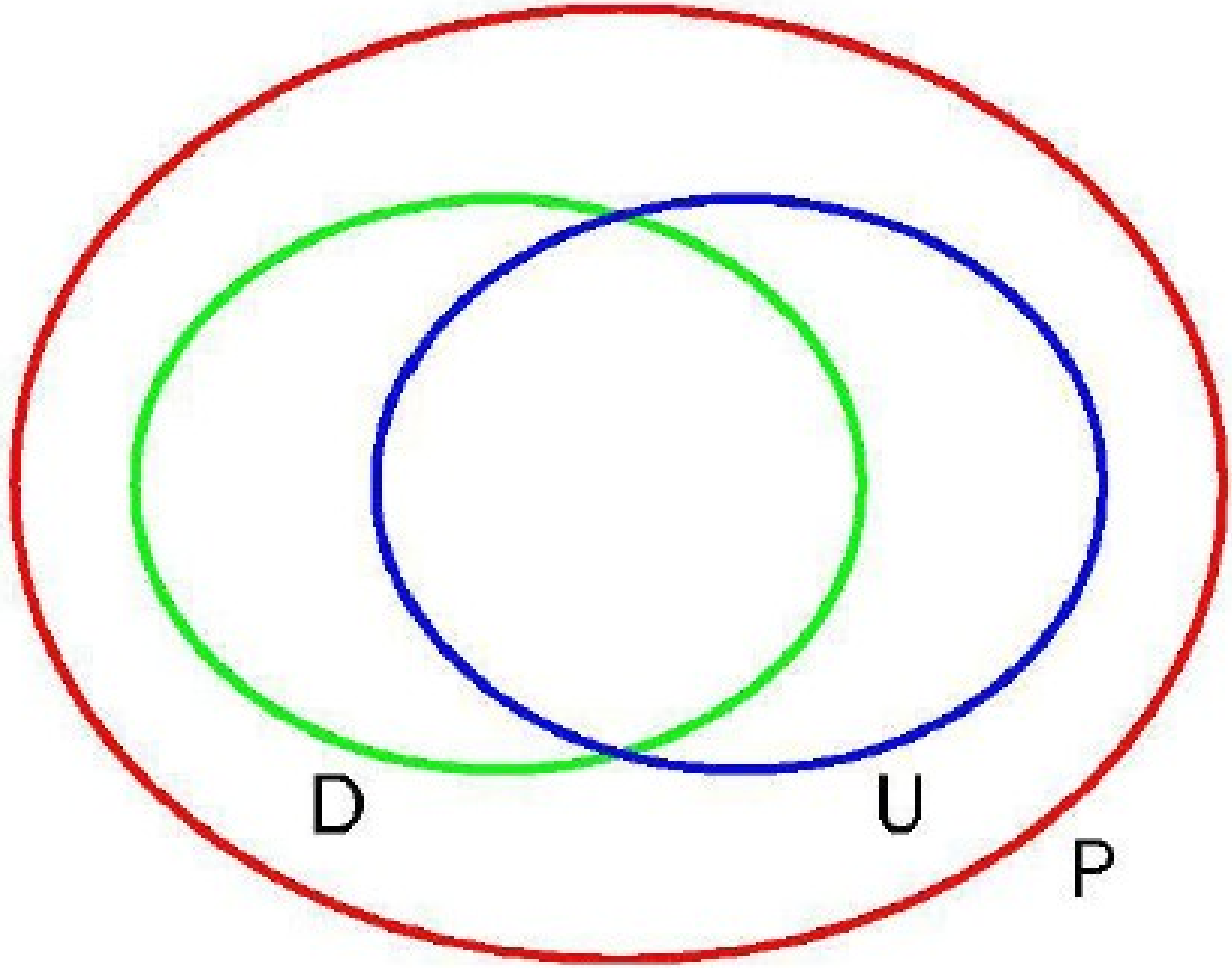
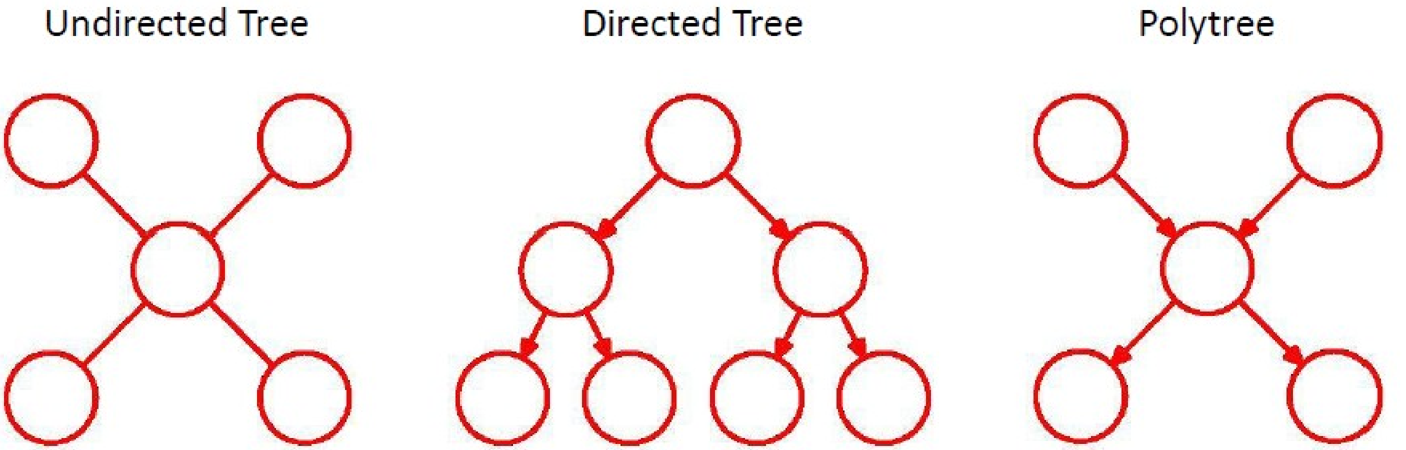


Figure from Bishop- Pattern Recognition and ML

Trees



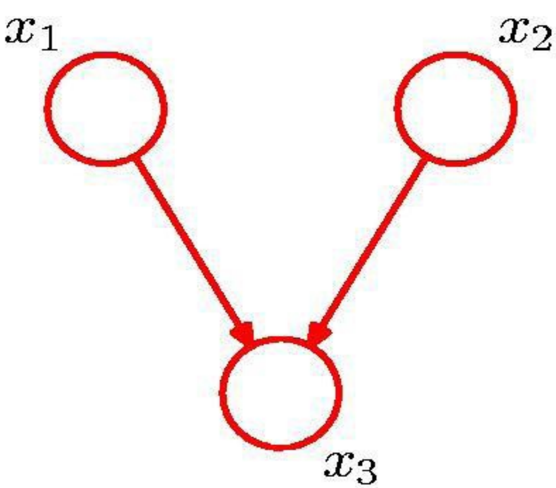
Tree - No Cycles

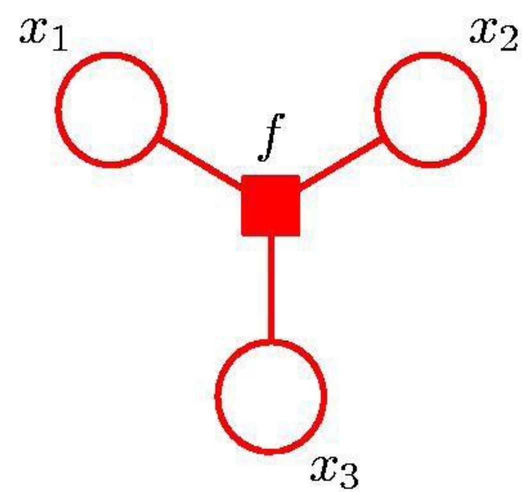
Directed Rooted Tree

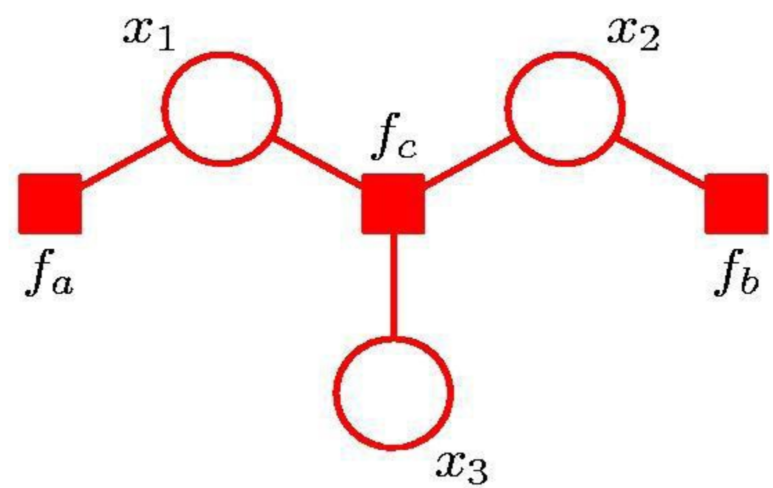
w/o arrows = tree

Figure from Bishop- Pattern Recognition and ML

Factor Graphs from BN







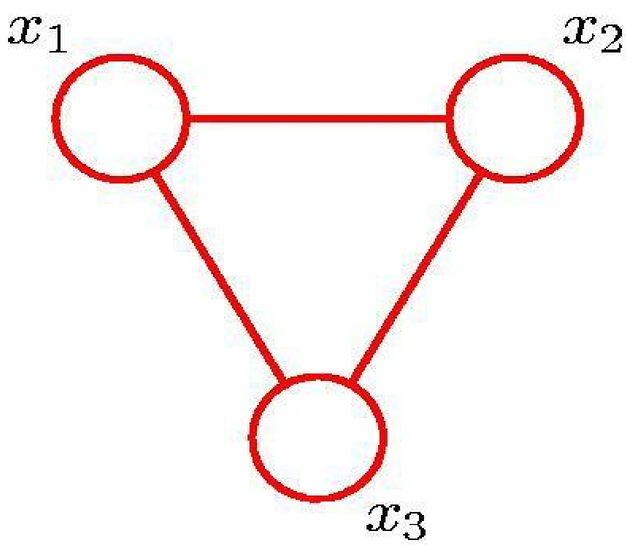
$$
\begin{aligned}
p(\mathbf{x}) \;=\; & p(x_1)p(x_2) \\
& p(x_3|x_1, x_2)
\end{aligned}
$$

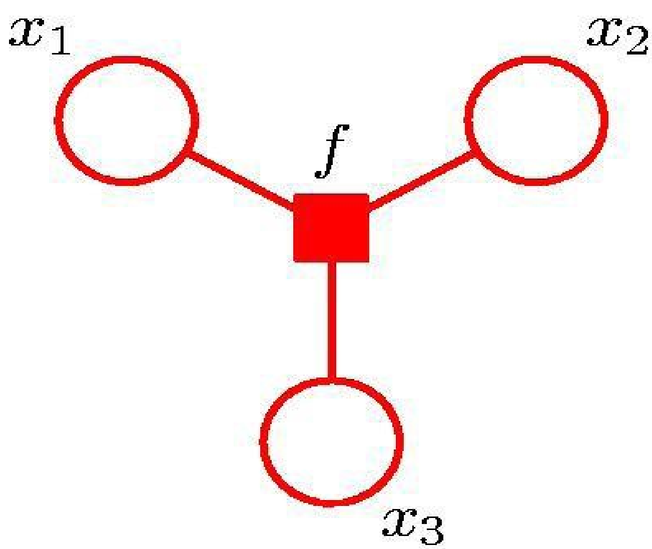
$$
\begin{aligned}
f_a(x_1) &= p(x_1) \\
f_b(x_2) &= p(x_2) \\
f_c(x_1, x_2, x_3) &= p(x_3|x_1, x_2)
\end{aligned}
$$

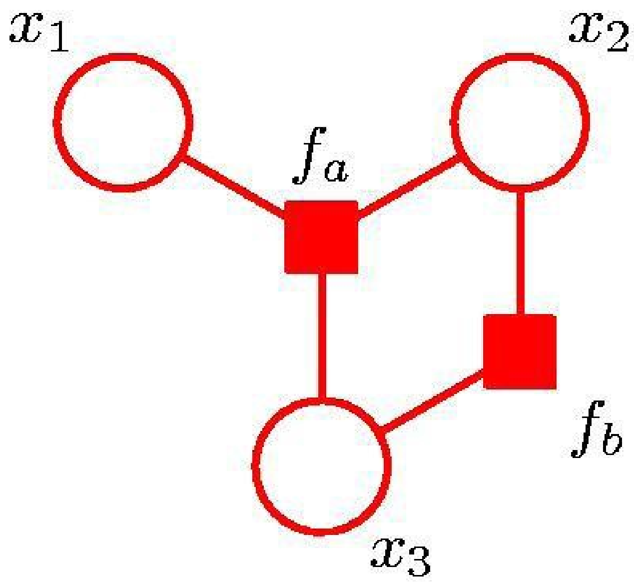
$$
\begin{aligned}
f(x_1, x_2, x_3) = & \\
& p(x_1)p(x_2)p(_3|x_1, x_2)
\end{aligned}
$$

Figure from Bishop- Pattern Recognition and ML

Factor Graphs from Undirected Graphs







$$\psi(x_1, x_2, x_3)$$

$$f(x_1, x_2, x_3) = \psi(x_1, x_2, x_3)$$

$$f_a(x_1, x_2, x_3) f_b(x_2, x_3) = \psi(x_1, x_2, x_3)$$

Figure from Bishop- Pattern Recognition and ML

Inference What?

Inference = computing Z? *Or is it all about avoiding computing* Z*?*

(or Z with some evidence observed)

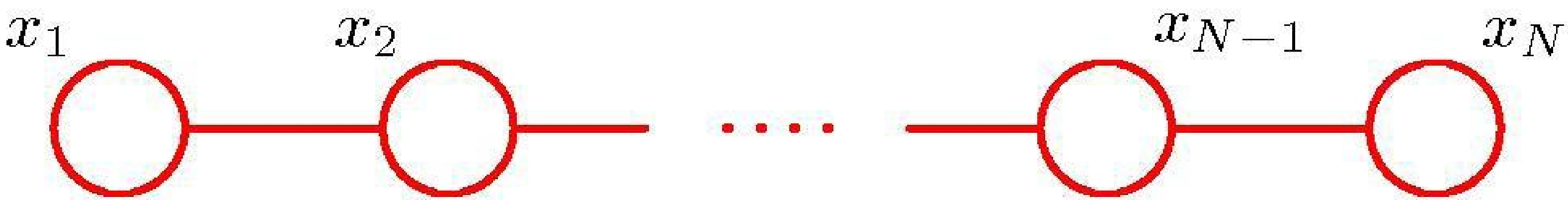
$p(X)$=?

$p(X \mid E)$=?

$\text{argmax}_x\, p(X)$=?

$\text{argmax}_x\, p(X|E)$=?

Inference on a Markov chain

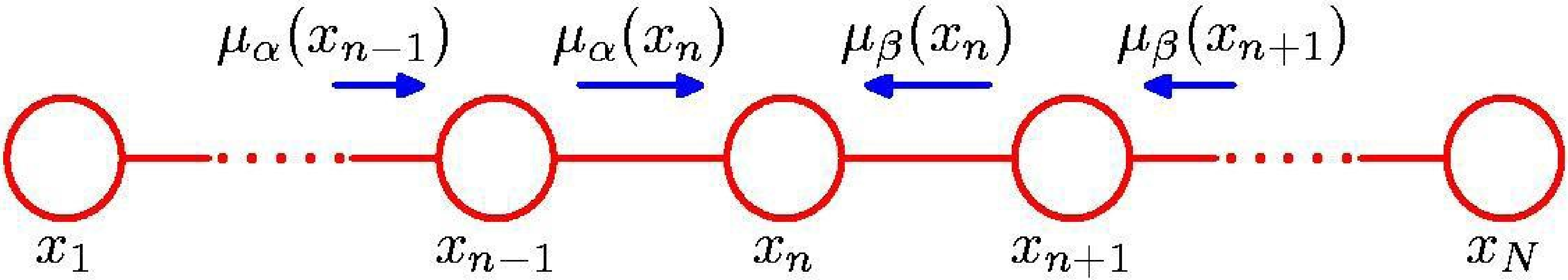


$$p(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\cdots\psi_{N-1,N}(x_{N-1}, x_N)$$

$$p(x_n) = \sum_{x_1}\cdots\sum_{x_{n-1}}\sum_{x_{n+1}}\cdots\sum_{x_N} p(\mathbf{x})$$

Figure from Bishop- Pattern Recognition and ML

Inference on a Markov chain

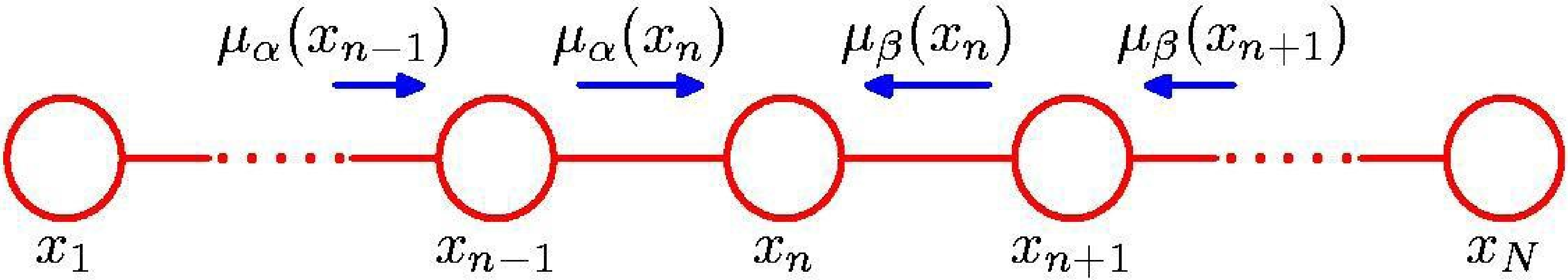


$$p(x_n) = \frac{1}{Z} \underbrace{\left[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \cdots \left[\sum_{x_1} \psi_{1,2}(x_1, x_2) \right] \cdots \right]}_{\mu_\alpha(x_n)}$$

$$\underbrace{\left[\sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \cdots \left[\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N) \right] \cdots \right]}_{\mu_\beta(x_n)}$$

Figure from Bishop- Pattern Recognition and ML

Inference on a Markov chain

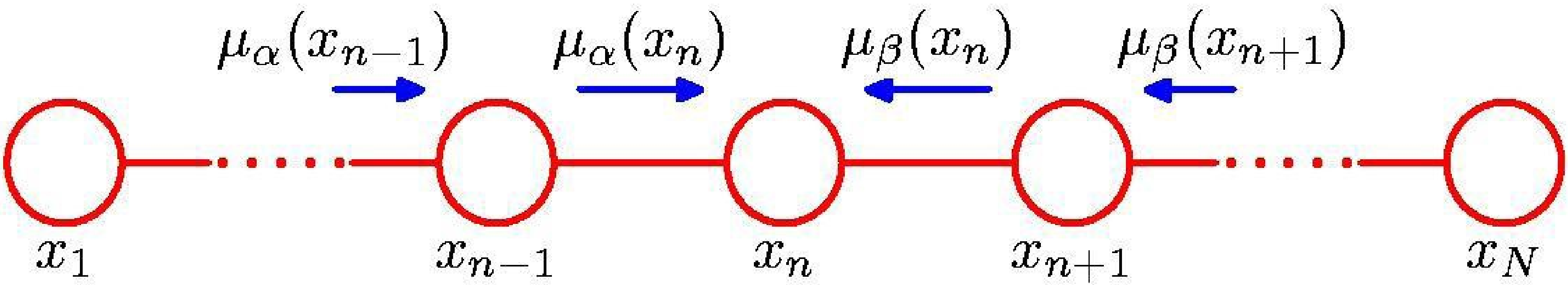


$$
\begin{aligned}
\mu_\alpha(x_n) &= \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \left[\sum_{x_{n-2}} \cdots \right] \\
&= \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \mu_\alpha(x_{n-1}).
\end{aligned}
$$

$$
\begin{aligned}
\mu_\beta(x_n) &= \sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \left[\sum_{x_{n+2}} \cdots \right] \\
&= \sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \mu_\beta(x_{n+1}).
\end{aligned}
$$

Figure from Bishop- Pattern Recognition and ML

Inference on a Markov chain



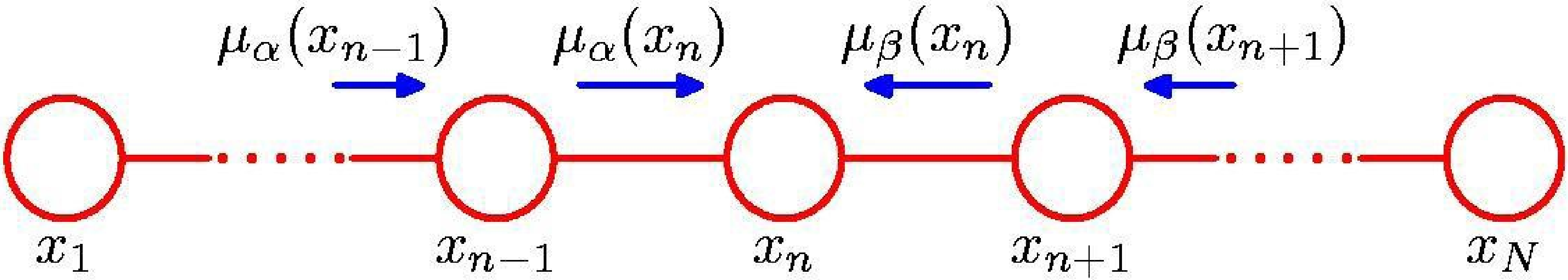
$$\mu_\alpha(x_2) = \sum_{x_1} \psi_{1,2}(x_1, x_2)$$

$$\mu_\beta(x_{N-1}) = \sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)$$

$$Z = \sum_{x_n} \mu_\alpha(x_n)\mu_\beta(x_n)$$

Figure from Bishop- Pattern Recognition and ML

Inference on a Markov chain



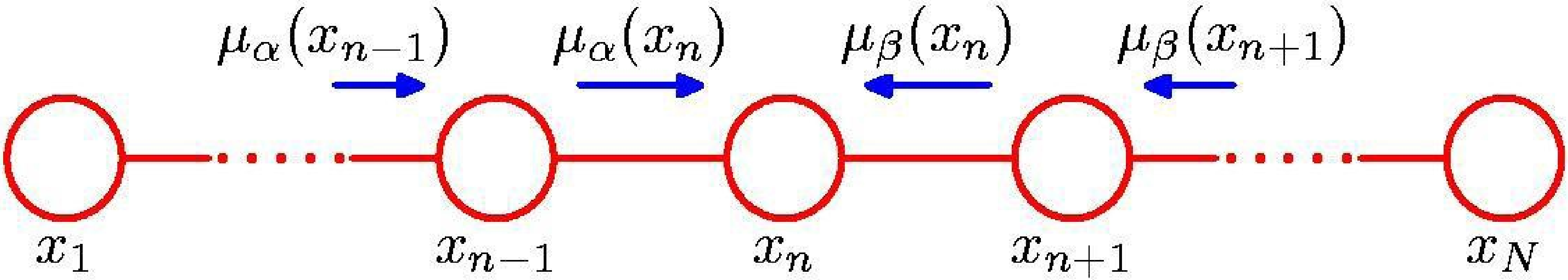
To compute local marginals:

- Compute and store all forward messages: $\mu_\alpha(x_n)$
- Compute and store all backward messages: $\mu_\beta(x_n)$
- Compute Z at any node x_m.
- Compute for all variables required:

$$p(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$$

Figure from Bishop- Pattern Recognition and ML

Inference on a Markov chain



- This is called in the book Sum Product
- Also called Message Passing
- Also Called Variable Elimination
- Point is one gets the marginals for a chosen node

$$p(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$$

Figure from Bishop- Pattern Recognition and ML

Variable elimination

$$Z = \sum_{abcd} \phi_1(a,b)\, \phi_2(b,c)\, \phi_3(c,d)$$

a_0	b_0	1
a_0	b_1	5
a_1	b_0	10
a_1	b_1	1

X

b_0	c_0	3
b_0	c_1	2
b_1	c_0	0
b_1	c_1	1.3

X

c_0	d_0	1.1
c_0	d_1	5
c_1	d_0	1
c_1	d_1	1

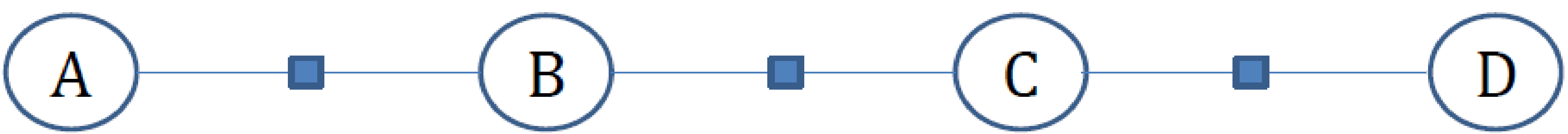


Figure from Lectures of: Veselin Stoyanov, Alexandre Klementiev and Shane Bergsman

The Dumb Way

a_0	b_0	1
a_0	b_1	5
a_1	b_0	10
a_1	b_1	1

X

b_0	c_0	3
b_0	c_1	2
b_1	c_0	0
b_1	c_1	1.3

a_0	b_0	c_0	3
a_0	b_0	c_1	2
a_0	b_1	c_0	0
a_0	b_1	c_1	6.5
a_1	b_0	c_0	30
a_1	b_0	c_1	20
a_1	b_1	c_0	0
a_1	b_1	c_1	1.3

X

c_0	d_0	1.1
c_0	d_1	5
c_1	d_0	1
c_1	d_1	1

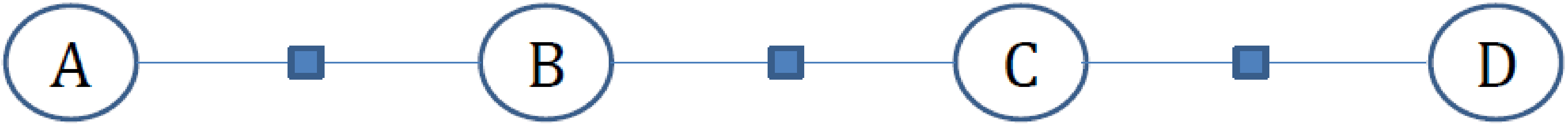


Figure from Lectures of: Veselin Stoyanov, Alexandre Klementiev and Shane Bergsman

Variable elimination

$$Z = \sum_{abcd} \phi_1(a,b)\ \phi_2(b,c)\ \phi_3(c,d)$$

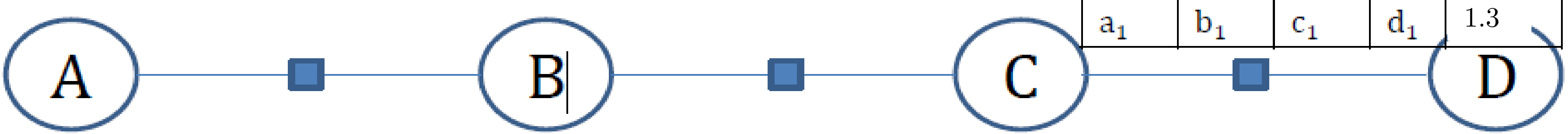
a_0	b_0	c_0	3
a_0	b_0	c_1	2
a_0	b_1	c_0	0
a_0	b_1	c_1	6.5
a_1	b_0	c_0	30
a_1	b_0	c_1	20
a_1	b_1	c_0	0
a_1	b_1	c_1	1.3

X

c_0	d_0	1.1
c_0	d_1	5
c_1	d_0	1
c_1	d_1	1

=

a_0	b_0	c_0	d_0	3.3
a_0	b_0	c_0	d_1	15
a_0	b_0	c_1	d_0	2.2
a_0	b_0	c_1	d_1	2
a_0	b_1	c_0	d_0	2
a_0	b_1	c_0	d_1	0
a_0	b_1	c_1	d_0	0
a_0	b_1	c_1	d_1	6.5
a_1	b_0	c_0	d_0	6.5
a_1	b_0	c_0	d_1	33
a_1	b_0	c_1	d_0	150
a_1	b_0	c_1	d_1	20
a_1	b_1	c_0	d_0	20
a_1	b_1	c_0	d_1	0
a_1	b_1	c_1	d_0	0
a_1	b_1	c_1	d_1	1.3



$Z = 260.9$

Figure from Lectures of: Veselin Stoyanov, Alexandre Klementiev and Shane Bergsman

Variable elimination

$$\sum_a \phi_1(a,b)$$

a_0	b_0	1
a_0	b_1	5
a_1	b_0	10
a_1	b_1	1

b_0	11
b_1	6

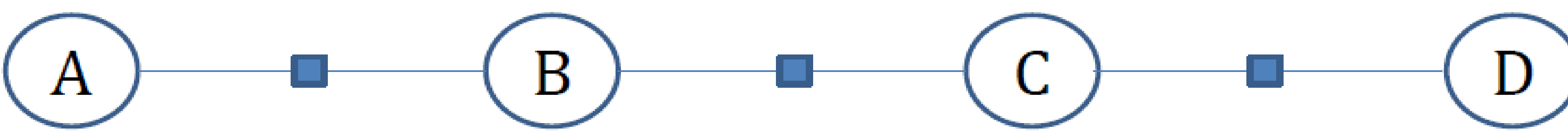
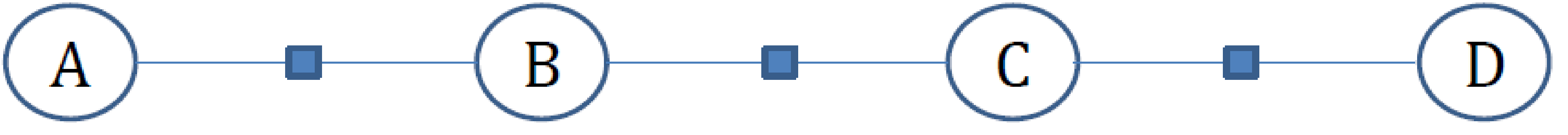


Figure from Lectures of: Veselin Stoyanov, Alexandre Klementiev and Shane Bergsman

Variable elimination

$\sum_a \phi_1(a,b)$

b_0	11
b_1	6



Variable elimination

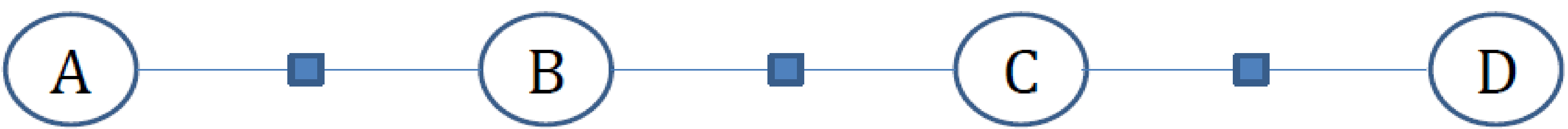
$$\left(\sum_a \phi_1(a,b)\right) \phi_2(b,c)$$

b_0	11
b_1	6

X

b_0	c_0	3
b_0	c_1	2
b_1	c_0	0
b_1	c_1	1.3

b_0	c_0	33
b_0	c_1	22
b_1	c_0	0
b_1	c_1	7.8



Variable elimination

$$\sum_b\left(\left(\sum_a \phi_1(a,b)\right) \phi_2(b,c)\right)$$

b_0	11
b_1	6

X

b_0	c_0	3
b_0	c_1	2
b_1	c_0	0
b_1	c_1	1.3

b_0	c_0	33
b_0	c_1	22
b_1	c_0	0
b_1	c_1	7.8

c_0	33
c_1	29.8

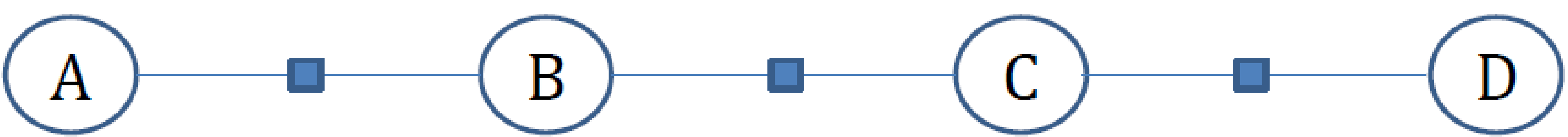


Figure from Lectures of: Veselin Stoyanov, Alexandre Klementiev and Shane Bergsman

Variable elimination

$$\sum_b \left(\left(\sum_a \phi_1(a,b) \right) \phi_2(b,c) \right)$$

c_0	33
c_1	29.8

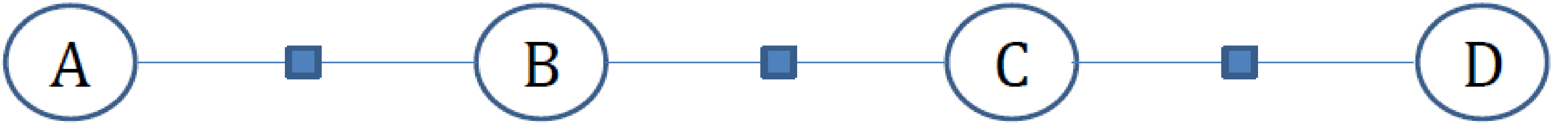


Figure from Lectures of: Veselin Stoyanov, Alexandre Klementiev and Shane Bergsman

Variable elimination

$$\sum_c((\sum_b((\sum_a \phi_1(a,b))\ \phi_2(b,c)))\ \phi_3(c,d))$$

c_0	33
c_1	29.8

XΣ

c_0	d_0	1.1
c_0	d_1	5
c_1	d_0	1
c_1	d_1	1

d_0	66.1
d_1	194.8

Z= 260.9

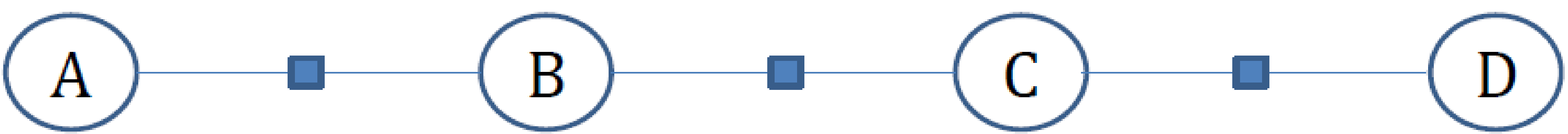


Figure from Lectures of: Veselin Stoyanov,
Alexandre Klementiev and Shane Bergsman

Variable Elimination

- Actually if we were smarter in writing the tables as matrices this ends up looking like multiplying a vector of all ones by a series of matrices.
- Show on board….

a_0	b_0	1
a_0	b_1	5
a_1	b_0	10
a_1	b_1	1

X

b_0	c_0	3
b_0	c_1	2
b_1	c_0	0
b_1	c_1	1.3

X

c_0	d_0	1.1
c_0	d_1	5
c_1	d_0	1
c_1	d_1	1

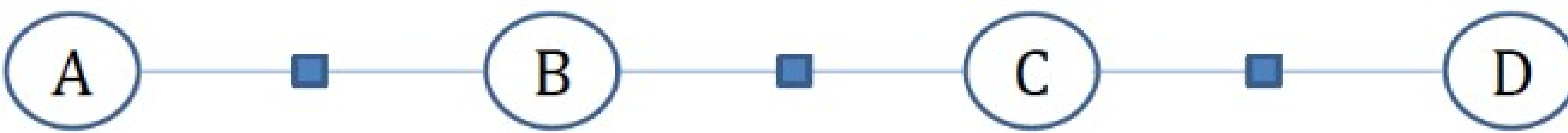


Figure from Lectures of: Veselin Stoyanov, Alexandre Klementiev and Shane Bergsman

Sum-Product Algorithm

Objective:

- to obtain an efficient, exact inference algorithm for finding marginals;
- in situations where several marginals are required, to allow computations to be shared efficiently.
- Key idea: Distributive Law $ab + ac = a(b + c)$

  2 mult + 1 add → 1 mult. + 1 add

Clique Graphs

- Imagine marginalizing in order: (CD)(H)IGLSJ

  Will walk through later.

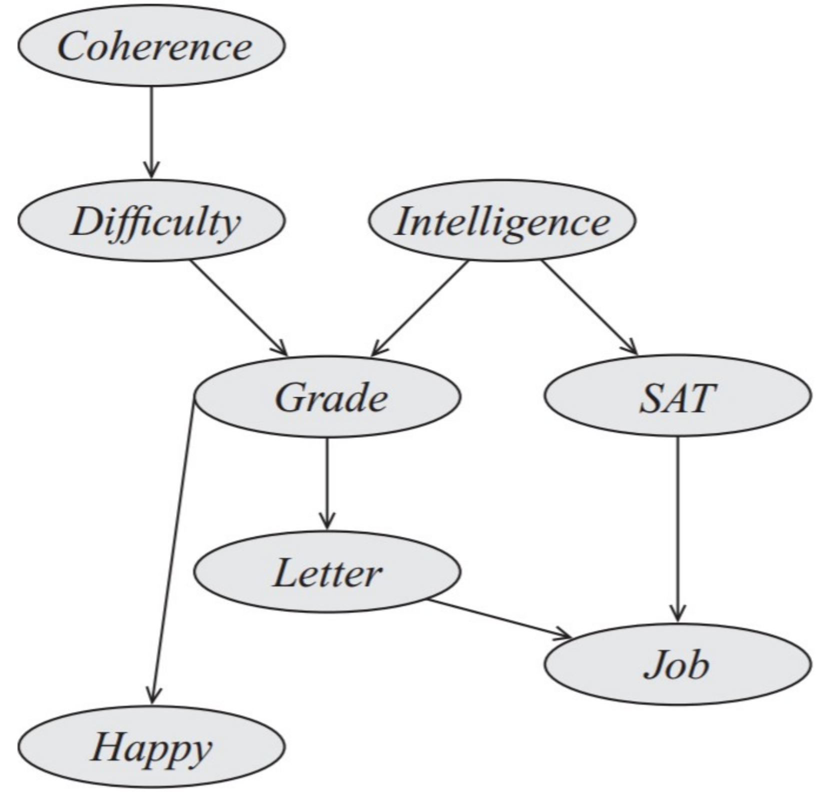
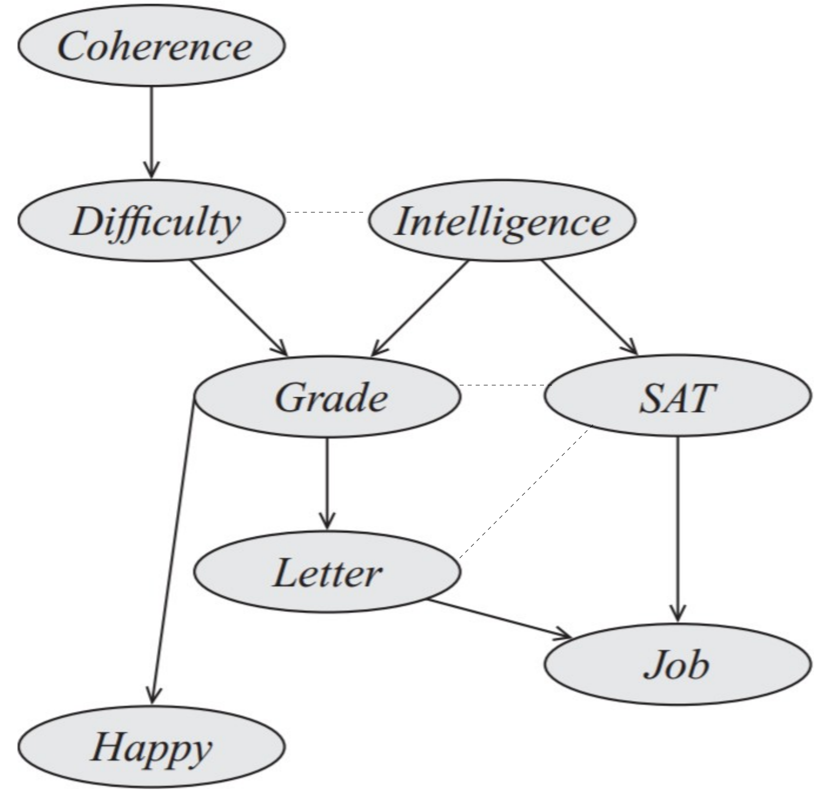


Figure: Koller and Freidman – PGM Principles and Techniques

Clique Graphs

- Imagine marginalizing in order: (CD)(H)IGLSJ

  Will walk through later.



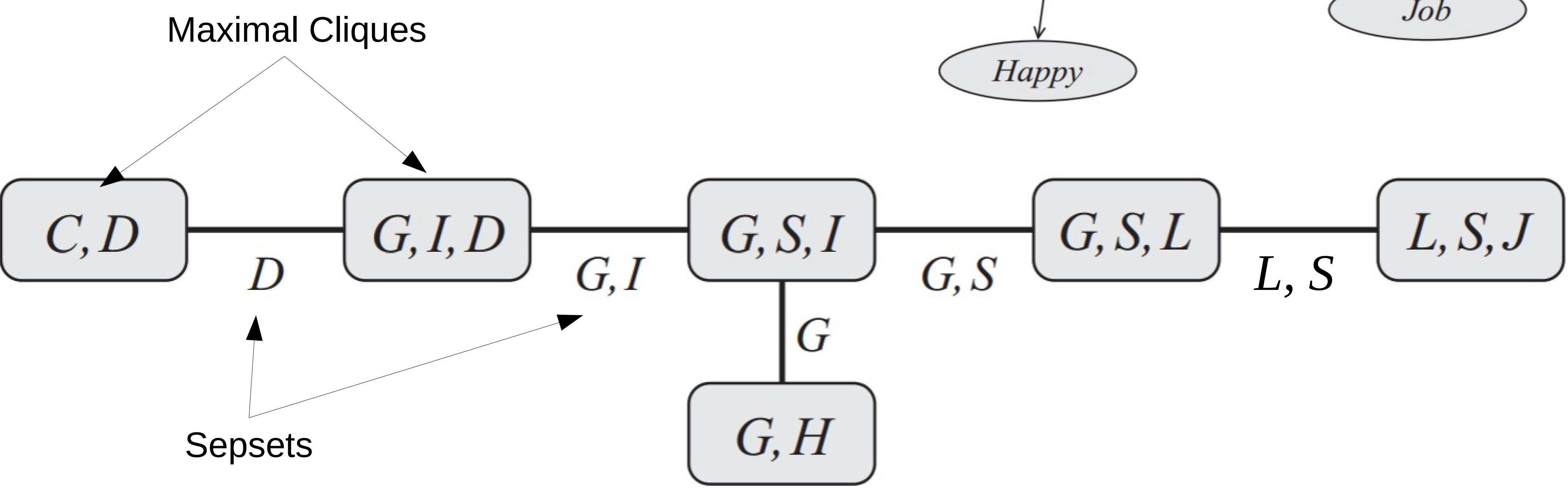
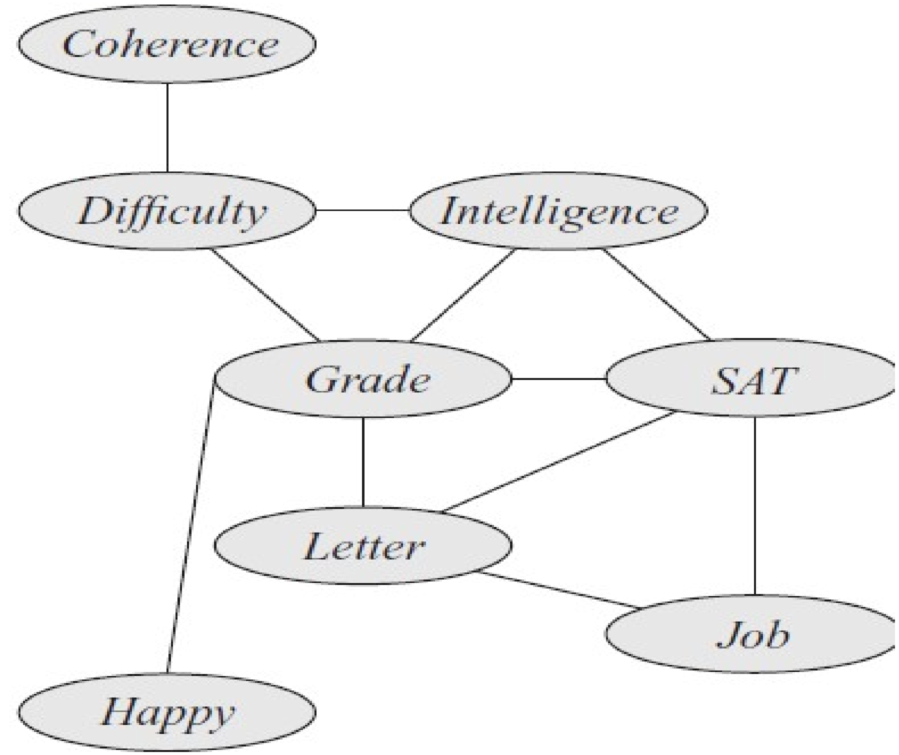


Figure: Koller and Freidman – PGM Principles and Techniques

Clique Graphs

- Imagine marginalizing in order: (CD)(H)IGLSJ

  Will walk through later.



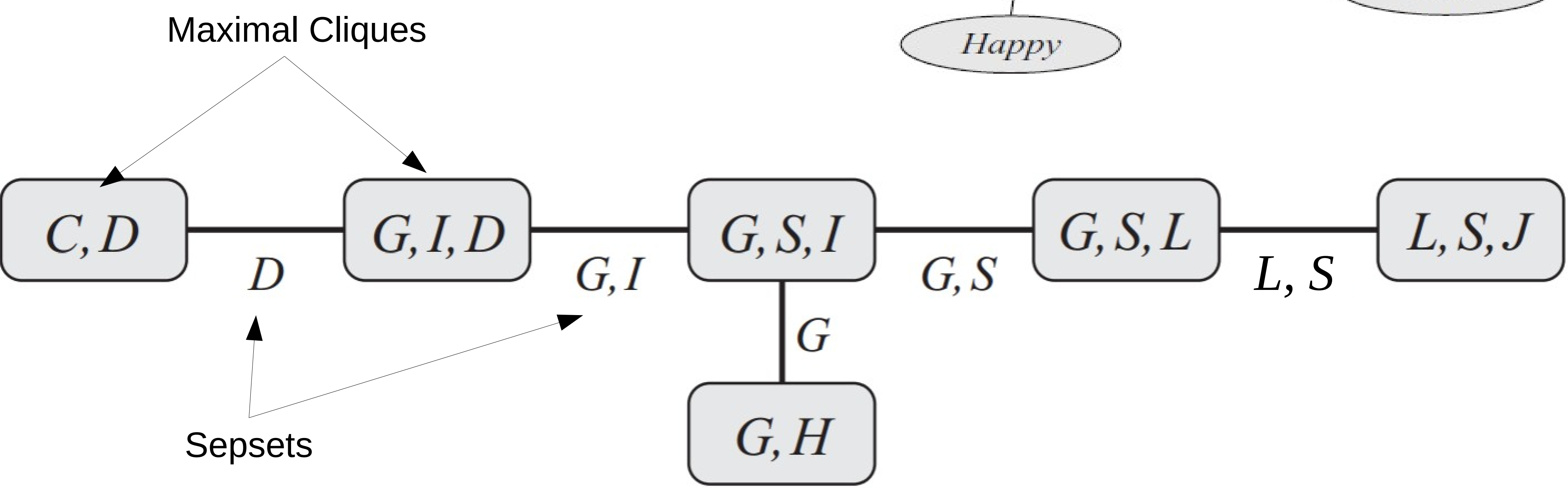
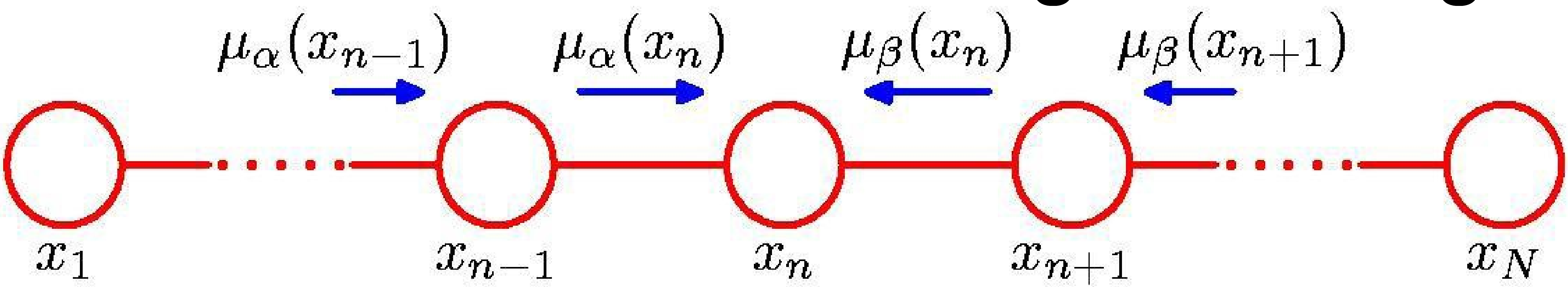


Figure: Koller and Freidman – PGM Principles and Techniques

Sum-Product Algorithm by analogy to Markov chain Message Passing



Generalize this to Clique trees

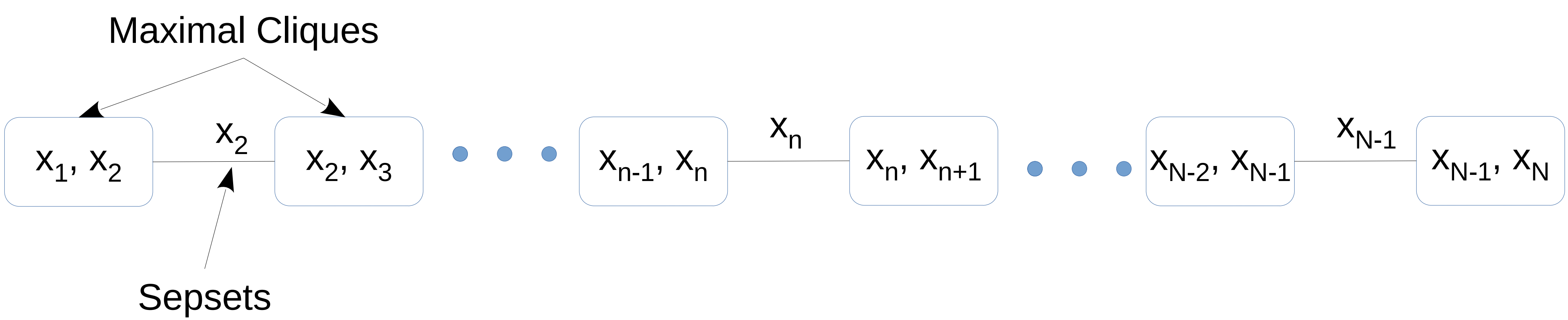
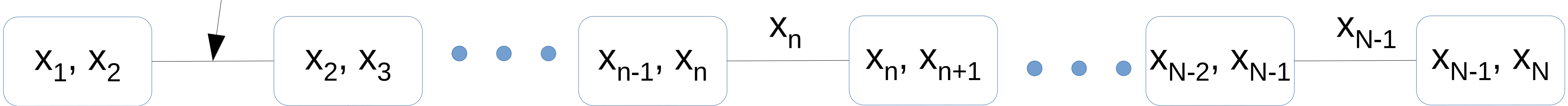


Figure from Bishop- Pattern Recognition and ML

Sum-Product Algorithm

Sum over Cluster 1 minus the sepset

$\delta_{1\rightarrow 2}(x_2) = \Sigma_{x1}\ \psi_1(x_1,x_2)$



$\psi_1 = \phi_1(x_1,x_2)$

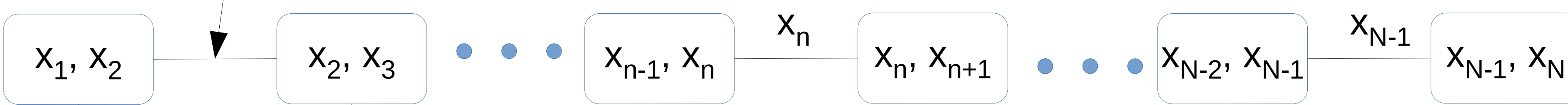
$\psi_2 = \phi_2(x_2,x_3)\ \Pi\ \delta_{1\rightarrow 2}(x_2)$

Product over all incoming messages

Sum-Product Algorithm

$\delta_{1\to 2}(x_2) = \Sigma_{x1}\, \psi_1(x_1,x_2)$

In a more general cluster graph the sum can be over several sepset variables.



$\psi_1 = \phi_1(x_1,x_2)$

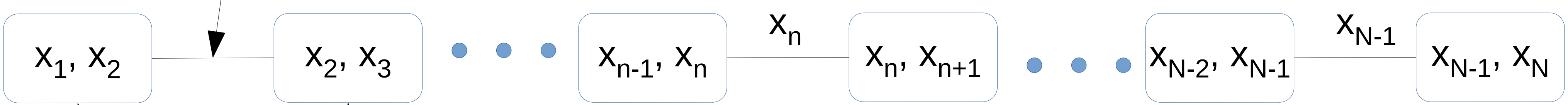
$\psi_2 = \phi_2(x_2,x_3)\ \Pi\ \delta_{1\to 2}(x_2)$

Compute messages at any cluster that is ready, ie has received all incoming messages from edges other than the one being computed.
In a tree that might mean more than one other edge.

Repeat until all messages in both directions have been computed.
In a tree one must go up and down all branches to leaves.

Sum-Product Algorithm

$\delta_{1\to 2}(x_2) = \Sigma_{x1}\, \psi_1(x_1,x_2)$



$\psi_1 = \phi_1(x_1,x_2)$

$\psi_2 = \phi_2(x_2,x_3)\ \Pi\ \delta_{1\to 2}(x_2)$

Notice Product is over incoming messages (outgoing one is left out) and uses the original factor, ϕ_2 , even when the return message is computed. (Different when we do the belief update version)

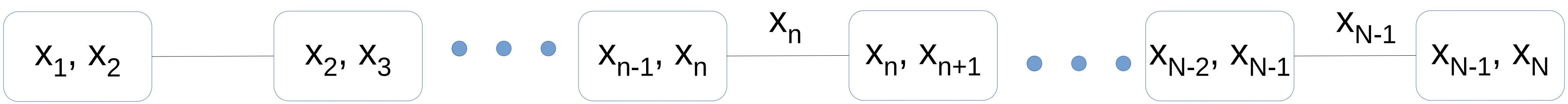
$\beta_i(C_i) = \phi_i(C_i)\ \Pi\ \delta_{j\to i}(S_{ij}) \quad = \Sigma_{\chi\text{-}Ci}\ \Pi_j\ \phi_j(C_j) \quad = Z\, P(C_i)$ = 'Cluster belief'

Over all Neighbors j

Product over all Clusters j

Sum over all variables not in C_i

Calibrated Clique Tree



$$\beta_i(C_i) = \phi_i(C_i) \prod \delta_{j \rightarrow i}(S_{ij})$$

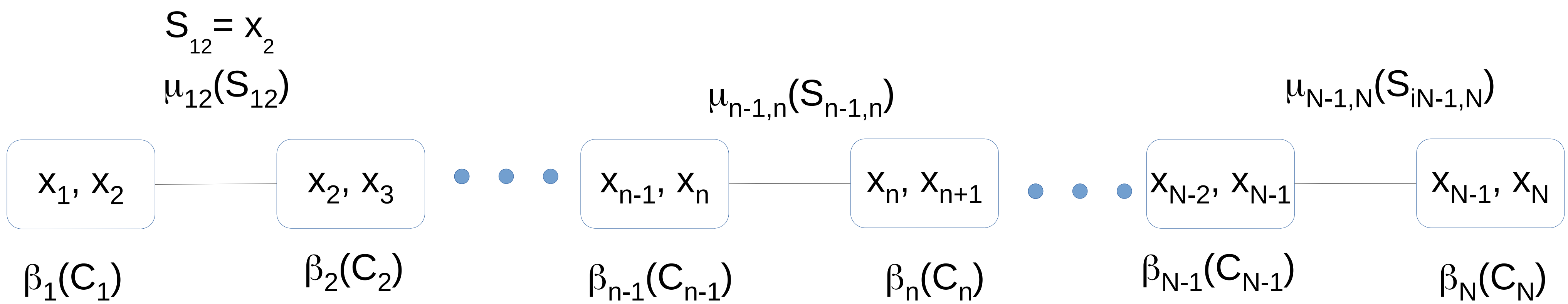
Adjacent Cliques are calibrated if:

$$\Sigma_{C_i - S_{ij}} \; \beta_i(C_i) \; = \; \Sigma_{C_j - S_{ij}} \; \beta_j(C_j)$$

This is called the 'sepset belief', $\mu_{ij}(S_{ij})$
In belief update this will become the ‘message’.

Running the Sum-product Algorithm leads to calibrated trees.

Calibrated Clique Trees



$$\beta_i(C_i) = \phi_i(C_i) \prod_j \delta_{j \to i}(S_{ij})$$

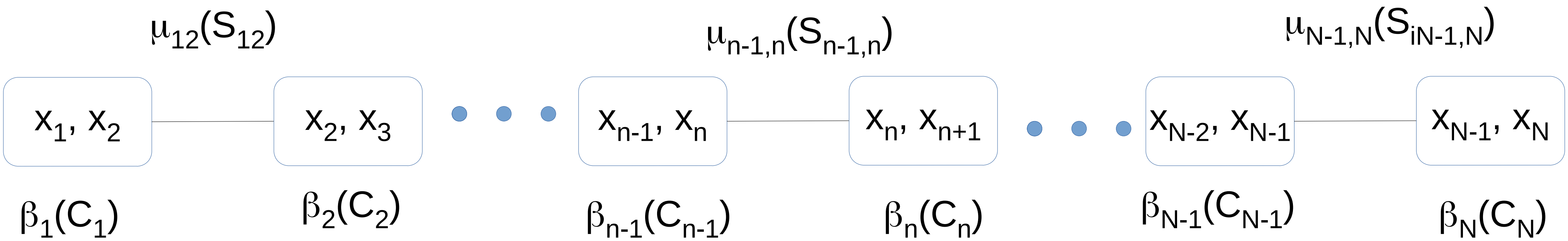
$$\mu_{ij}(S_{ij}) = \sum_{C_i - S_{ij}} \beta_i(C_i) = \sum_{C_j - S_{ij}} \beta_j(C_j)$$

$$\tilde{P}(\chi) = \frac{\prod_i \beta_i(C_i)}{\prod_{i-j} \mu_{ij}(S_{ij})}$$

We see the denominator removes the double count for the sepsets

Using this one can compute the belief of combinations of cliques

Belief Update Message Passing

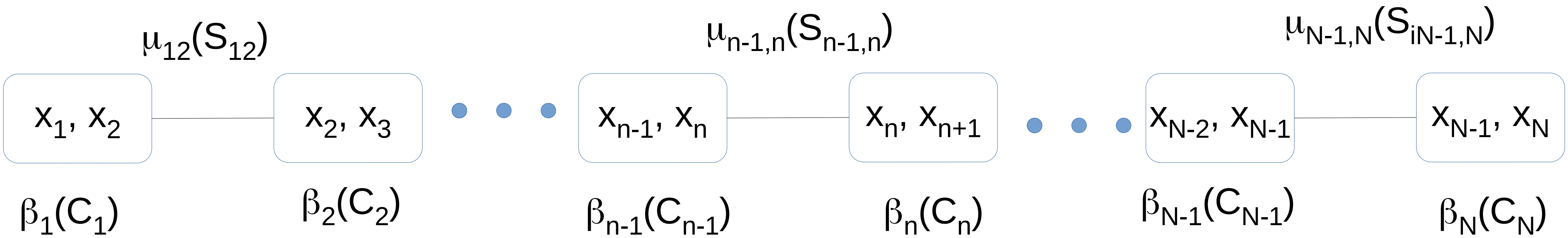


Initialize:

$\beta_i(C_i) = \phi_i(C_i)$	Notice this is not yet the marginal since it does not reflect the fact that some of the variables are in the other potentials. (is not calibrated)
$\mu_{ij}(S_{ij}) = 1$	This is a weird table with all 1's in every element. (also not normalized)

We will just reformulate the previous sum product to allow us to generalize it to loopy graphs

Belief Update Message Passing



Initialize:

$\beta_i(C_i) = \phi_i(C_i)$

$\mu_{ij}(S_{ij}) = 1$

While $\exists$ an uninformed clique:

Select an Edge $i \rightarrow j$:

$\sigma_{ij}(S_{ij}) = \Sigma_{C_i - S_{ij}} \; \beta_i(C_i)$ Sum out of cluster i's potential all but the sepset variables for this edge.

$\beta_j(C_j) = \beta_j(C_j)\, \sigma_{ij}(S_{ij}) \,/\, \mu_{ij}(S_{ij})$. Multiply cluster j potential by the ratio of new to old message.

$\mu_{ij}(S_{ij}) = \sigma_{ij}(S_{ij})$ Replace the old message with the new.

You can check that if you choose to do the message passing in the same order as in the sum product, then the operations are all the same.

Inference and Queries

$\mu_{12}(S_{12})$ $\quad\cdots\quad$ $\mu_{n-1,n}(S_{n-1,n})$ $\quad\cdots\quad$ $\mu_{N-1,N}(S_{iN-1,N})$

x_1, x_2 — x_2, x_3 • • • x_{n-1}, x_n — x_n, x_{n+1} • • • x_{N-2}, x_{N-1} — x_{N-1}, x_N

$\beta_1(C_1)$ $\quad$ $\beta_2(C_2)$ $\quad$ $\beta_{n-1}(C_{n-1})$ $\quad$ $\beta_n(C_n)$ $\quad$ $\beta_{N-1}(C_{N-1})$ $\quad$ $\beta_N(C_N)$

$$\tilde{P}(\chi) = \frac{\prod_i \beta_i(C_i)}{\prod_{i-j} \mu_{ij}(S_{ij})}$$

Incremental Updates:

$$\tilde{P}(\chi\text{-}Y, Y=y) = \mathbf{1}(Y=y) \frac{\prod_i \beta_i(C_i)}{\prod_{i-j} \mu_{ij}(S_{ij})}$$

Notice that these are not normalized.
We would have to compute Z to get probabilities.

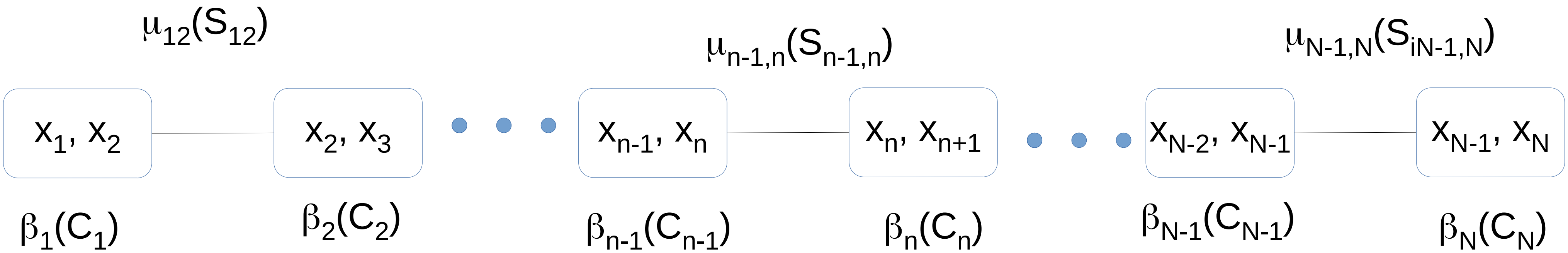
For i and j adjacent:

$$\tilde{P}(C_i, C_j) = \frac{\beta_i(C_i)\, \beta_j(C_j)}{\mu_{ij}(S_{ij})}$$

Notice this is magic for computing marginals!

For further separated clusters see Alg. 10.4

Magic Marginals



$$\tilde{P}(\chi) = \frac{\prod_i \beta_i(C_i)}{\prod_{i-j} \mu_{ij}(S_{ij})}$$

For i and j adjacent:

$$\tilde{P}(C_i, C_j) = \frac{\beta_i(C_i)\ \beta_j(C_j)}{\mu_{ij}(S_{ij})}$$

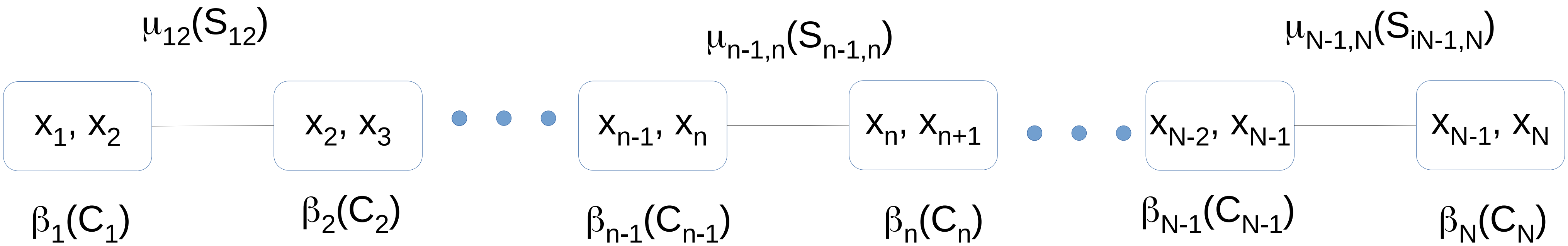
For any i and j

$$\tilde{P}(C_i, C_j) = \Sigma_{\text{path -Ci -Cj}} \frac{\prod_k \beta_k(C_k)}{\prod_{k-l} \mu_{kl}(S_{kl})}$$

Products along path from i to j

We only need to do inference on a sub-tree

Evidence



Incremental Updates:

$$\tilde{P}(\chi - Y, Y=y) = \frac{\mathbf{1}(Y=y)\ \prod_i \beta_i(C_i)}{\prod_{i-j} \mu_{ij}(S_{ij})}$$

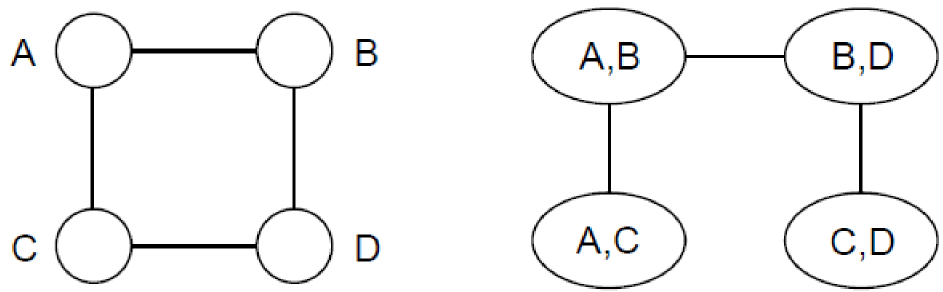
Set to 0 all $\beta_i(C_i)$ elements that are not consistent with evidence.
Then recompute the normalization.

Message Passing in Clique Trees

- Many interesting algorithms are special cases:
- calculation of posterior probabilities in mixture models
- Baum-Welch algorithm for hidden Markov models
- posterior propagation for probabilistic decision trees
- Kalman Filter updates

From Jordan, PGM Intro

Message Passing in Clique Trees



- Note that C appears in two non-neighboring cliques.
- Question: What guarantee do we have that the probability associated with C in these two cliques will be the same?
- Answer: Nothing. In fact this is a problem with the algorithm as described so far. It is not true that in general local consistency implies global consistency.

From Jordan, PGM Intro

Clique Trees

A triangulated graph is one in which no cycles with four or more nodes exist in which there is no chord
We triangulate a graph by adding chords:

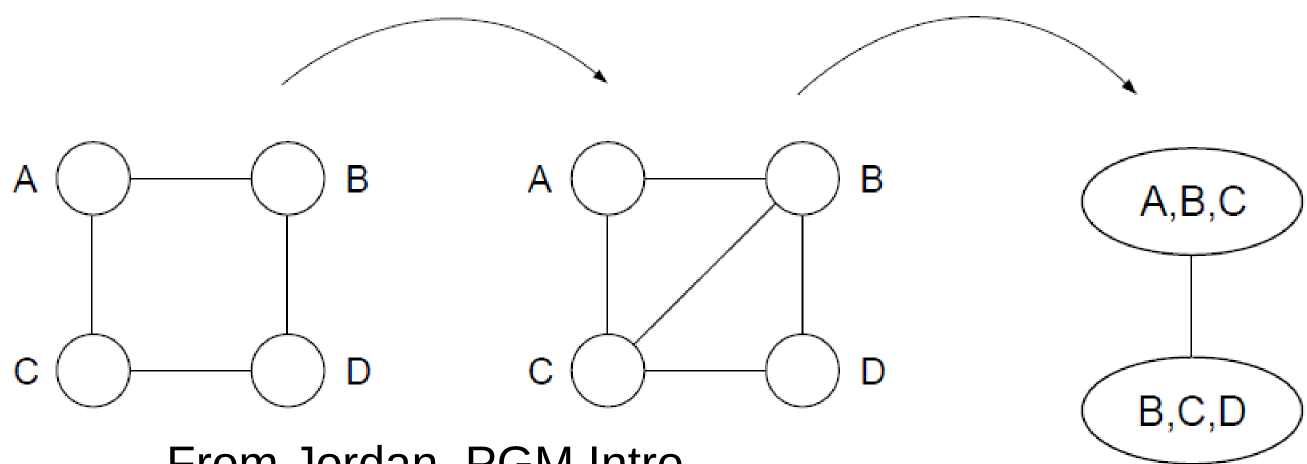
- A ‘cluster graph’ has a potential of a product over every factor with a scope contained in the cluster

A triangulated graph is one in which no cycles with four or more nodes exist in which there is no chord
We triangulate a graph by adding chords:

- A ‘Cluster Tree’ is a cluster graph that has no cycles
- A ‘Clique Tree’ is a triangulated cluster tree that has the running intersection property.

Triangulate a graph by adding chords

- A triangulated graph is one in which no cycles with four or more nodes exist in which there is no chord
- A clique tree for a triangulated graph has the running intersection property: if a node appears in two cliques, it appears everywhere on the path between the cliques
- Thus local consistency implies global consistency for such clique trees

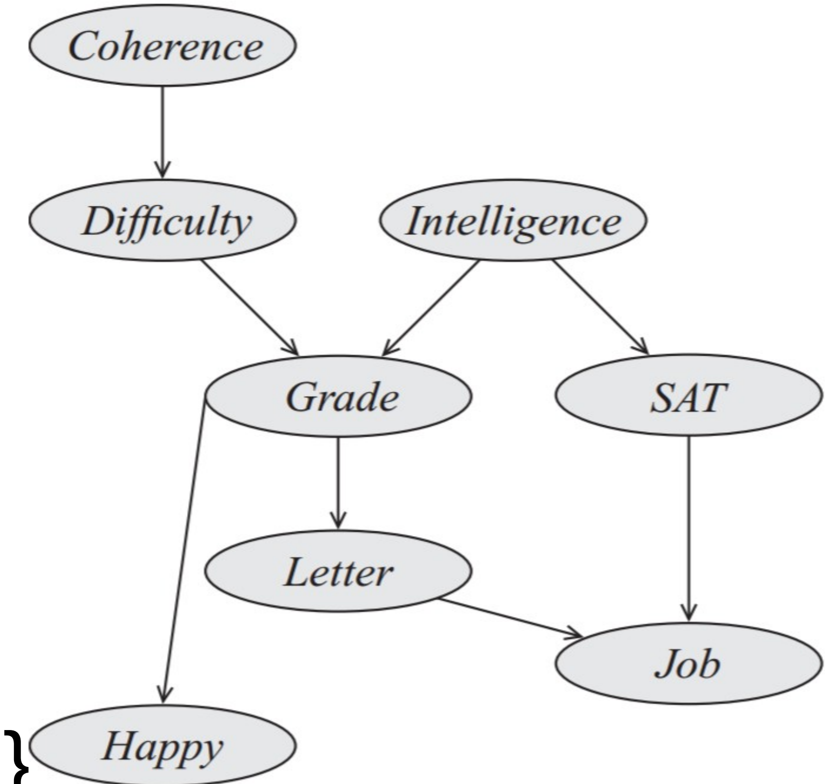


From Jordan, PGM Intro

The Junction Tree Algorithm

- Exact inference on general graphs.
- Works by turning the initial graph into a junction tree and then running the sum-product algorithm.
- Intractable on graphs with large cliques.

Clique Graphs



- Imagine marginalizing in order: (C)(D)(H)(I)(G)(SLJ)
- $\phi(D) = \Sigma_C P(D|C)P(C)$: {C,D}
- $\phi(IG) = \Sigma_D P(G|DI)\phi(D)$: {D,I,G}
- $\phi(G) = \Sigma_H P(H|G)$: {G,H}
- $\phi(GS) = \Sigma_I \phi(IG)P(S|I)P(I)$: {G,I,S} and so on ->

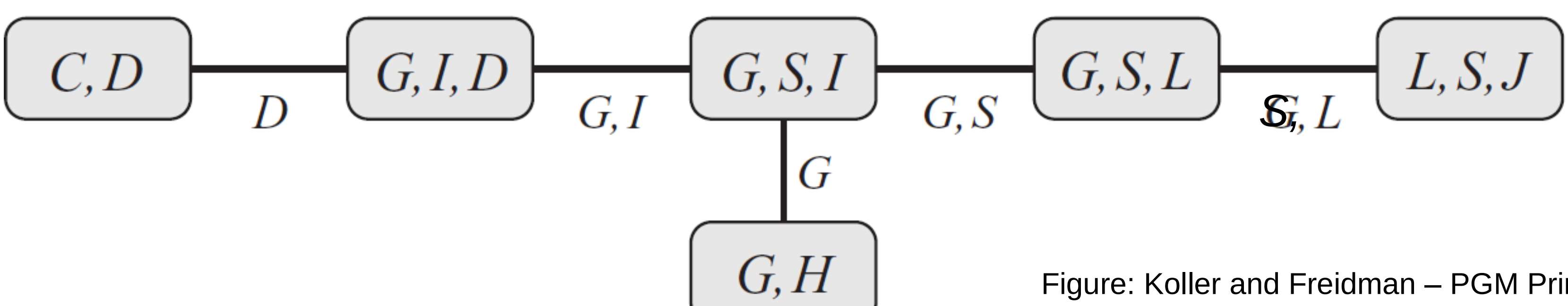


Figure: Koller and Freidman – PGM Principles and Techniques

Building A Clique tree
AKA Junction Tree

Clique Tree from Chordal Graphs:

First Moralize the Graph

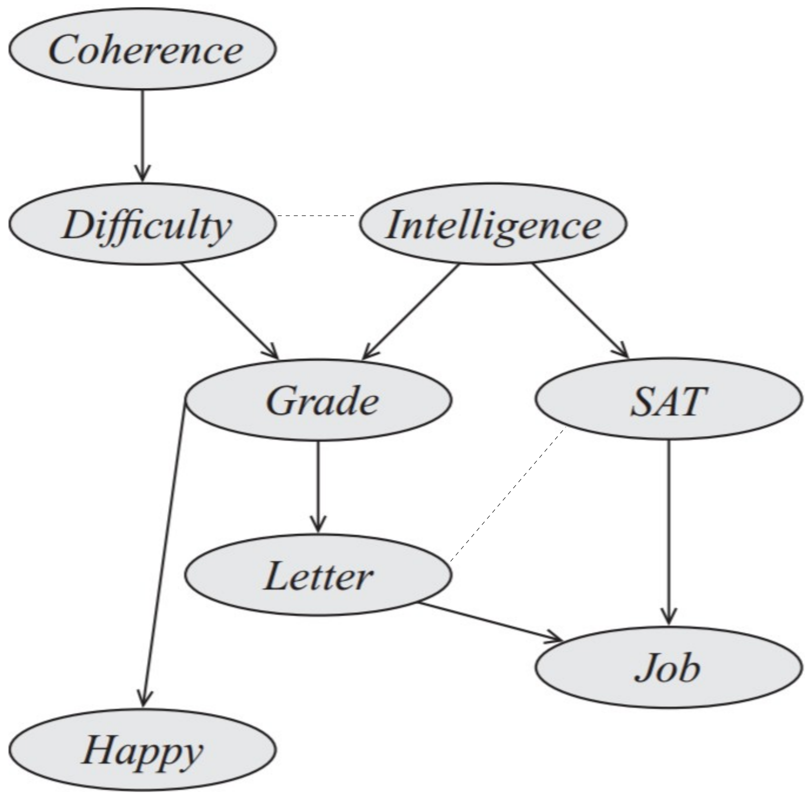


Figure: Koller and Freidman – PGM Principles and Techniques

Building A Clique tree
AKA Junction Tree

Clique Tree from Chordal Graphs:

First Moralize the Graph.

Make it undirected.

Then do triangulation on all cycles.

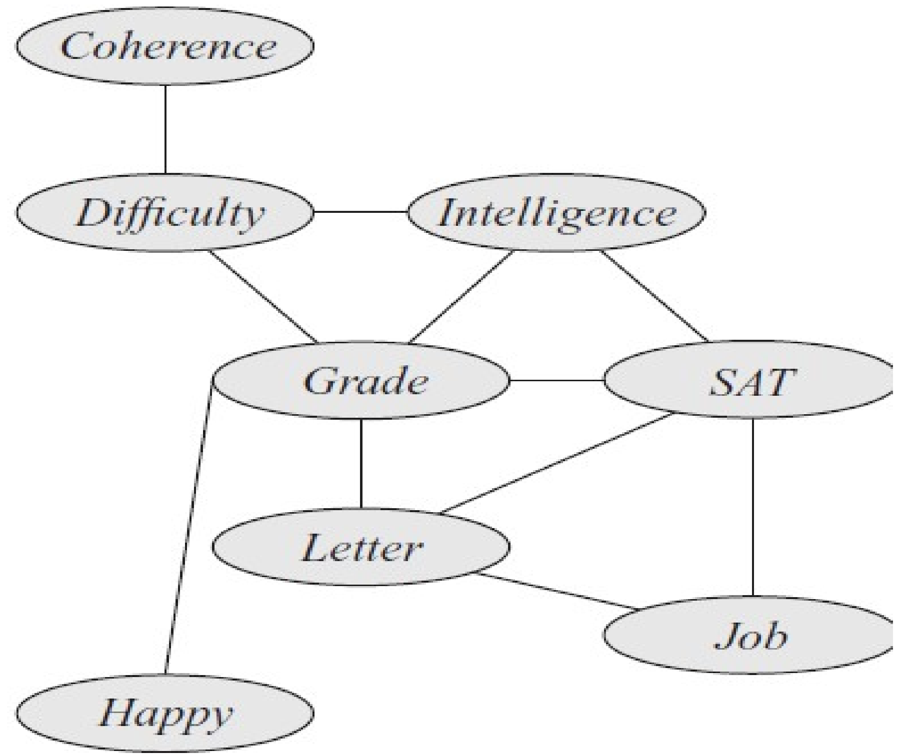
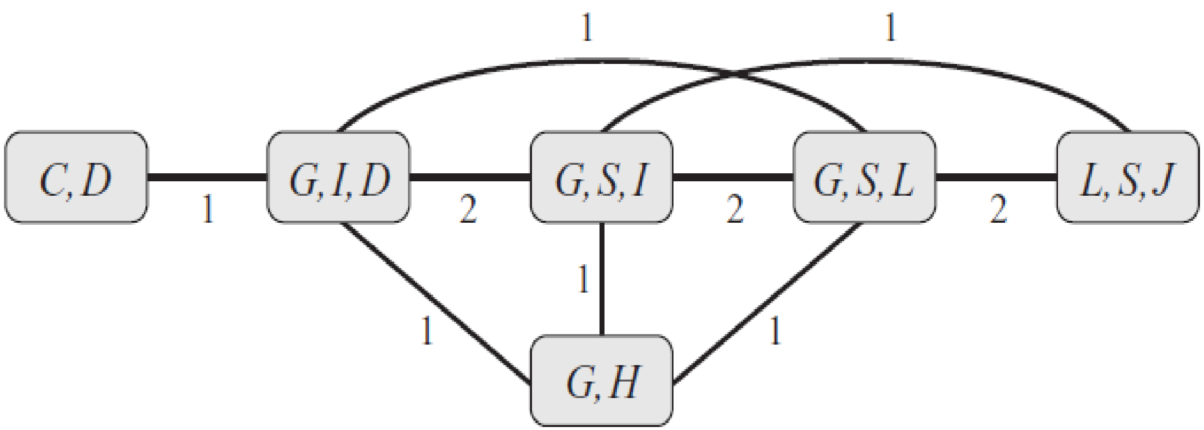
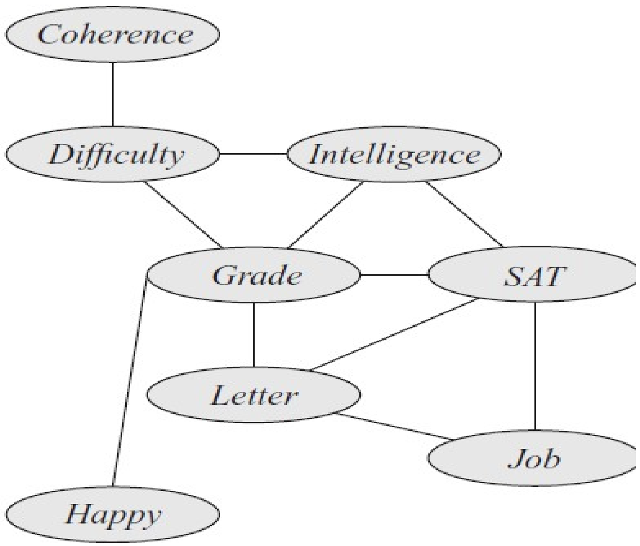


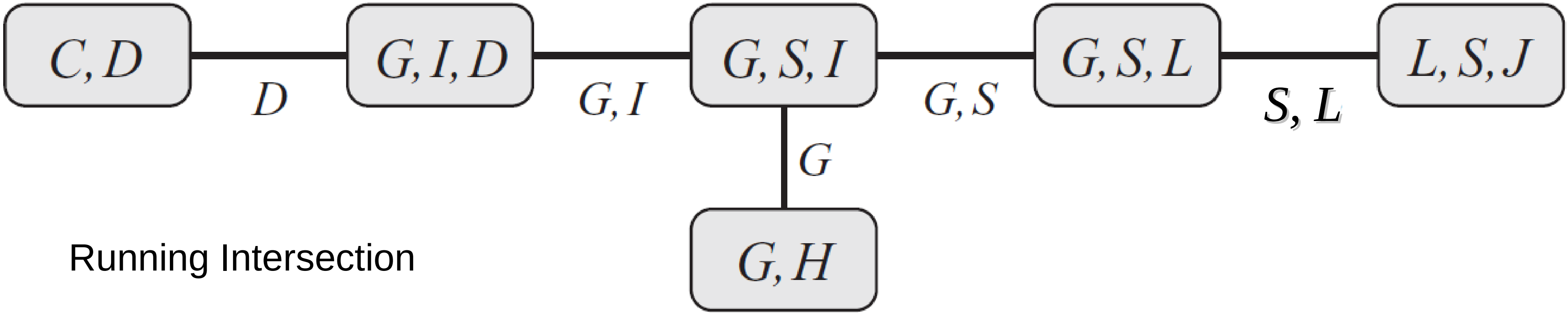
Figure: Koller and Freidman – PGM Principles and Techniques

Building A Clique tree
AKA Junction Tree



The Cluster Graph



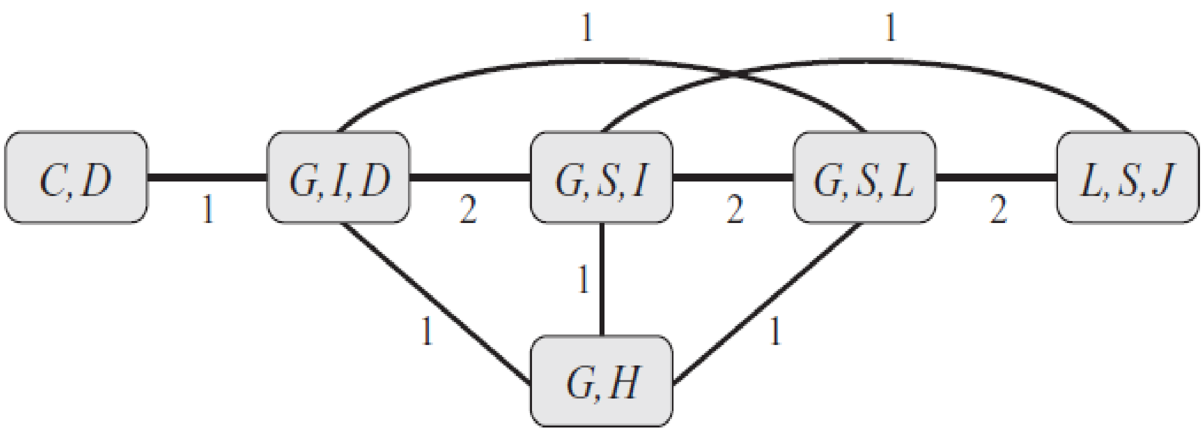


Running Intersection

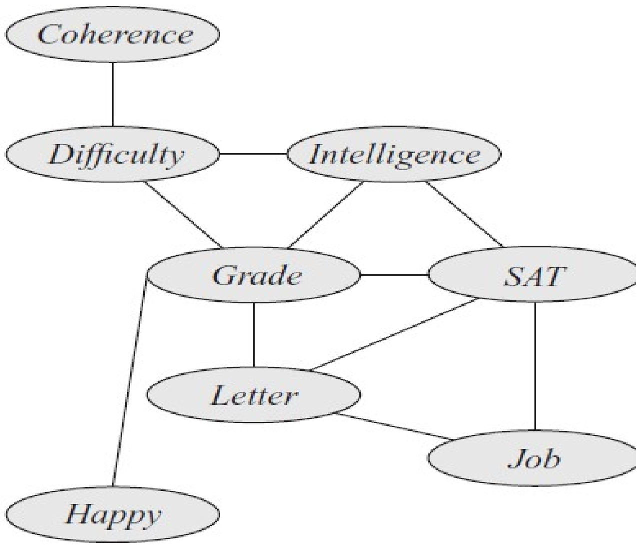
The Clique Tree (Maximal spanning tree)

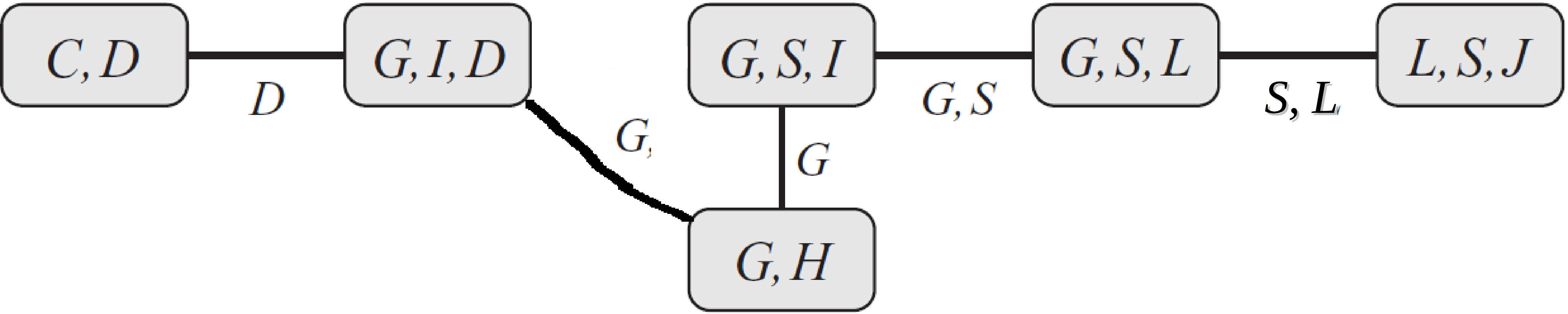
Figure: Koller and Freidman – PGM Principles and Techniques

Building A Clique tree AKA Junction Tree



The Cluster Graph





Not Maximal → NOT Running Intersection

Figure: Koller and Freidman – PGM Principles and Techniques

Variable Elimination vs Sum-Product, vs. Message Passing vs Belief Propagation(update) vs Junction Tree

- All are at the core, the same algorithm! (marginalization)
- Perhaps one says VE (=sum-product) when one does not save the calculations for reuse.
- Junction Tree refers to a pre-processing step of triangulating into chordal graphs.
- BP one formulates as ‘updating’ the sepset beliefs.
- Loopy BP can be done for approx inference.
- Message passing is generic and can be anything that organizes the calculation.
- $\mu_{ij}(S_{ij}) = \delta_{j \to i}(S_{ij})\ \delta_{i \to j}(S_{ij})$

Names of Algorithms for Computing Z

- Variable Elimination, VE: Take one variable at a time and eliminate it.
- Sum-Product for VE, S-PVE: Do the operations of VE by the more systematic way with message passing.

Names of Algorithms for Computing Z

- Wikipedia: 'Belief propagation, also known as sum-product message passing, is a message passing algorithm for performing inference on graphical models...'
- Book is also vague to draw boundaries between some of these.
- VE seems to be one variable at a time with an ordering (but of course that ordering might allow you to do cliques marginalizing more than one).
- Then there are two flavors of message passing algorithms that work on any tree:
  - Sum-Product Message Passing: The one were you just compute and save all messages using the original factors
  - Belief Update Message Passing: Where you have both edge (Sepset) beliefs and node (clique) beliefs updating as you go starting from an inititalization.