

# CSCI E-82a

## Probabilistic Programming and AI

### Lecture 3

# Exact Inference for Graphical Models

Steve Elston



HARVARD  
Extension School

Copyright 2019, Stephen F Elston. All rights reserved.

# Agenda

- Exact Inference for Graphical Models
- The Variable Elimination Algorithm
- Factors in Graphical Models
- Factor Example
- Variable Elimination in Undirected Graphs
- Queries with Evidence
- Problems with Variable Elimination
- The Belief Propagation Algorithm
- Background for Junction Tree Algorithm
- Junction Tree Algorithm
- Relationship between Junction Tree and Variable Elimination

# Inference for Graphical Models

**Inference** is the process of making **queries** on models to receive posterior distributions or marginal distributions of selected variables

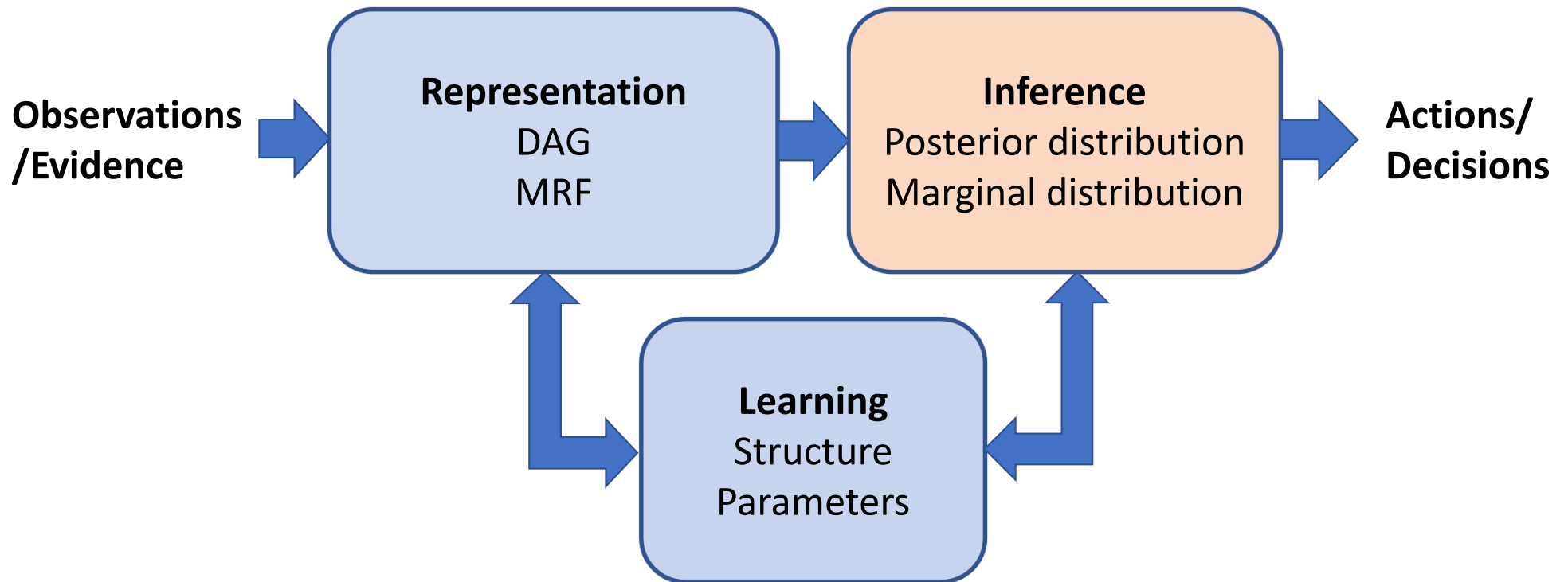
- In other words, inference is the process of getting useful answers from a model!
- **Evidence** (observations) is typically a component of a query
- Query can be on one or more variables

# Inference for Graphical Models

In the past 2 lessons our focus was on **representation**

- Representation allows us to define or represent a model
- We have investigated two representations:
  - Bayes networks or directed acyclic graphs (DAGs)
  - Undirected graphical networks, Markov random fields (MRF)
- **Efficient representation**
  - Represent independences
  - Potentials for cliques
- A good representation is required to perform inference

# Focus on Inference for Graphical Models



Schematic of intelligent agent using directed graphical model

# Inference for Graphical Models

Inference is a **computationally difficult** problem

- Direct tabular solutions can be **computationally infeasible**
- Computational complexity is NP, or  $O(n^k)$
- But, we can work with **distributions factored on a graph by independencies**

# Inference for Graphical Models

Two major classes of inference algorithms

- In this lecture we focus on **exact inference algorithms**
  - Generally computationally efficient for problems with limited numbers of variables and states
  - Algorithms provide a basis for understanding other methods
- Take up **approximate methods** in later lesson
  - Highly scalable to high dimensional problems
  - Accommodate continuous variables

# Inference for Graphical Models

There are no exact algorithms which have guaranteed low computational complexity

- In practice, a number of algorithms work well most of the time
- But, computational complexity can change radically with ordering of variables



# Inference for Graphical Models

We focus on the 3 most widely used algorithms

- Variable elimination
- Message passing or belief propagation
  - Generalizes variable elimination on trees
- Junction tree
  - Generalizes belief propagation to complex graphs
- There are many other variations and research continues

# The Variable Elimination Algorithm

Variable elimination works by marginalizing out variables one by one

- Variables are ordered
- Working in order, the variables are marginalized out, **eliminated**
- The result is the marginal distribution of the variable queried
- Finding the most efficient elimination order is an NP hard problem
- Different ordering of the variables changes computational complexity a lot!

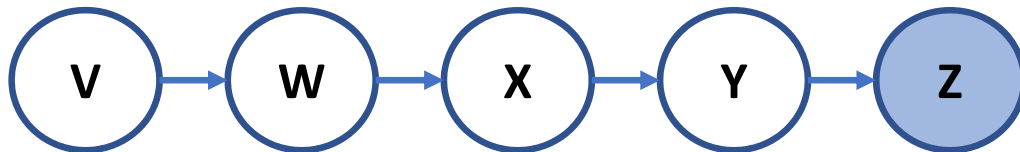
# The Variable Elimination Algorithm

Example of variable elimination on a **chain graph**

- The joint distribution factors as follows:

$$P(V, W, X, Y, Z) = P(V) P(W | V) P(X | W) P(Y | X) P(Z | Y)$$

- The DAG is a chain:

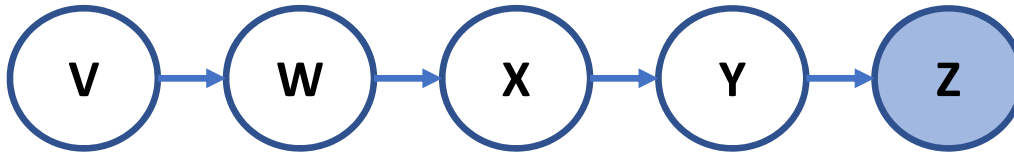


- The query is to find the marginal distribution of Z

# The Variable Elimination Algorithm

Example of variable elimination on a chain graph

- Starting with the DAG chain:



- The query is to find the marginal distribution of Z:

$$P(Z) = \sum_V \sum_W \sum_X \sum_Y P(V, W, X, Y, Z)$$

- Or for the factorized distribution:

$$P(Z) = \sum_V \sum_W \sum_X \sum_Y P(V) P(W | V) P(X | W) P(Y | X) P(Z | Y)$$

# The Variable Elimination Algorithm

Example of variable elimination on a chain graph

- First, eliminate the variable  $V$
- Start with the factorized distribution:

$$P(Z) = \sum_V \sum_W \sum_X \sum_Y P(V) P(W | V) P(X | W) P(Y | X) P(Z | Y)$$

- Rearrange terms and the summation:

$$P(Z) = \sum_W \sum_X \sum_Y P(X | W) P(Y | X) P(Z | Y) \sum_V P(V) P(W | V)$$

# The Variable Elimination Algorithm

Example of variable elimination on a chain graph

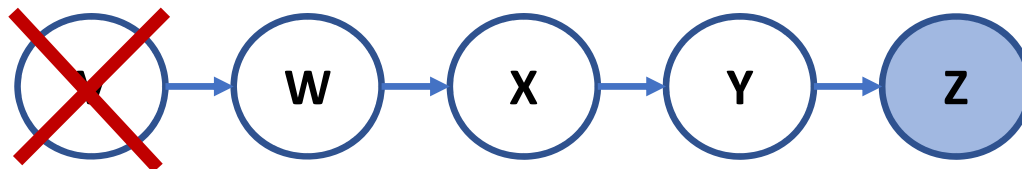
- The marginal of the variable  $W$  is:

$$p(W) = \sum_V P(V) P(W | V)$$

- The expression of the marginal distribution of  $Z$  is:

$$P(Z) = \sum_W \sum_X \sum_Y P(X | W) P(Y | X) P(Z | Y) p(W)$$

- The variable  $V$  is eliminated and the DAG is now:



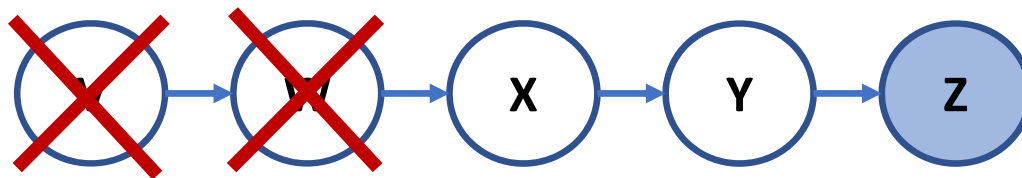
# The Variable Elimination Algorithm

Example of variable elimination on a chain graph

- Now eliminate W, making the marginal of Z:

$$\begin{aligned} P(Z) &= \sum_X \sum_Y P(Y | X) P(Z | Y) \sum_W p(W) P(X | W) \\ &= \sum_X \sum_Y P(Y | X) P(Z | Y) p(X) \end{aligned}$$

- The variable W is eliminated and the DAG is now:



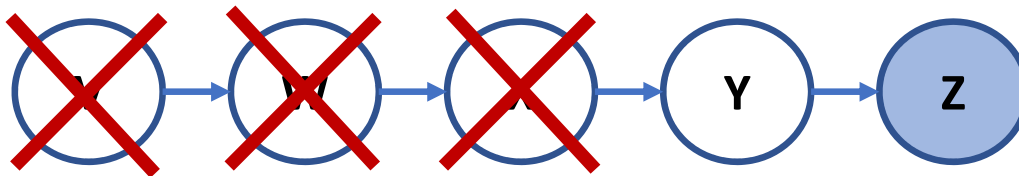
# The Variable Elimination Algorithm

Example of variable elimination on a chain graph

- Now eliminate  $X$ , making the marginal of  $Z$ :

$$\begin{aligned} P(Z) &= \sum_Y P(Z | Y) \sum_X p(X) P(Y | X) \\ &= \sum_Y P(Z | Y) p(Y) \end{aligned}$$

- The variable  $X$  is eliminated and the DAG is now:





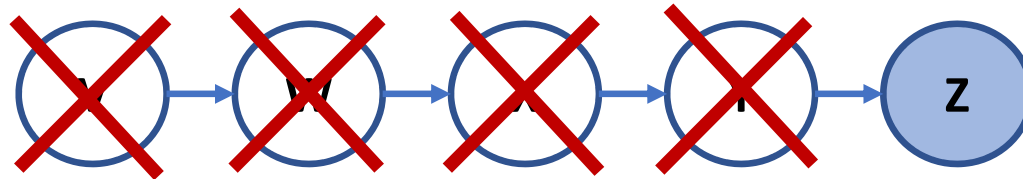
# The Variable Elimination Algorithm

Example of variable elimination on a chain graph

- Finally, eliminate Y, and the marginal of Z is:

$$P(Z) = \sum_Y p(Z) P(Z | Y)$$

- The DAG now just has one variable Z:

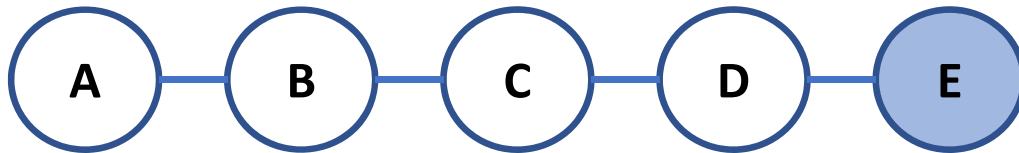


- The variable elimination process is now complete

# Variable Elimination for Undirected Graphs

The same principle of variable elimination can be applied to the potentials of an undirected graph

- Start with the undirected chain graph:



- The marginal distribution is represented by the sum over the variables of the product of the 4 potentials:

$$P(E) = \sum_A \sum_B \sum_C \sum_D \frac{1}{Z} \phi(A, B) \phi(B, C) \phi(C, D) \phi(D, E)$$

# Factors in Graphical Models

**Factors** are a generalized formulation

- A factor is a multidimensional table which assigns a value to a set of variables
- CPTs are factors for DAGs
- Potentials are factors for MRFs
- A joint distribution can be computed using general factors:

$$p(x_1, \dots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(x_c)$$

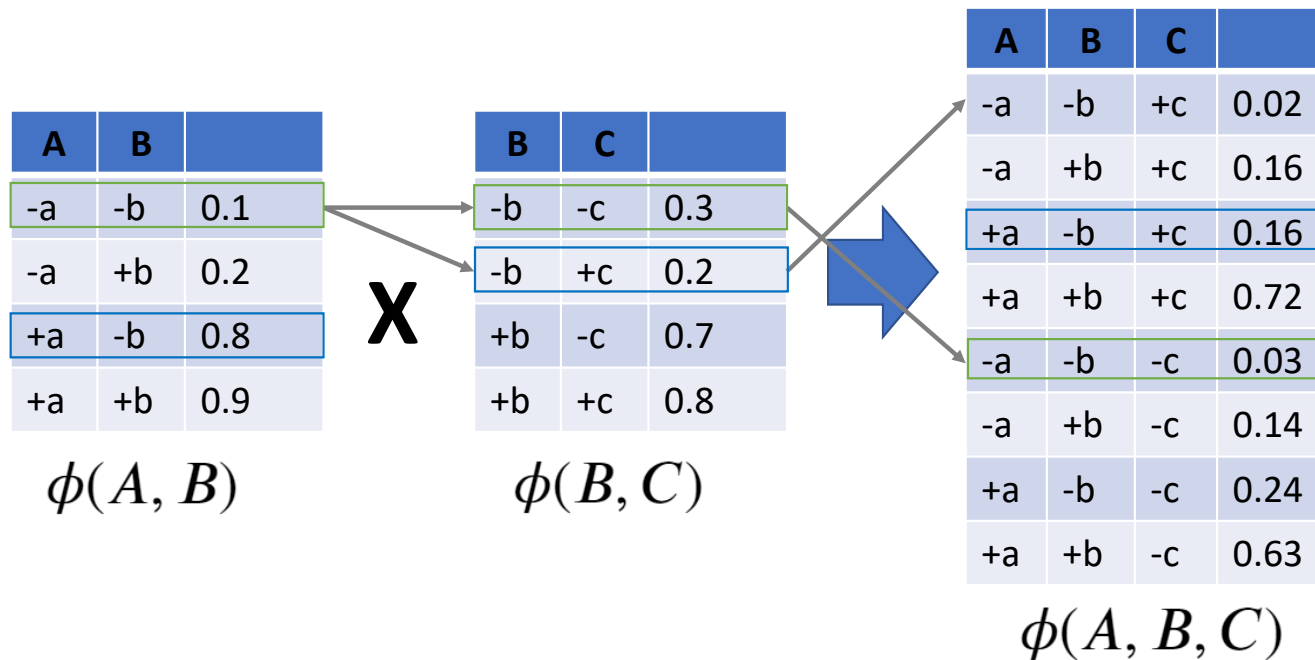
- There are two essential operations on factors
  - Computing a product
  - Marginalization

# Factor example

Taking the product of two factors with a common variable

$$\phi(A, B, C) := \phi(A, B) \times \phi(B, C)$$

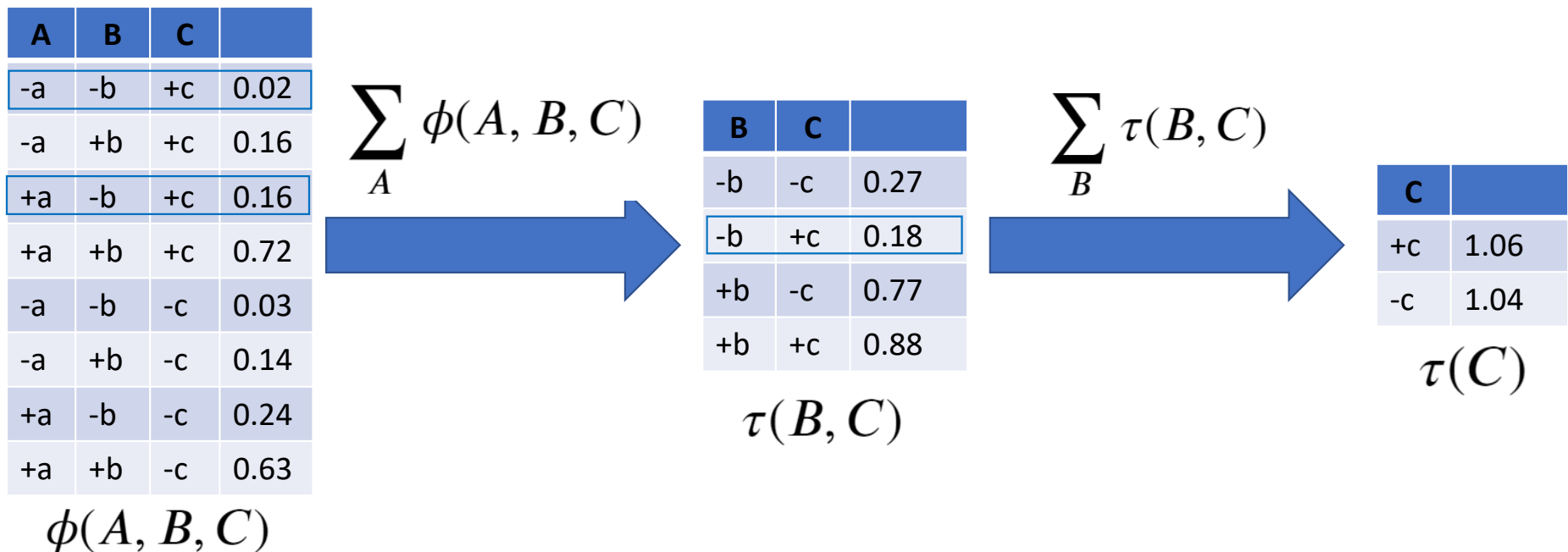
Computationally is an outer product



# Factor example

Computing the (unnormalized) marginal factor

Compute factor  $\tau(C)$  by marginalizing  $\phi(A, B, C)$



# Variable Elimination for Undirected Graphs

The same principle of variable elimination can be applied to the factors of an undirected graph

- Distribution is factorized by the potentials:

$$P(E) = \sum_A \sum_B \sum_C \sum_D \frac{1}{Z} \phi(A, B) \phi(B, C) \phi(C, D) \phi(D, E)$$

- As with the DAG example, start by eliminating the first variable in the chain, A:

$$\begin{aligned} P(E) &= \frac{1}{Z} \sum_B \sum_C \sum_D \phi(B, C) \phi(C, D) \phi(D, E) \underbrace{\sum_A \phi(A, B)}_{\tau(B)} \\ &= \frac{1}{Z} \sum_B \sum_C \sum_D \phi(B, C) \phi(C, D) \phi(D, E) \tau(B) \end{aligned}$$

# Variable Elimination for Undirected Graphs

The same principle of variable elimination can be applied to the factors of an undirected graph

- Continuing with the variable elimination:

$$\begin{aligned} P(E) &= \frac{1}{Z} \sum_C \sum_D \phi(C, D) \phi(D, E) \underbrace{\sum_B \phi(B, C) \tau(B)}_{\tau(C)} \\ &= \frac{1}{Z} \sum_D \phi(D, E) \underbrace{\sum_C \phi(C, D) \tau(C)}_{\tau(D)} \\ &= \frac{1}{Z} \sum_D \phi(D, E) \tau(D) \\ &= \frac{1}{Z} \tau(E) \end{aligned}$$

# Queries with Evidence

How do we incorporate evidence into a query?

- Need to compute the conditional distribution:

$$p(Y \mid E = e) = \frac{p(Y, E = e)}{p(E = e)}$$

- $P(X, Y, E)$  is the distribution of query variables  $Y$ , evidence variables  $E$  and unobserved variables  $X$
- To compute  $p(Y \mid E = e)$ , sum every factor  $\phi(X, Y, E)$  where  $E$  is in scope of  $Y$  and is set to  $e$
- Variable elimination is used to compute  $p(E = e)$
- In other words, adding evidence reduces the number of states of the evidence variable



# The Variable Ordering Problem

Need to find the most efficient ordering of variables

- Finding **optimal elimination order** is **NP hard!**
- In practice, use one of **many possible heuristics**
- The same problem occurs for belief propagation and the junction tree algorithm

# The Variable Ordering Problem

Need to find the most efficient ordering of variables

- Some of the most commonly used heuristics
- ***Min-neighbors***: Choose a variable with the fewest dependent variables
- ***Min-weight***: Choose variables that minimize the size of product of the cardinalities of its dependent variables
- ***Min-fill***: Choose vertices that minimize the size (dimensions) of the factor that will be added to the graph

# Problems with Variable Elimination

Why is variable elimination not sufficient for all inference problems?

- Computational complexity depends on order of elimination
- Finding optimal elimination order is NP hard!
- Each query requires a new series of elimination
  - Inefficient for multiple queries

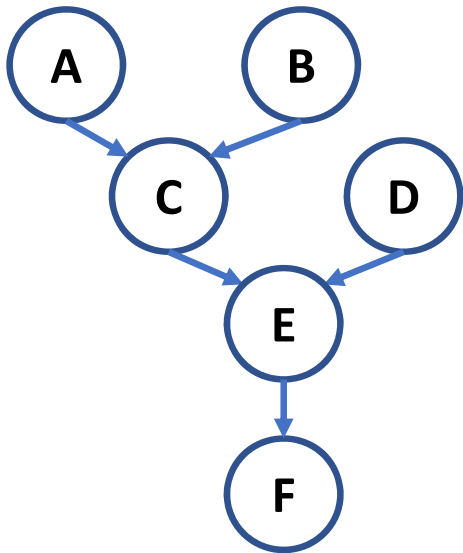
# The Message Passing Algorithm

**Message passing algorithm** enables multiple queries from a single set of messages

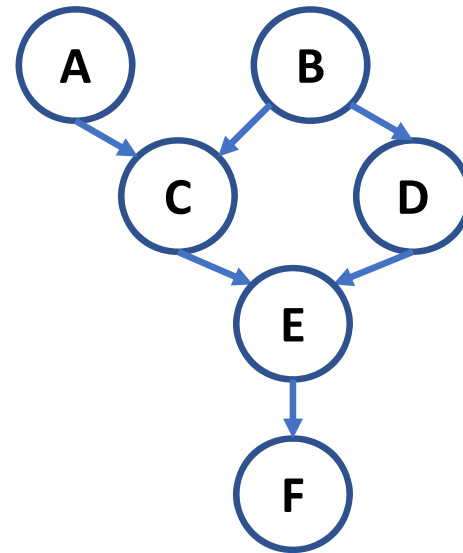
- Also known as the **belief propagation algorithm**
- Message passing algorithm works on **tree graphs**
- The values passed along the tree are **factors**
  - Factors are functions of potentials
  - Factors are not probabilities!
- Intermediate results are saved for future queries in the form of **messages**

# The Message Passing Algorithm

**Definition:** A tree graph is a undirected acyclic graph, in which any pair of nodes are connected by exactly one path.



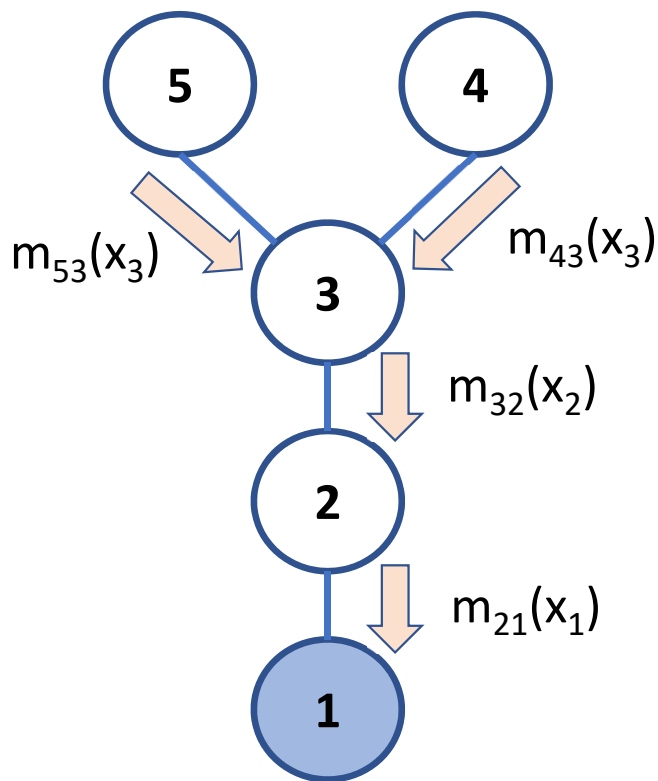
Example of a **tree graph**



This is **not a tree graph!**

# The Message Passing Algorithm

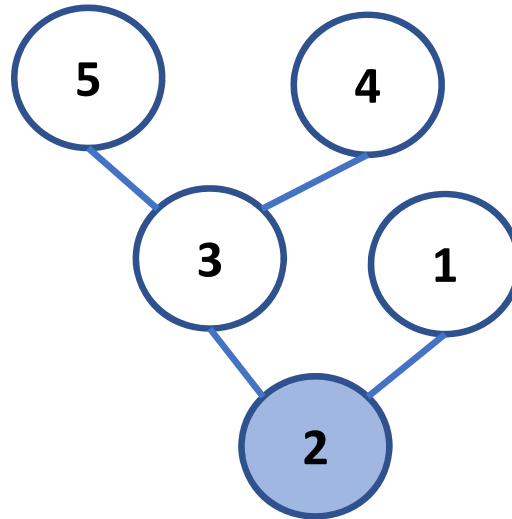
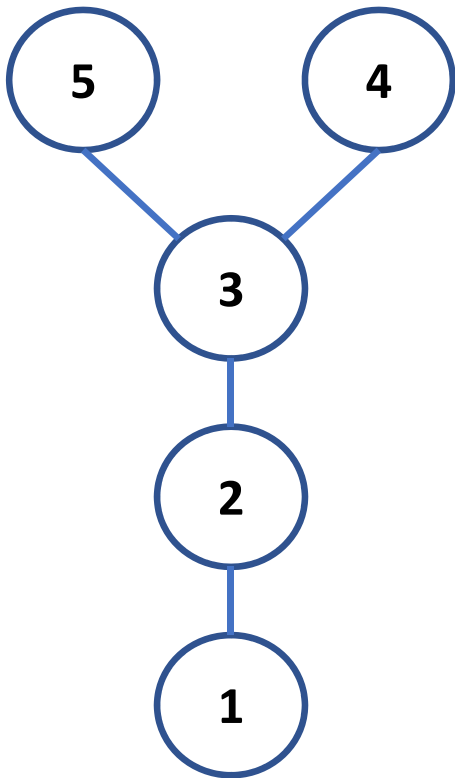
**Example: query variable 1 in tree graph**



- Variable 1 at root of tree
- Start from the leaf nodes passing messages down the tree
- Messages **propagate belief**
- When each node receives messages from all 'upstream' neighbors, then emits a message down
- Final message is to the root or query node
- Executes in **linear time!**

# The Message Passing Algorithm

What happens if we want to query node 2?



- Node 2 becomes the root
- The result is still a tree
- But, we **must pass new messages!**

# The Belief Propagation Algorithm

Is there an approach that only requires **passing messages one time?**

- **Yes!**
- **Pass messages in both directions**
- **Cache** messages
- Add **evidence**
- Then, any **query is computed using save messages**



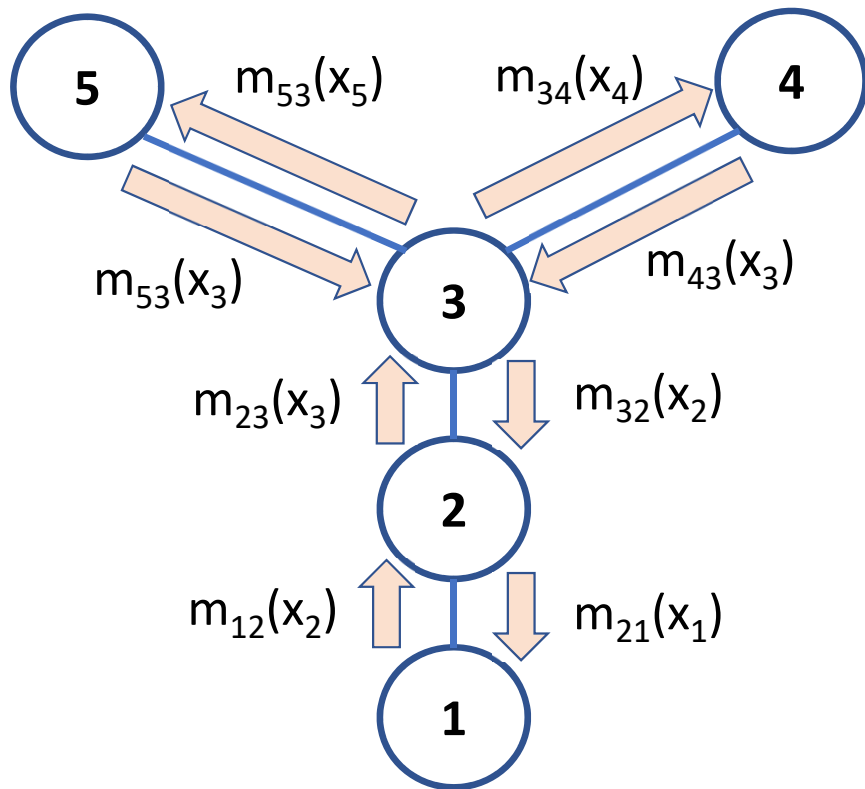
# The Belief Propagation Algorithm

Steps of the two-step message passing algorithm:

1. CPTs are updated with evidence
2. Nodes **collect** messages emitted by their neighbors.
  - **Leaf nodes** emit messages at the start of the collection step.
3. Once a node has collected messages from neighbors it **distributes** or **emits** messages to its neighbors.
  - In the distribution phase, a node may only emit a message once it has collected messages from all other neighbors.

# The Belief Propagation Algorithm

Messages are passed both directions



- 'Up' the tree from the leaves
- 'Down' the tree from the root
- Caching complete set of messages allows query on any node
- **Message  $m_{ij}$  propagates a belief from one node to a neighbor**
- **Pass all messages in  $2 \times \text{node count time!}$**

# The Belief Propagation Algorithm

## **Upward pass:**

- Each leaf in the junction tree sends a message to its parent
  - Message is the marginal of its table; the sum of variables not in the separator.
- When a parent receives a message from a child, it multiplies its table by the message table to update its table
- When a parent receives messages from all its children, it acts as the next leaf
- Process continues until the root receives messages from all children

# The Belief Propagation Algorithm

## **Downward pass:**

- Reverse of upward pass, starting at root
- Root sends a message to each child
- Root divides current table by the message received from the child
  - Marginalize out variables not in the separator
  - Send result to the child
- Each child multiplies its table by its parent's table
- Child then acts as the root until leaves are reached
- Find marginal of variables by summing out variables as needed
- Done!

# The Message Passing Algorithm

## Message passing as **variable elimination**

- We can formulate **variable elimination** in terms of **factors**

$$\tau_{jk}(x_k) = \sum_{x_j} \phi(x_k, x_j) \tau_j(x_j)$$

- Sum-product of messages passes from one variable, i, to the next, j:

$$m_{ji}(x_i) = \sum_{x_j} \phi(x_j) \phi(x_i, x_j) \prod_{f \in Pa(j) \setminus i} m_{fj}(x_j)$$

Where,

$\phi(x_j)$  = *edge factor* ; or junction factor

$\phi(x_i, x_j)$  = *clique factor* ; includes the nodes

And,  $f \in Pa(j) \setminus i$  indicates all the parents of  $x_j$  except  $i$

# The Belief Propagation Algorithm

**Add evidence** to a query:

- Factor is update using evidence with the relationship:

$$\phi^E(x_i) = \phi(x_i) \delta(x_i, \bar{x}_j)$$

Where

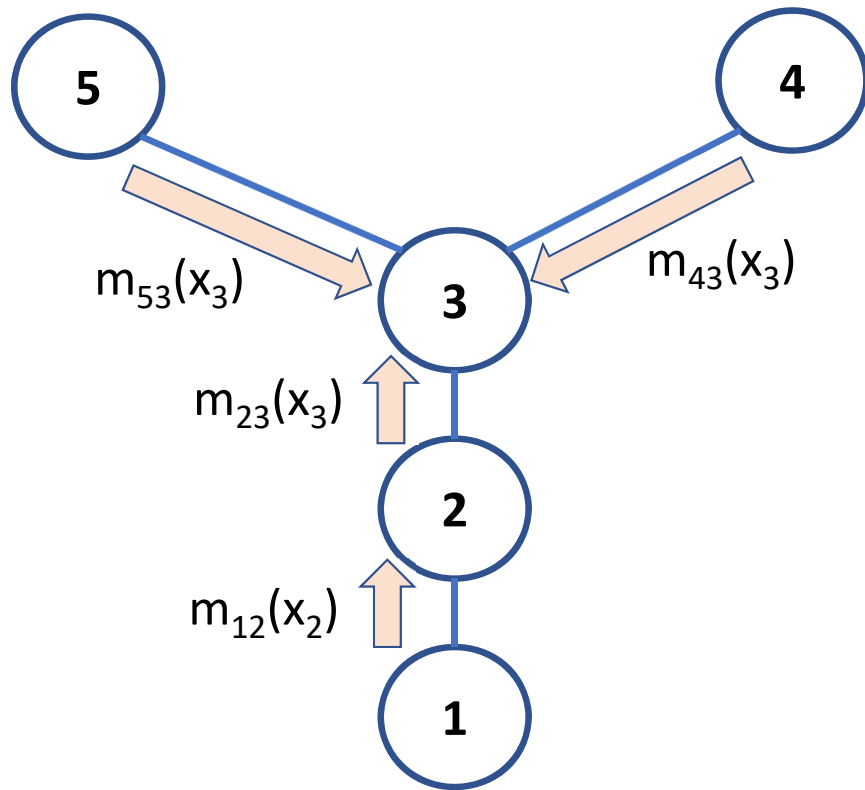
$$\delta(x_i, \bar{x}_j) = 1 \text{ if } i = j$$

$$\delta(x_i, \bar{x}_j) = 0 \text{ otherwise}$$

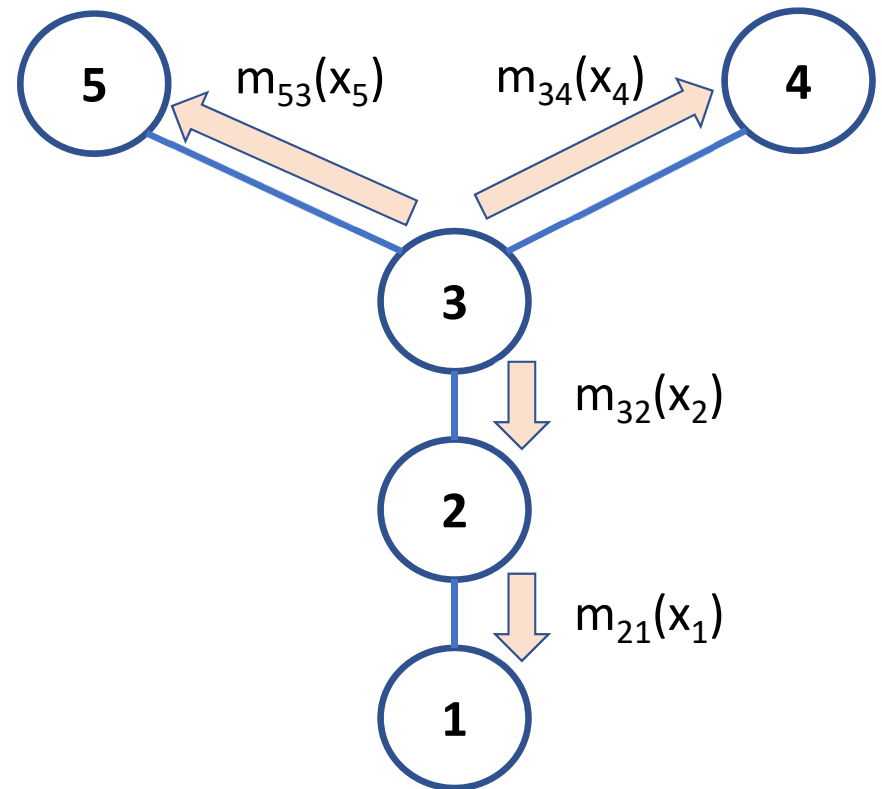
- Evidence reduces the number of states of variables

# The Belief Propagation Algorithm

Two message phases for each node



**Collect phase:** node receives messages from neighbors



**Emit phase:** The node emits messages to its neighbors

# The Belief Propagation Algorithm

The marginal distributions are computed

$$p(x_i) \propto \phi^E(x_i) \prod_{j \in N(i)} m_{ji}(x_i)$$

Where

$m_{ji}(x_i)$  is the message from the  $j$ th neighbor

$\phi(x_i)$  is the potential of the node

This is a form of **variable elimination!**



# Background for Junction Tree Algorithm

We need some machinery to understand the junction tree algorithm

- Factor graphs are used to form the junction tree
- Running intersection property is used to construct the junction tree
- Sum product algorithm used for belief propagation on junction tree
  - Or, get MAP value with max product algorithm
- Triangulation used to break cycles to form tree

# Factor Graphs

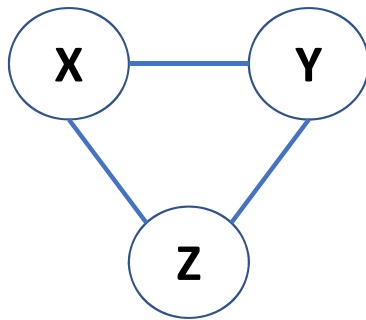
**Factor graphs** are another representation of graphical models

- A factor is a representation of an unnormalized conditional distribution
- Potentials are a specific case of factor
- Factor graphs are the basis of **message passing algorithms**
- Factor graphs include **variables and (intersection) factors**
- Constructing factor graphs relies on the **running intersection property**:

For each pair of cliques  $U$ ,  $V$  with intersection  $S$ , all cliques on the path between  $U$  and  $V$  contain  $S$ .

# Factor Graphs

The factor graph decomposition is **not unique**



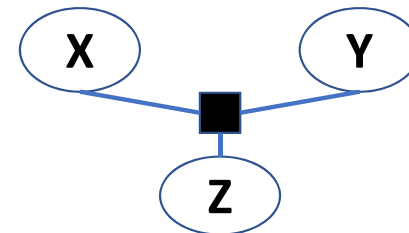
## Markov Network

Can use a single potential

$$p(X, Y, Z) = \phi(X, Y, Z)$$

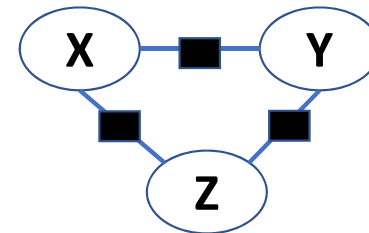
Or, 3 cliques

$$p(X, Y, Z) = \phi(X, Y)\phi(X, Z)\phi(Z, Y)$$



## Factor Network

Single factor

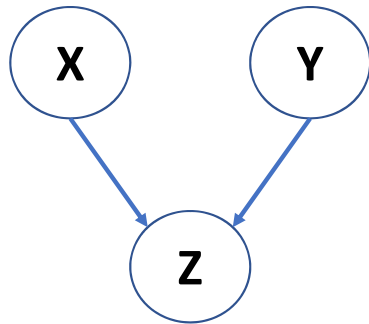


## Factor Network

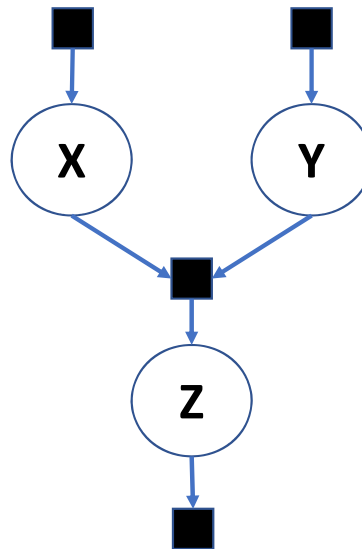
Three intersection factors

# Factor Graphs

Turn a DAG into a factor graph:



**Directed Graph**

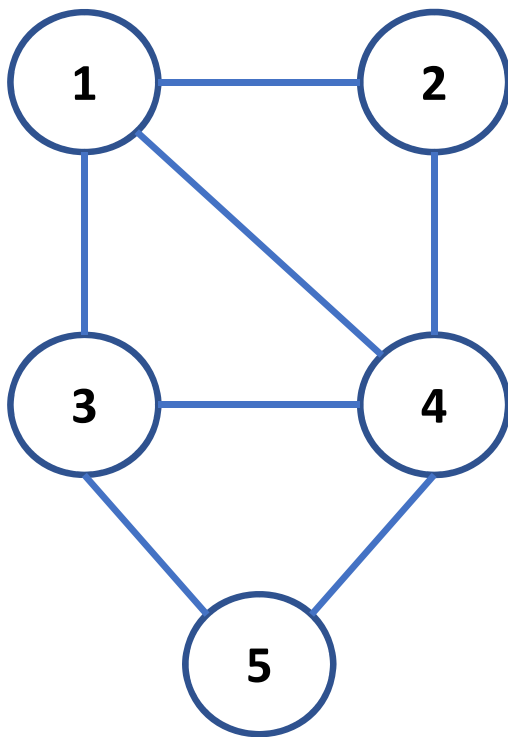


**Factor Graph**

- Start with a DAG
- There is one factor for the nodes
- Leaf nodes have factors
- Root node has a terminal factor

# Running Intersection Property

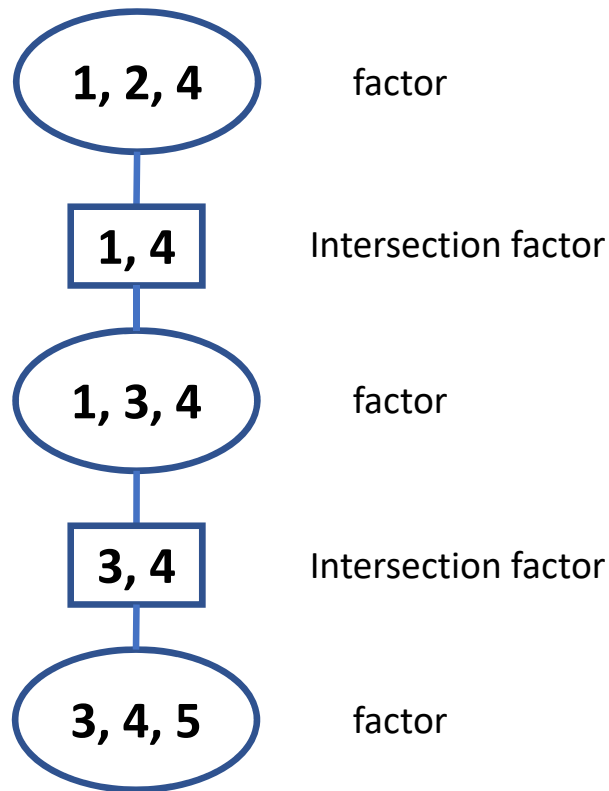
Example of running intersection property



- There are three maximal cliques:  
 $\{1,3,4\}$ ,  $\{1,2,4\}$ , and  $\{3,4,5\}$
- With intersections:  
 $\{1,4\}$  between  $\{1,3,4\}$  and  $\{1,2,4\}$   
 $\{3,4\}$  between  $\{1,2,4\}$  and  $\{3,4,5\}$
- In summary:  
node 4 is in 4 cliques and both intersections  
node 1 is in 2 cliques and one intersection  
node 3 is in 2 cliques and one intersection

# Running Intersection Property

Example of running intersection property



- Start with factor  $\{1, 2, 4\}$
- Add factor  $\{1, 3, 4\}$
- Intersection is factor  $\{1, 4\}$
- Add factor  $\{3, 4, 5\}$
- Intersection is factor  $\{3, 4\}$

# Sum-Product Algorithm

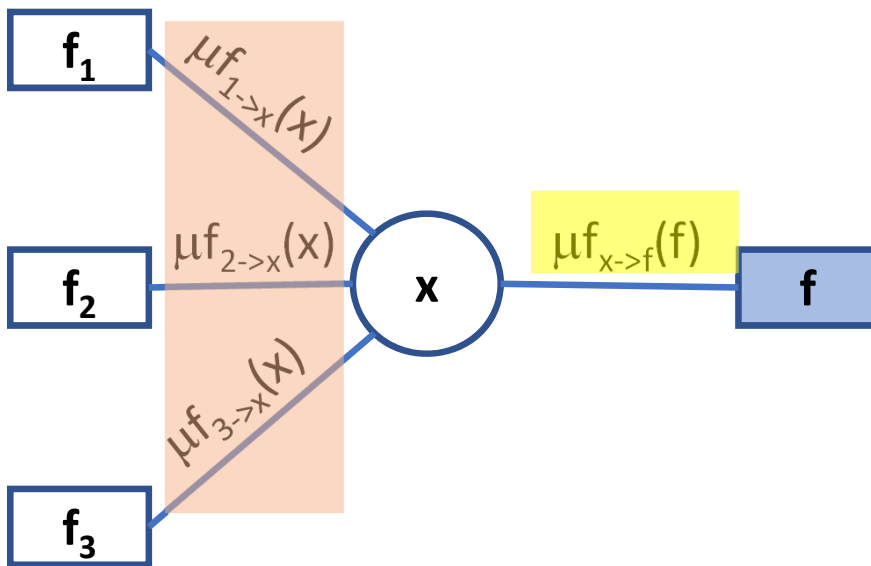
There are **two types of messages** in factor graphs

- The sum-product algorithm is constructed from these messages
- Compute the values for two types of messages
- Variable to factor message
- Factor to variable message

# Sum-Product Algorithm

Variable to factor message

$$\mu_{f_{x \rightarrow f}}(f)$$



- Start with variable  $x$
- Variable receives messages from factors
- Variable emits a message to factor
- The value of the message is the product of the messages from neighbor (na) factors

$$\mu_{x \rightarrow f}(x) = \prod_{h \in na(x) \setminus f} \mu_{h \rightarrow x}(x)$$

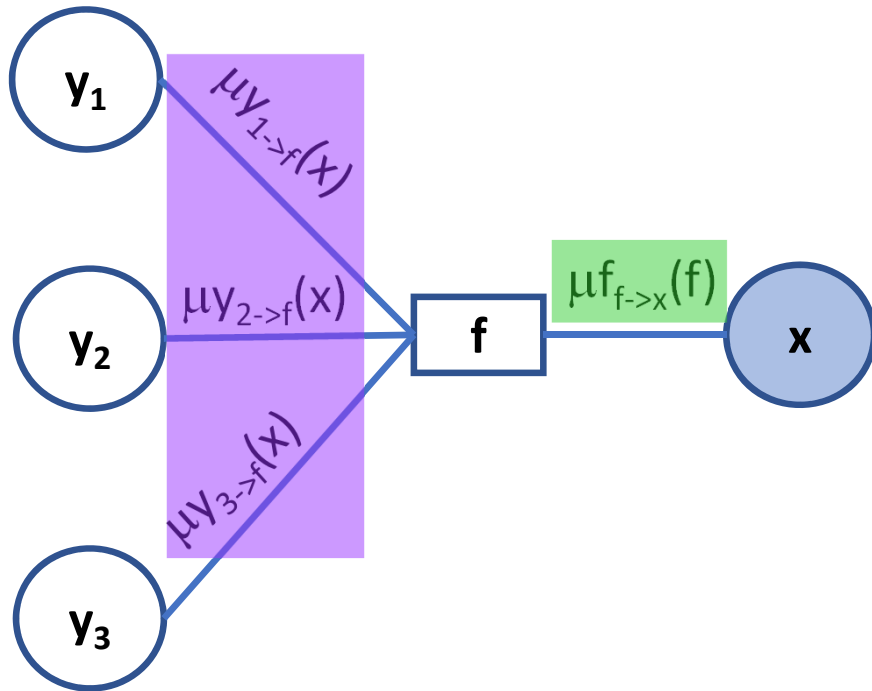
Where the notation  $\setminus f$  excludes the factor  $f$



# Sum-Product Algorithm

Factor to variable message

$$\mu_{f \rightarrow x}(f)$$



- Start with factor  $f$
- Factor receives messages from variables
- The factor emits a message to the variable
- The value of the message is the sum of product of the messages

$$u_{f \rightarrow x}(x) = \sum_{\chi_f \setminus x} \psi_f(\chi_f) \prod_{y \in na(f) \setminus x} u_{y \rightarrow f}(y)$$

Where the notation  $\setminus x$  excludes the current variable

And  $\sum_{\chi_f \setminus x} \psi_f(\chi_f)$  is the sum over all states of the variables  $\chi_f \setminus x$

# Max-Product Algorithm

Sometimes we just want the maximum or most probable value

- Use the **max-product algorithm**
- Max-product algorithm replaces the sum with a max operation
- The **maximum a posteriori (MAP)** value is found with argmax operation

# Max-Product Algorithm

## Formulation of the max-product algorithm

- Variable to factor message remains unchanged – no sum

$$u_{x \rightarrow f}(x) = \prod_{h \in na(x) \setminus f} u_{h \rightarrow x}(x)$$

- The factor to variable message is then:

$$u_{f \rightarrow x}(x) = \operatorname{argmax}_{\chi_f \setminus x} \phi_f(\chi_f) \prod_{y \in na(f) \setminus x} u_{y \rightarrow f}(y)$$

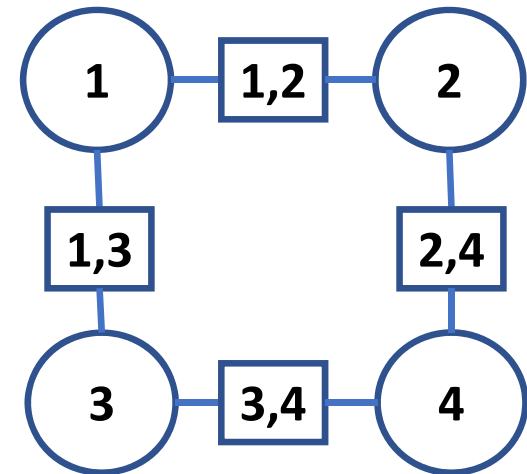
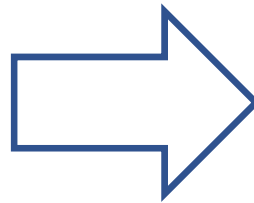
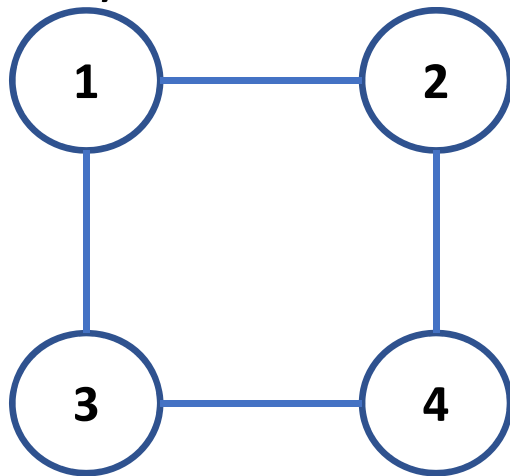
- Algorithm is a form of **linear programming**

See sections 13.1, 13.2 and 13.3 of Koller and Friedman for details

# Triangulation

Triangulation is a process of breaking cycles in an undirected graph

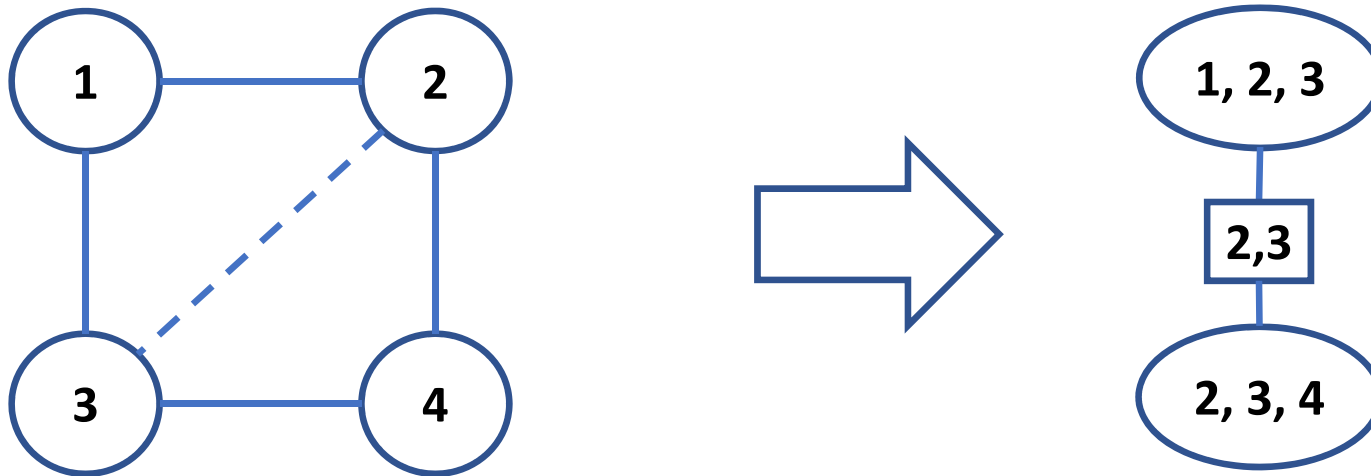
- The Markov network has a **cycle**
- Cycle has 4 or more nodes
- Trees cannot have cycles
- Turn Markov network into factor graph: the factor graph also has a cycle



# Triangulation

Triangulation is a process of breaking cycles in an undirected graph

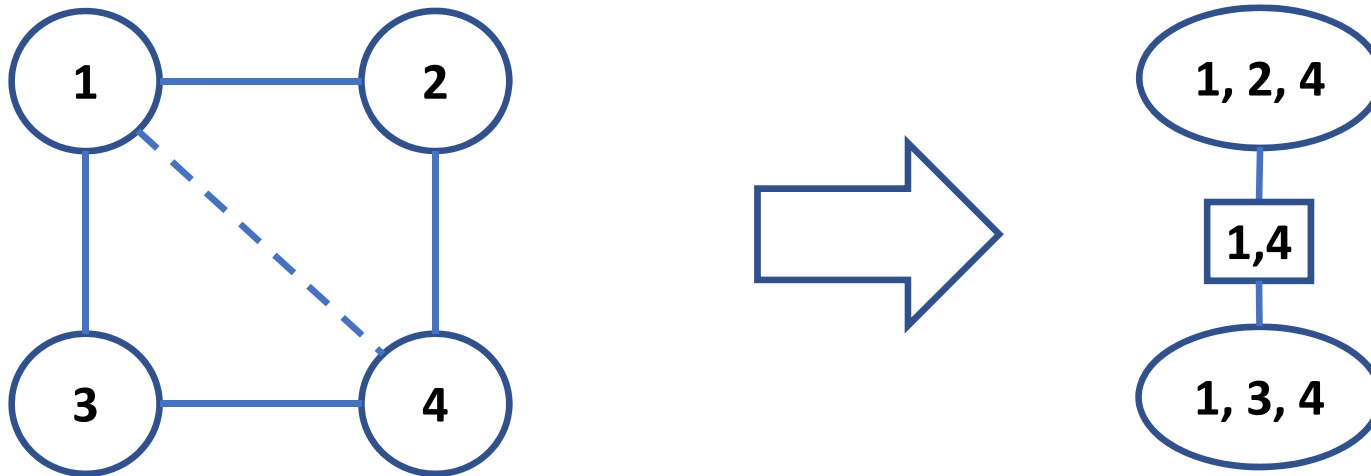
- How to break the cycle of the Markov network?
- Add an **edge to triangulate the graph**
- Turn Markov network into factor graph: **the factor graph is now a tree!**



# Triangulation

Triangulation is a process of breaking cycles in an undirected graph

- Or, add a **different edge** to triangulate the graph
- The factor graph is again a **tree**!
- The triangulation and factor graph are **not unique**!



# Junction Tree Algorithm

## Steps of the Junction Tree algorithm

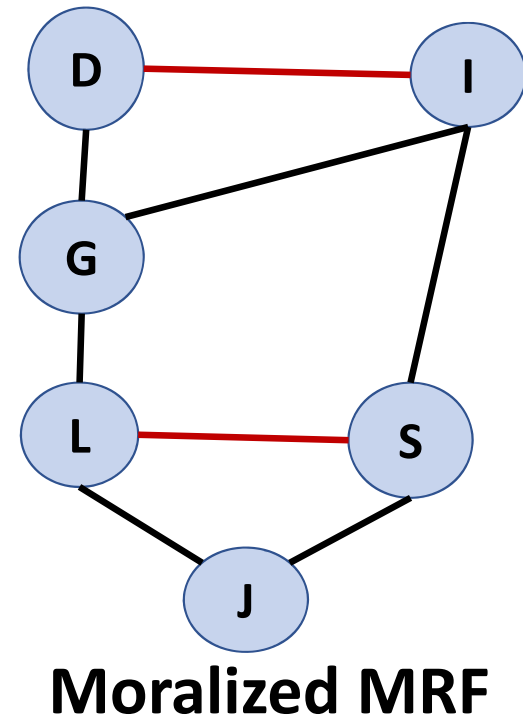
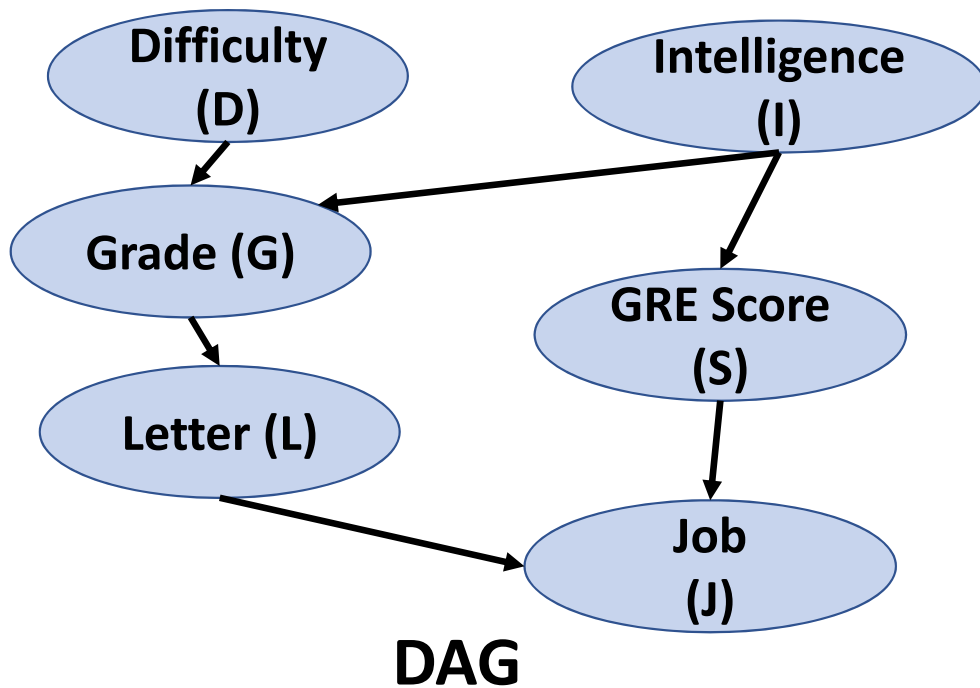
1. If Bayes network, moralize
2. If Bayes network, remove directional arrows
3. Triangulate the undirected graph if cycles
4. Build a clique tree
5. Build a junction tree, a factor graph
6. Choose a root
7. Populate the cliques
8. Perform belief propagation with sum-product algorithm

# Junction Tree Example

Start with the familiar student example

Remove direction from arrows

Moralize the graph

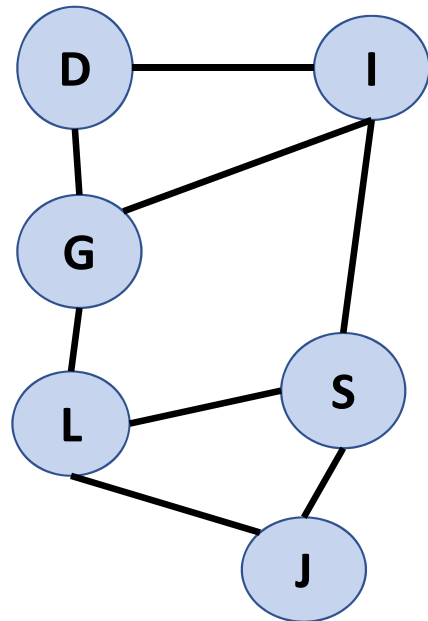




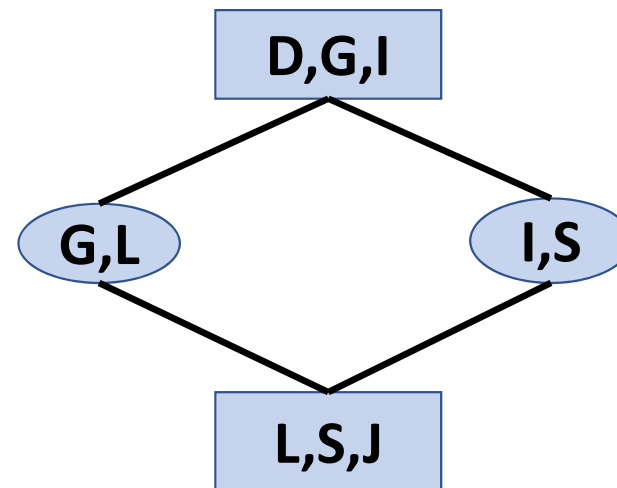
# Junction Tree Example

Try to build a clique tree

- Wait! This is **not** a tree!
- It has a cycle
- Must triangulate



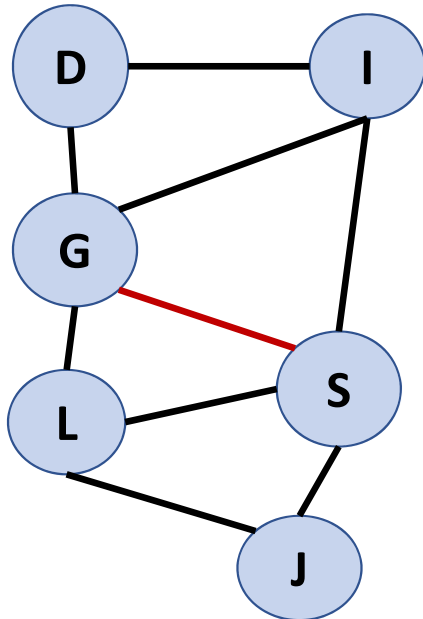
**Moralized MRF**



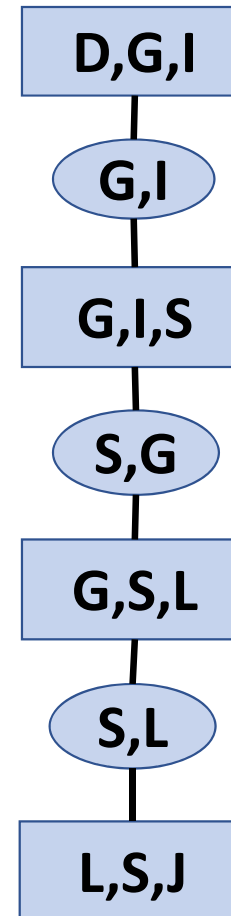
# Junction Tree Example

## Triangulate

- Build the clique tree using the **running intersection property**
- The chain is a tree so message passing works!



**Moralized MRF**



# Relationship Between Junction Tree and Variable Elimination

There is a strong relationship between the Junction tree and variable elimination algorithms

- Combining all remaining tables containing a variable eliminates it
- A node in the junction tree corresponds to the variable in variable elimination
- An arc in the junction tree shows the flow of data for the elimination computation

# Vocabulary

- **Exact inference algorithms** = methods to get a marginal distribution
- **Variable Elimination Algorithm** = marginalize variables one-by-one in order, to solve for a marginal distribution of 1 variable
- **Factor** = multidimensional table which assigns a value to a set of variables; representation of an unnormalized conditional distribution
- **Message Passing Algorithm** or **Belief Propagation Algorithm** = cascading queries that pass factors through a tree graph, to solve for a marginal distribution of 1 variable
- **Tree graph** = undirected acyclic graph, in which any pair of nodes are connected by exactly one path.

# Vocabulary

- **Message**  $m_{ji}(x_i)$  = the message from node  $j$  to node  $i$
- **Running Intersection Property** = for each pair of cliques  $U, V$  with intersection  $S$ , all cliques on the path between  $U$  and  $V$  contain  $S$ .
- **Triangulation** = a process of breaking cycles in an undirected graph
- **Cycle** has 4 or more nodes

# Key Points

- Faster computation is achievable by using distributions factored on a graph by independencies
- Categories of Inference on Graphical Models: exact inference, approximate
- 3 most common exact inference algorithms: variable elimination, message passing, junction tree
- Variable elimination algorithm: eliminate variables one by one; choice of variable order determines computational complexity
- Message passing algorithm: factors (not probabilities!) are passed along a tree graph; factors can be stored for speeding up later queries; messages “propagate belief”; linear time complexity

# Key Points

- Belief propagation algorithm: (similar to message-passing) leaf nodes emit messages, each node collects all neighbors' messages then emits messages to all its neighbors
- Factor graphs: another graphical model representation; the basis of message passing algorithms; have variables, factors, intersections
- Factor graph message types: variable to factor, factor to variable
- Junction tree algorithm: convert to Markov, triangulate, build clique tree, build factor graph, choose root, populate cliques, use sum-product algorithm to calculate marginal distribution