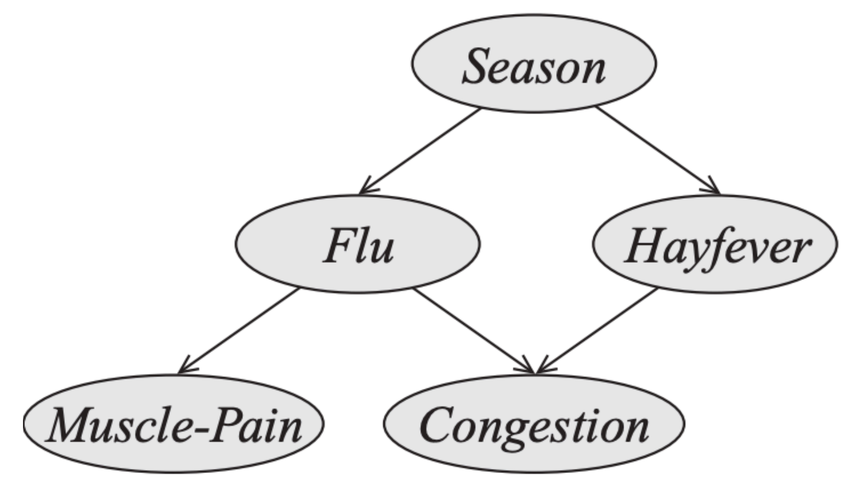
CS 7140: ADVANCED MACHINE LEARNING

Recap:Bayesian Network

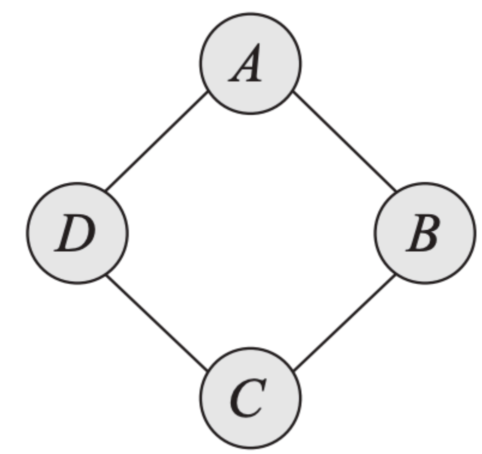
- Directed Acyclic Graph (DAG)
  - One node for each random variable X_i
  - One conditional distribution per node $P(X_i \mid Pa(X_i))$



- Bayesian Network chain rule:
$$P(X_1, X_2, \cdots, X_n) = \prod_i P(X_i \mid Pa(X_i))$$

Recap:Markov Random Field

- Undirected graph
  - One node for each random variable



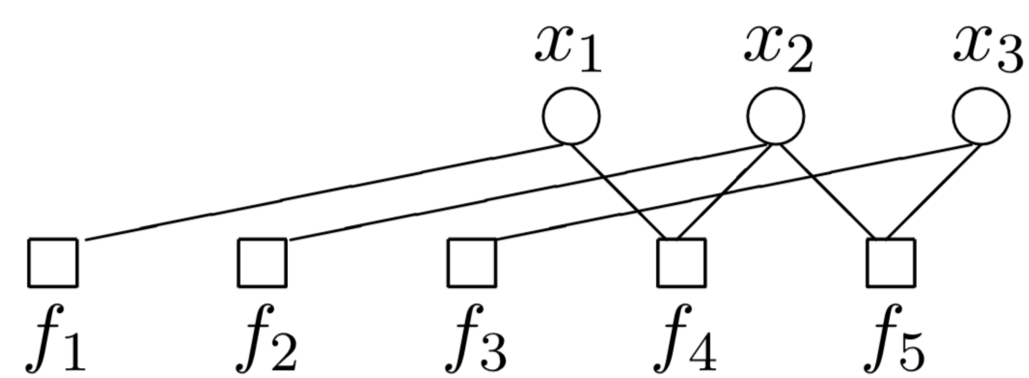
- Positive density
- Satisfy **Markov property**:
  - pairwise: $X_u \perp X_v | X_{V \setminus \{u,v\}}$
  - local: $X_v \perp X_{V \setminus N(v)} | X_{N(v)}$
  - global: $X_A \perp X_b | X_S$

$$(A \perp C \mid B, D)$$
$$(B \perp D \mid A, C)$$

$$P(A, B, C, D) = \frac{1}{Z}\phi_1(A, B)\phi_2(B, C)\phi_3(C, D)\phi_4(A, D)$$

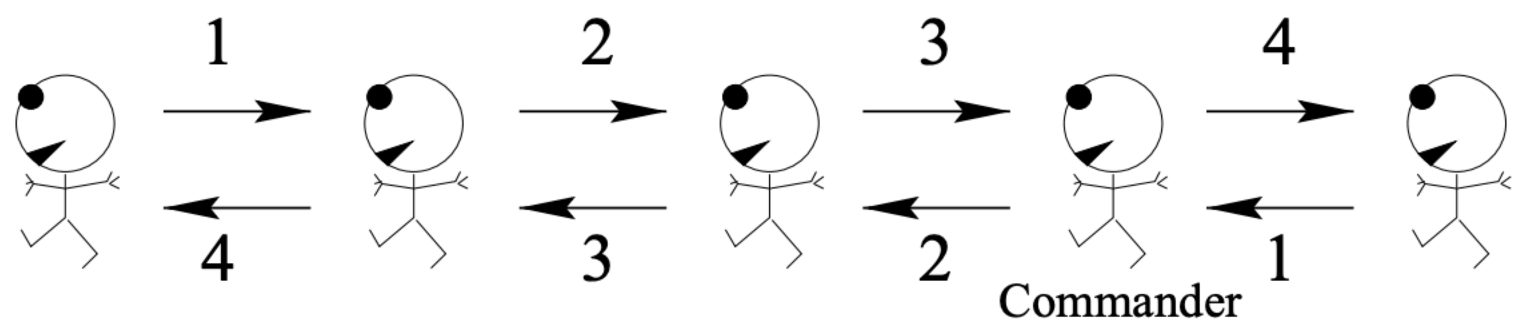
Factor Graph

- Represent variables and factors separation explicitly
- Bi-partie graph: variables and factors
- edges between variables and factors to indicate dependency



MESSAGE PASSING

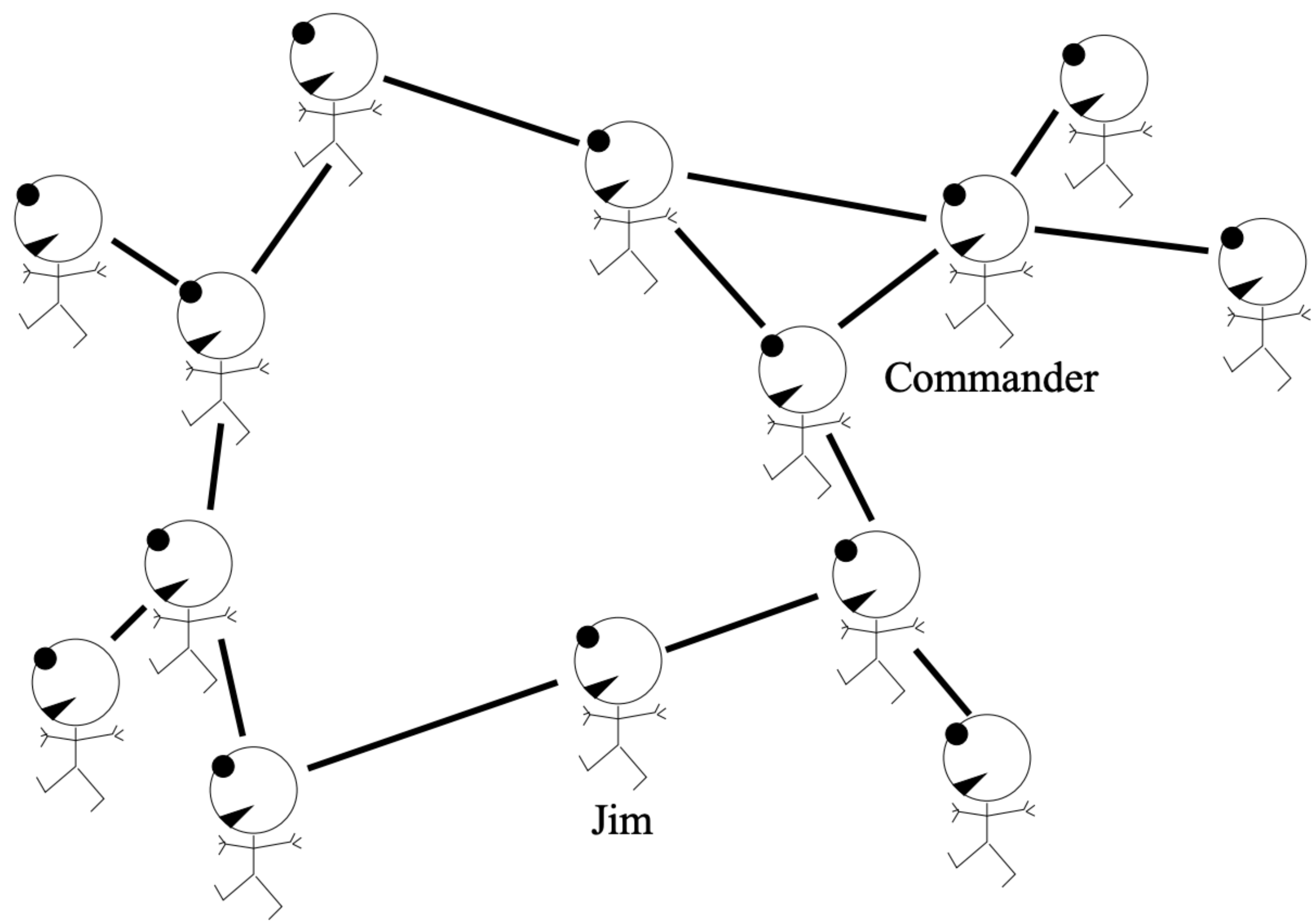
Counting Soldiers



1. If you are the front soldier, say 'one' to the soldier behind you.
2. If you are the rearmost soldier, say 'one' to the soldier in front of you.
3. If a soldier ahead of or behind you says a number to you, add one to it, and say the new number to the soldier on the other side.

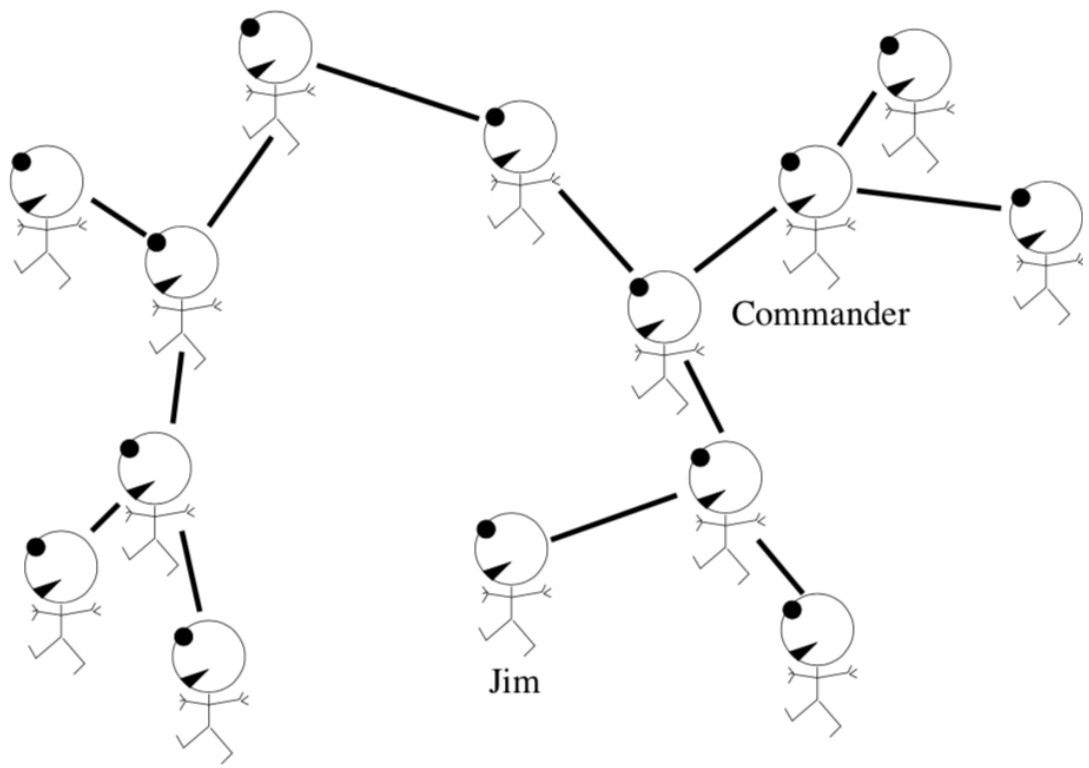
This solution requires only **local** communication hardware and simple computations (storage and addition of integers).

Separation



- A swarm of guerillas cannot be counted due to **cycles**

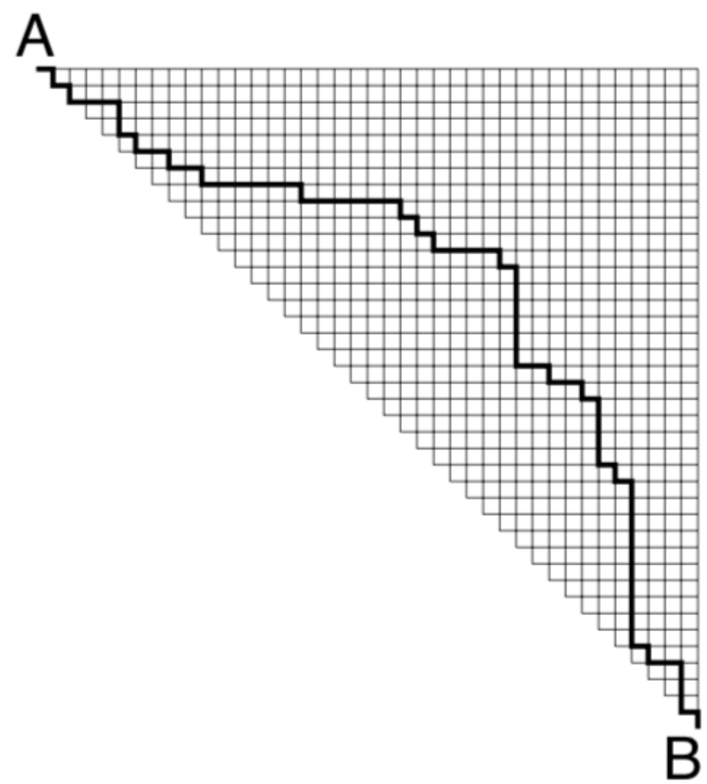
Separation



1. Count your number of neighbours, N.
2. Keep count of the number of messages you have received from your neighbours, m, and of the values v_1, v_2, $\ldots$, v_N of each of those messages. Let V be the running total of the messages you have received.
3. If the number of messages you have received, m, is equal to $N-1$, then identify the neighbour who has not sent you a message and tell them the number $V+1$.
4. If the number of messages you have received is equal to N, then:
   (a) the number $V+1$ is the required total.
   (b) for each neighbour n {
   say to neighbour n the number $V+1-v_n$.
   }

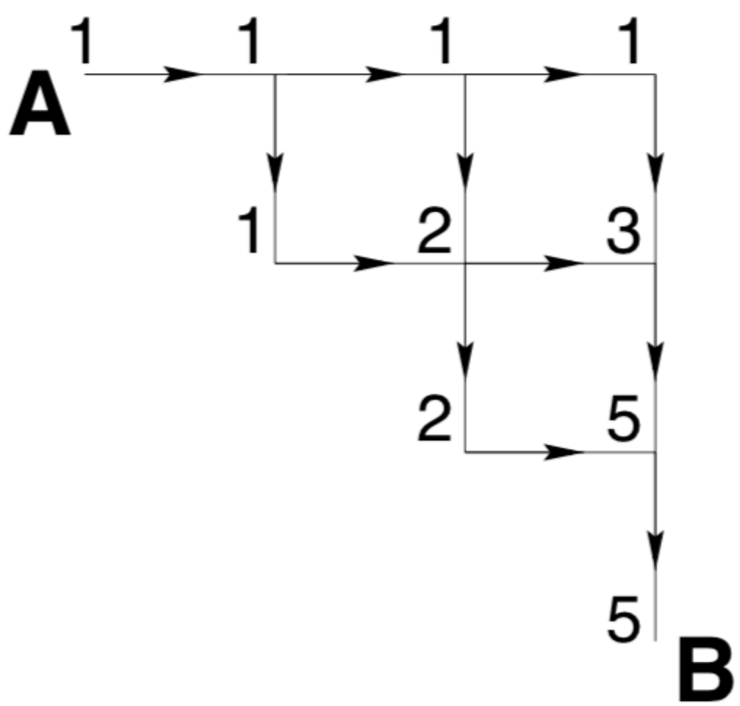
- message passing algorithm for counting on a tree.

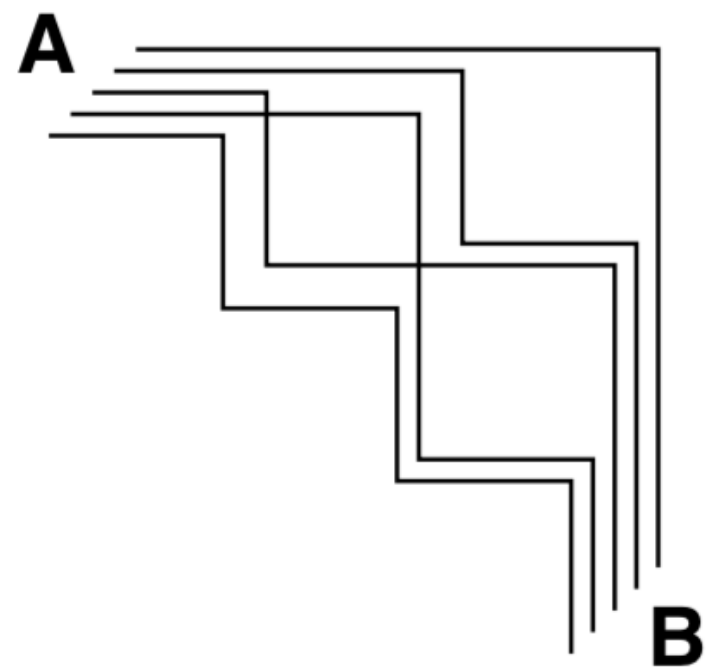
Path Counting



- **How many paths are there from A to B?**
- If a random path from A to B is selected, what is the probability that it passes through a particular node in the grid?
- How can a random path from A to B be selected?

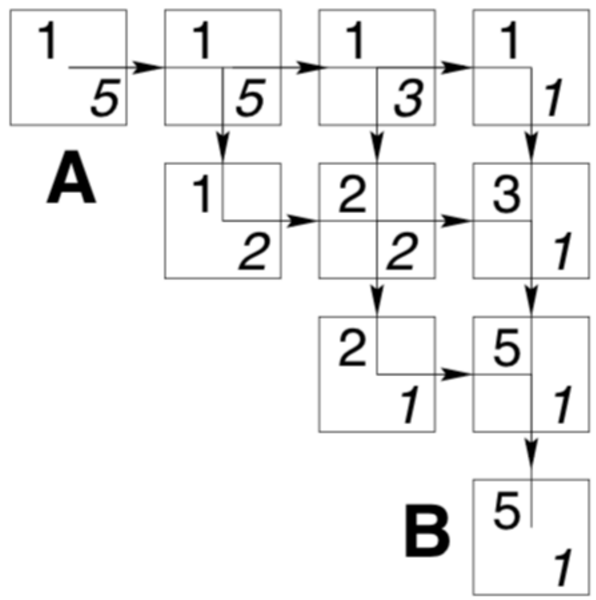
Message Passing





- Pick a point P in the grid.
- Every path from A to P must come in to P through one of its upstream neighbors
- Number of paths from A to P: sum of number of paths from A to each of those neighbors.

Path Counting

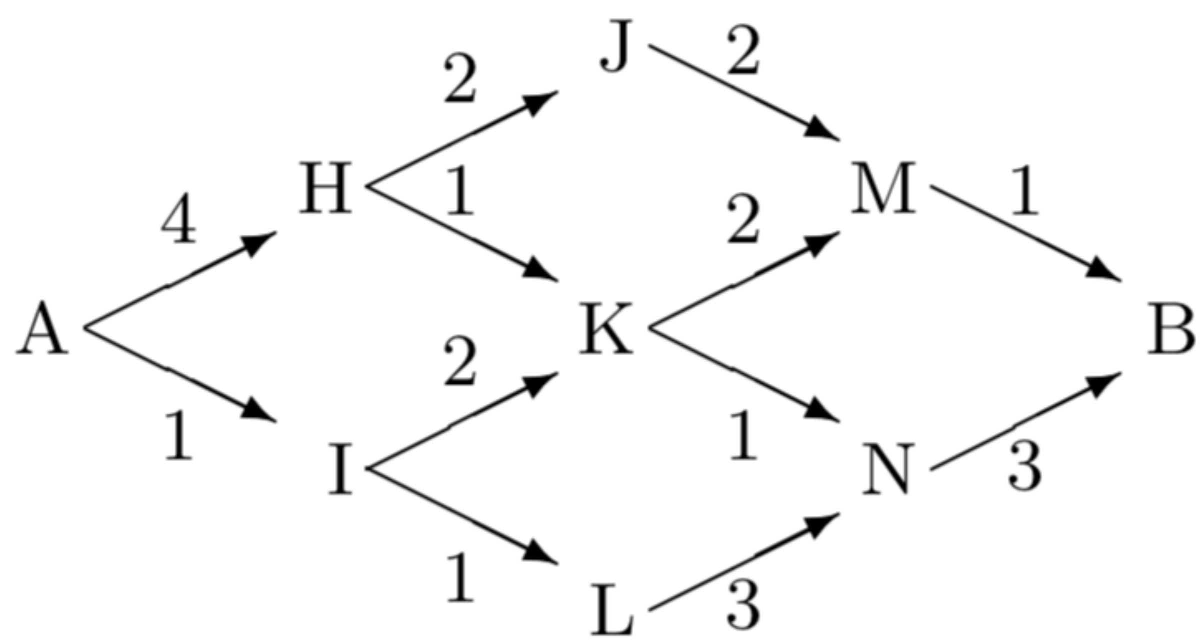


- forward pass F
- backward pass B
- total number of paths passing each node: F X B

sum-product algorithm

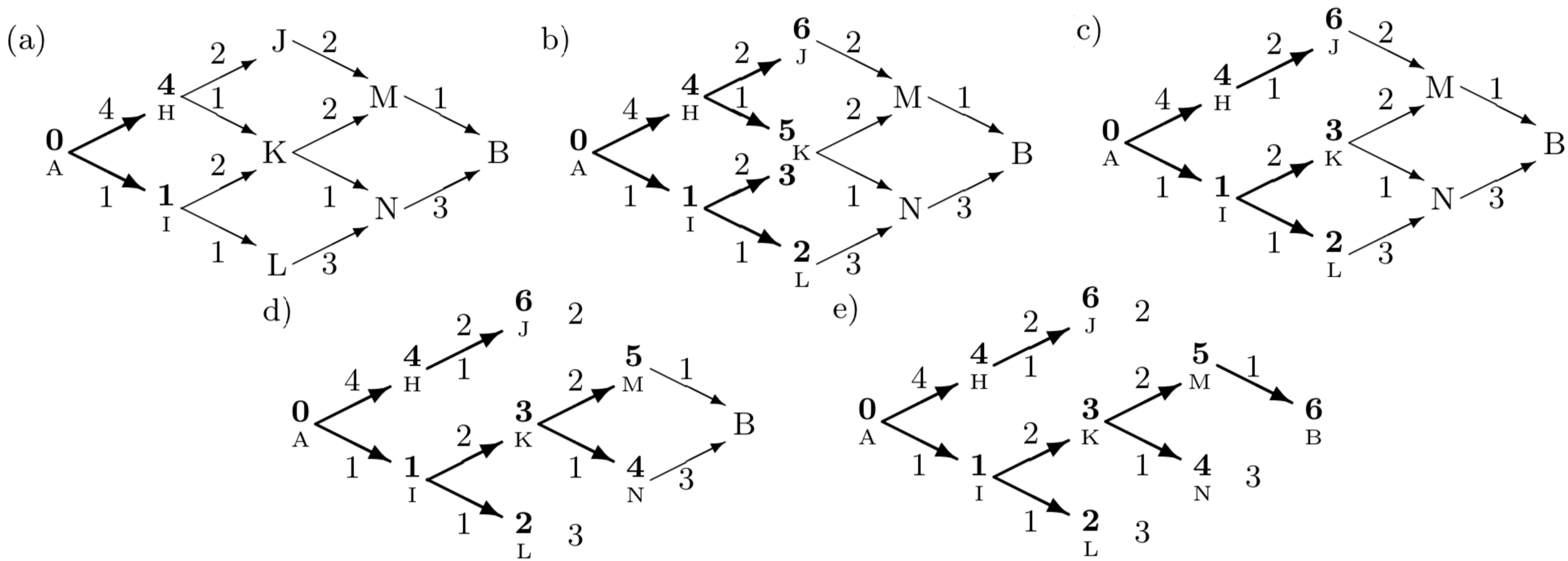
- How many paths are there from A to B?
- **If a random path from A to B is selected, what is the probability that it passes through a particular node in the grid?**
- How can a random path from A to B be selected?

Finding Lowest Cost Path



- You wish to travel as quickly as possible from Ambridge (A) to Bognor (B).
- The various possible routes are shown above, along with the cost in hours of traversing each edge in the graph.
- find for each node what the cost of the lowest-cost path to that node from A is.

Finding Lowest Cost Path



- computed by message passing, also known as **min-sum algorithm** or **Viterbi Algorithm**
- We deduce that the lowest-cost path is A–I–K–M–B.

Key Ideas

- some global functions have a separability property
- can be computed efficiently with **message passing**
- can also be solved with a deep neural network approach

EXACT INFERENCE

The General Problem

- A function P over N random variables is defined as the product of M factors

$$P(\mathbf{x}) = \prod_{m=1}^{M} f(\mathbf{x}_m)$$

- Normalized function

$$P(\mathbf{x}) = \frac{1}{Z} \prod_{m=1}^{M} f(\mathbf{x}_m)$$

- Normalization constant

$$Z = \sum_{\mathbf{x}} \prod_{m=1}^{M} f(\mathbf{x}_m)$$

The General Problem

- The normalization problem:

$$Z = \sum_{\mathbf{x}} \prod_{m=1}^{M} f(\mathbf{x}_m)$$

- The marginalization problem:

$$Z_n(x_n) = \sum_{\{x_{n'}\},\ n' \neq n} P(\mathbf{x})$$

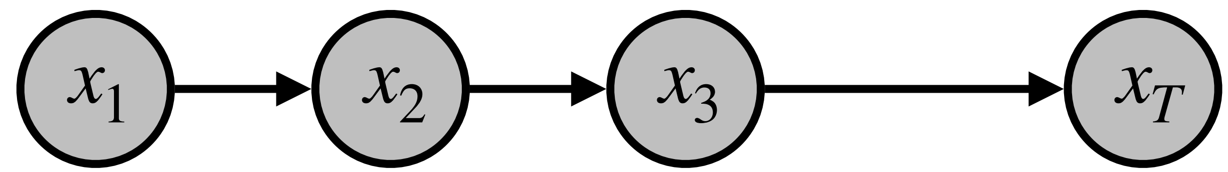
Variable Elimination

- Push in certain variables into the product

- Eliminate a variable at each time

- Different orderings may dramatically alter the running time

- It is NP-hard to find the best ordering



$$p(x_1) = \sum_{x_2, \cdots, x_{T-1}} p(x_1) \prod_{t=2}^{T} p(x_t \mid x_{t-1})$$

$$p(x_T) = \sum_{x_{T-1}} p(x_T \mid x_{T-1}) \sum_{T-2} p(x_{T-1} \mid x_{T-2}) \cdots \sum_{x_1} p(x_2 \mid x_1) p(x_1)$$

$$p(x_T) = \sum_{x_{T-1}} p(x_T \mid x_{T-1}) \sum_{T-2} p(x_{T-1} \mid x_{T-2}) \cdots \sum_{x_2} p(x_3 \mid x_2) \tau_2$$

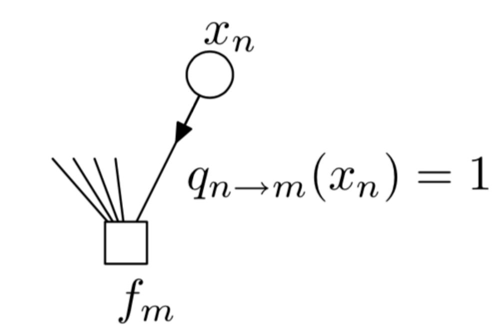
Sum-Product Algorithm

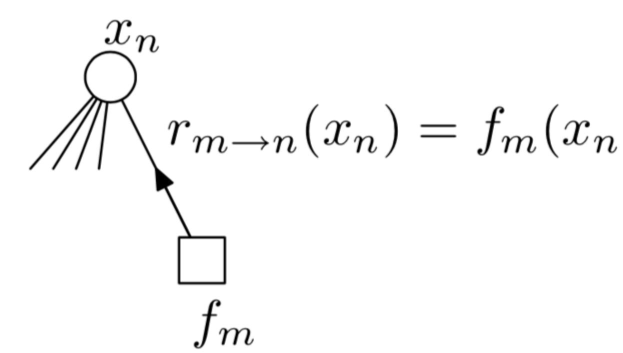
From variable to factor:

$$q_{n\to m}(x_n) = \prod_{m' \in \mathcal{M}(n)\backslash m} r_{m'\to n}(x_n). \tag{26.11}$$

From factor to variable:

$$r_{m\to n}(x_n) = \sum_{\mathbf{x}_m \backslash n} \left(f_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N}(m)\backslash n} q_{n'\to m}(x_{n'}) \right). \tag{26.12}$$





Two types passing along the edges in the factor graph:

- messages $q_{n\to m}$ from variable nodes to factor nodes
- messages $r_{m\to n}$ n from factor nodes to variable nodes.

The General Problem

- The normalization problem:

$$Z = \sum_{\mathbf{x}} \prod_{m=1}^{M} f(\mathbf{x}_m) \longrightarrow Z = \sum_{x_n} Z_n(x_n)$$

- The marginalization problem:

$$Z_n(x_n) = \sum_{\{x_{n'}\},\ n' \neq n} P(\mathbf{x}) \longrightarrow Z_n(x_n) = \prod_{m \in M(n)} r_{m \to n}(x_n)$$

A Factorization View

- Sum-product algorithm reexpresses the function

$$P(\mathbf{x}) = \prod_{m=1}^{M} \phi_m(\mathbf{x}_m) \prod_{n=1}^{N} \psi_n(x_n)$$

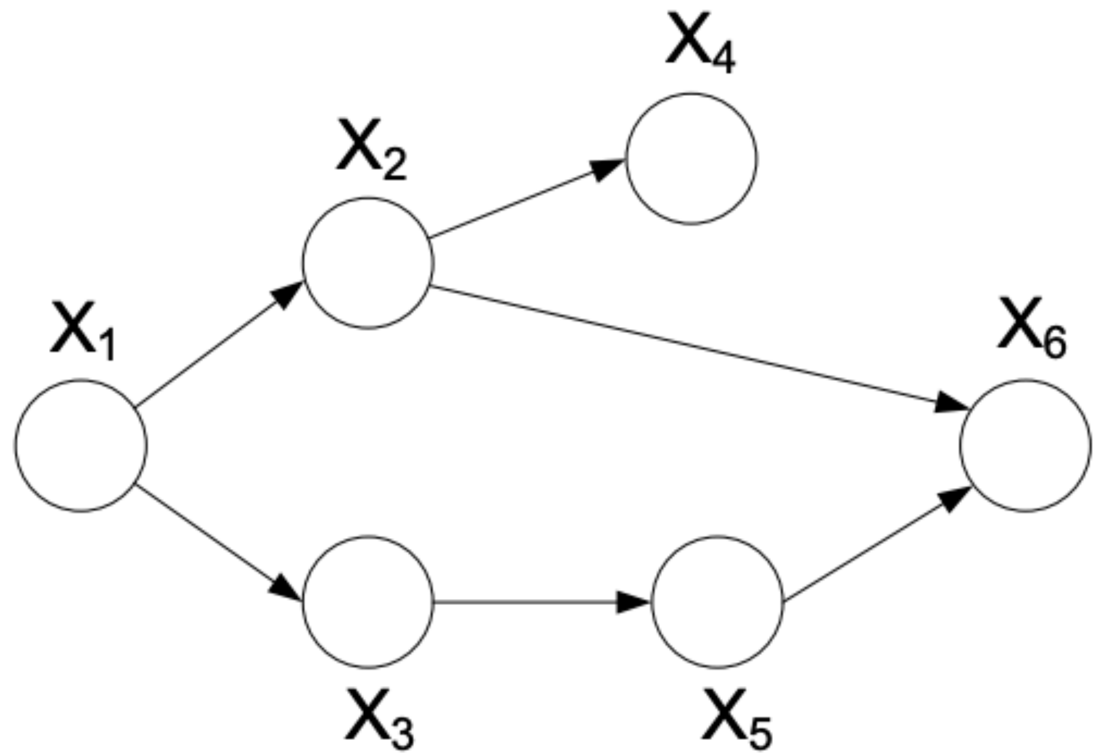
- Each factor ϕ_m is a factor node, and each $\psi_n(x_n)$ is a variable node

- $$\psi_n(x_n) = \prod_{m \in M(n)} r_{m \to n}(x_n) \quad \phi_m(\mathbf{x}_m) = \frac{f(\mathbf{x}_m)}{\prod_{n \in N(m)} r_{m \to n}(x_n)}$$

The min-sum algorithm

- **The maximization problem**. Find the setting of x that maximizes the product $P(\mathbf{x})$.
- Replace **add** and **multiply** with **max** and **multiply**
- Carried out in negative log likelihood —> **min-sum**
- As known as Viterbi algorithm

Junction Tree Algorithm



- What if the factor graph is not a tree?
  - **Exact**: junction tree
  - **Approximate**: Monte Carlo/Variational Method

Junction Tree Algorithm

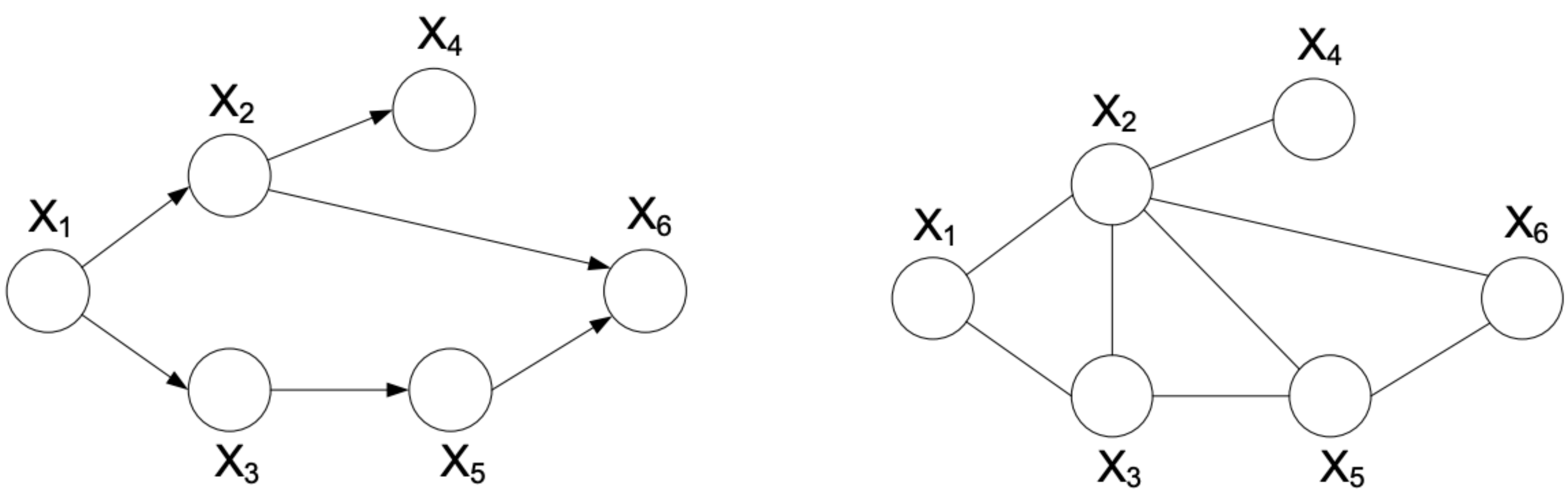
Use independence to convert graphs to trees

- **Moralize the graph**: directed to undirected
- **Triangulate the graph**: separation
- **Build a junction tree**: graph to tree
- **message passing**: sum product algorithm

Moralization

Convert a directed graph to a undirected graph

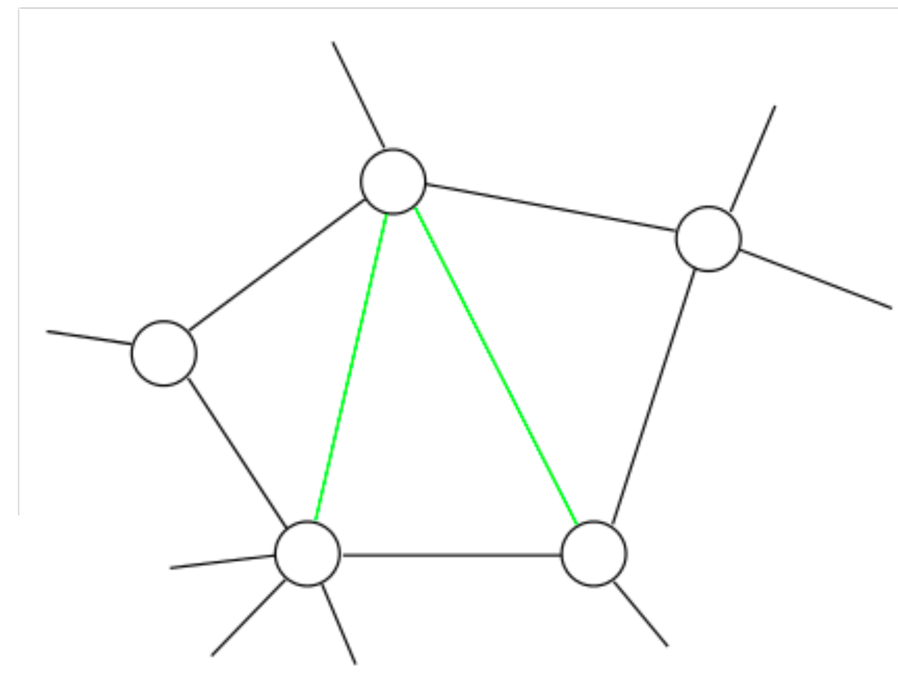
- Connect nodes that have common children
- Drop the arrows

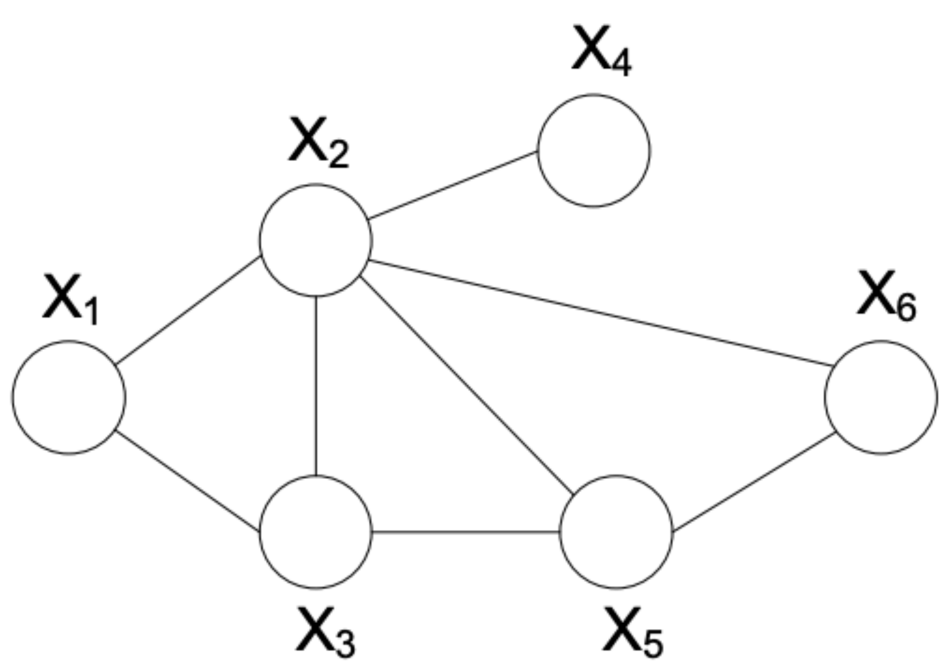


Triangulation

Make the graph chordal (no cycle/loops)

- Add links until there is no chordless cycle of 4 or more nodes
- Every induced cycle has exactly 3 nodes

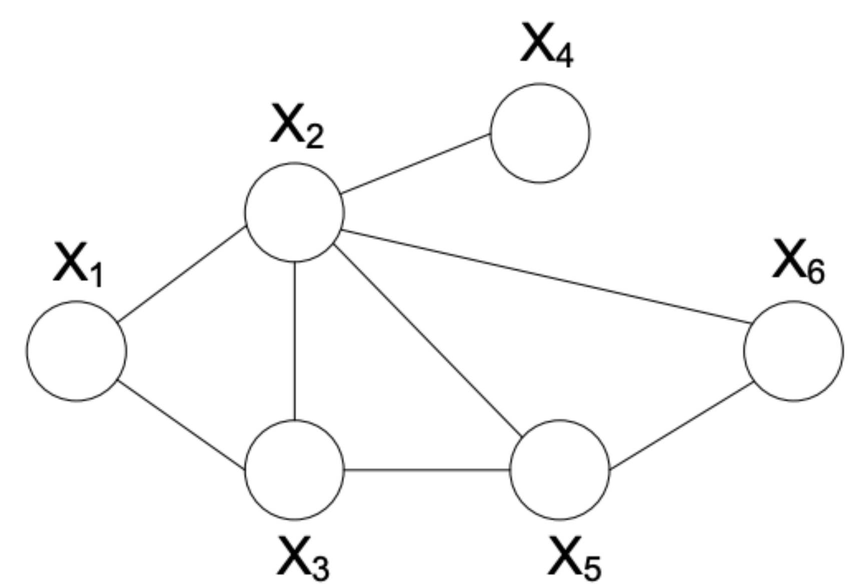


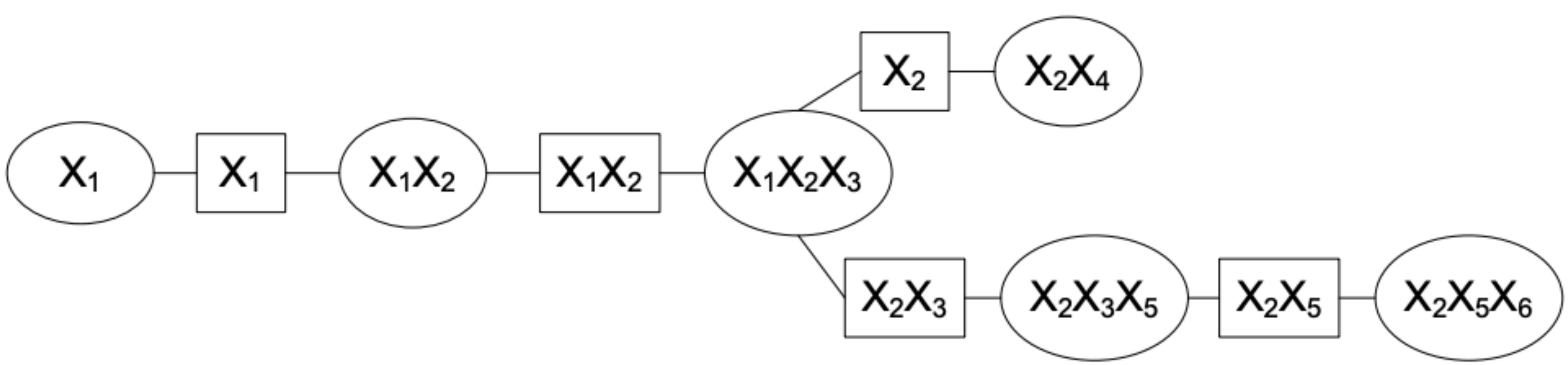


Build A Junction Tree

- Each node is a clique of variables, each edge is a potential function

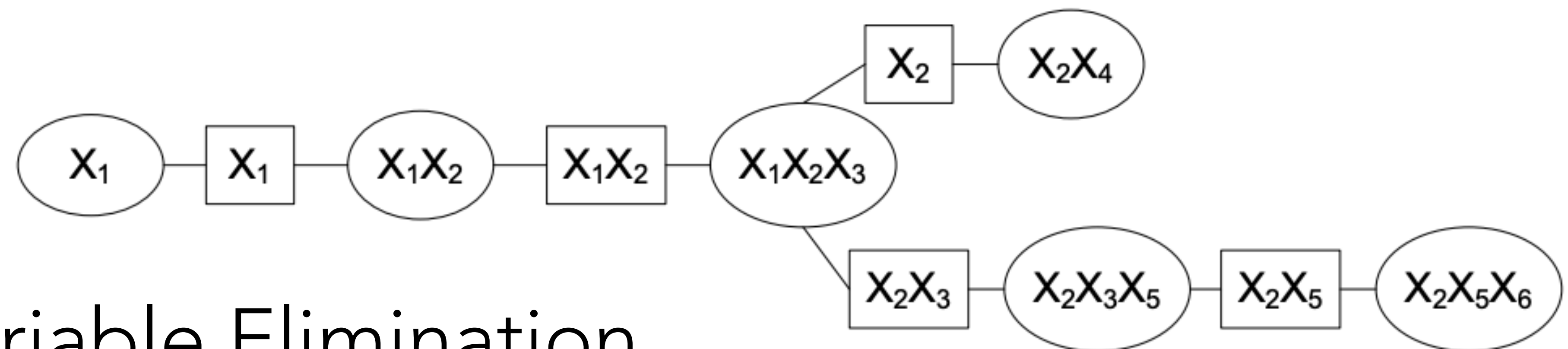
- Separator nodes contain clique intersection of variables





Message Passing

- Write a junction tree as potentials of its nodes:



- Variable Elimination

$$p(x_1) = \phi(x_1) \sum_{x_2} \phi(x_1, x_2) \sum_{x_3} \phi(x_1, x_2, x_3) \sum_{x_4} \phi(x_2, x_4) \sum_{x_5} \phi(x_2, x_3, x_5) \sum_{x_6} \phi(x_2, x_5, x_6)$$

- Message Passing

From variable to factor:

$$q_{n \to m}(x_n) = \prod_{m' \in \mathcal{M}(n) \setminus m} r_{m' \to n}(x_n). \tag{26.11}$$

From factor to variable:

$$r_{m \to n}(x_n) = \sum_{\mathbf{x}_m \setminus n} \left(f_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N}(m) \setminus n} q_{n' \to m}(x_{n'}) \right). \tag{26.12}$$