

# Inference in graphical models

CIS 620 Spring 2005  
Fernando Pereira

# Using MRFs

- *Inference*: given an MRF
  - Find most probable variable assignment
  - Find marginal probability of a partial assignment
- *Learning*:
  - Given model structure (variables, graph, and parameterized potentials)
  - Given training set of assignments
  - Find potential parameters maximizing training set probability (more refined versions possible)

# MRF inference

- Compute:
  - Best assignment
  - Probability of a partial assignment
- Naïve approach: enumerate all possible assignments (compatible with the partial assignment)
  - $O(k^v)$  for  $v$   $k$ -valued variables
- Can we do better?

# Marginalization

- Sum over all the variables we don't care about

$$\sum_{\mathbf{x} \sim \{x_A\}} f(\mathbf{x}) = \sum_{\mathbf{y}: y_A = x_A} f(\mathbf{y})$$

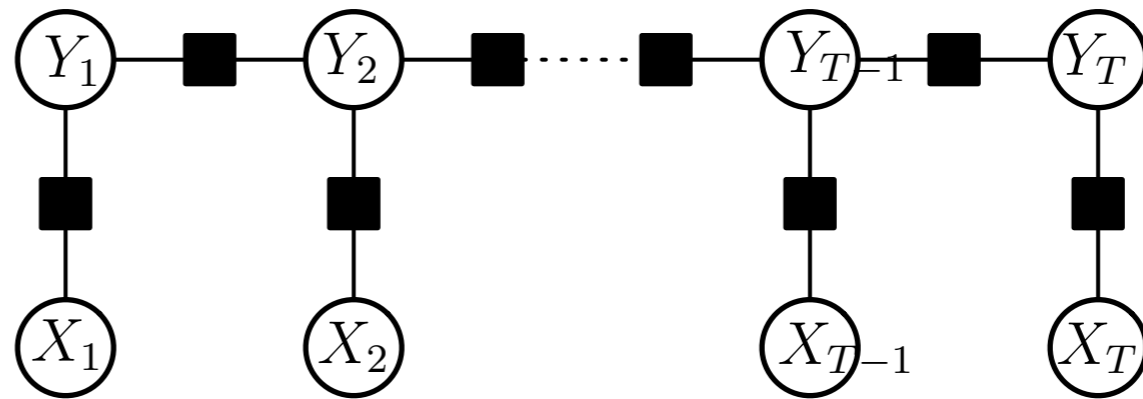
- *Main idea*: factor the sum according to the graph so that it can be more efficient

# Factor graphs

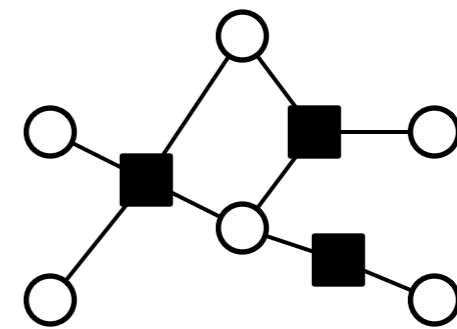
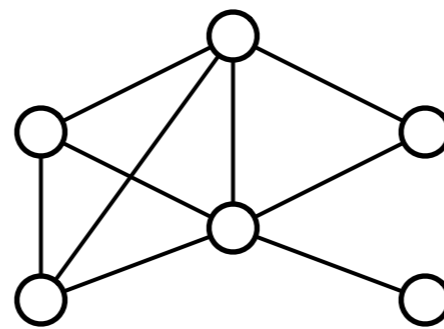
- Undirected graphical models are “too concrete”:
  - The graph is just a way of describing a set of clique potentials
  - Why not do that directly?
- Two kinds of entities
  - variables
  - *factors*: compute a potential from some variables

# Examples

HMM



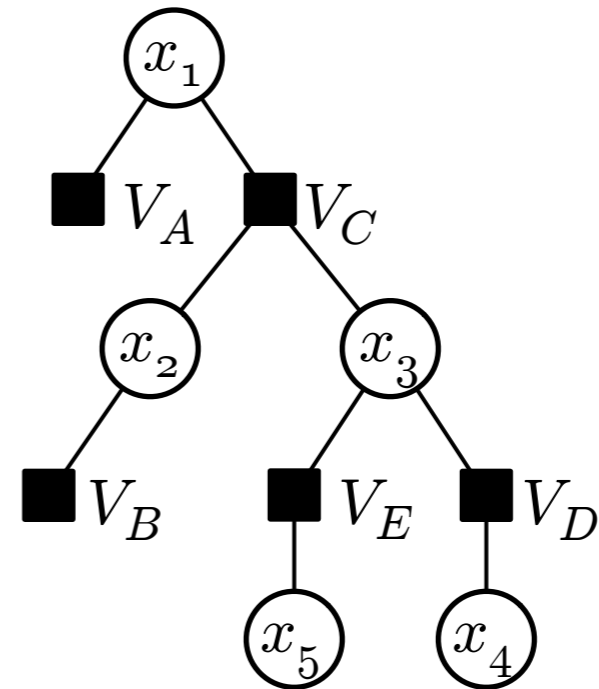
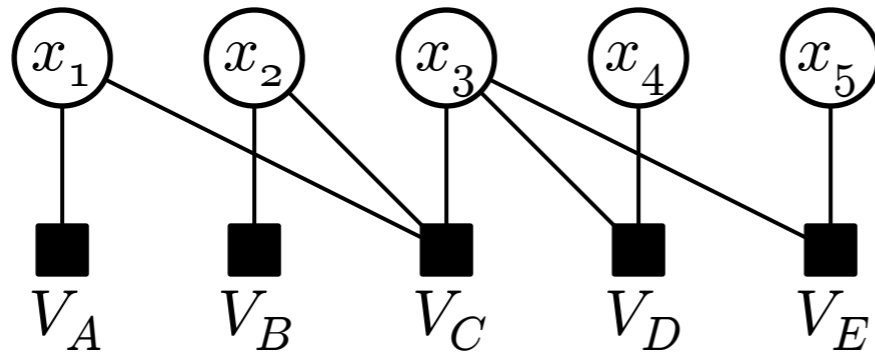
network with cycles



# Using a Factor Tree

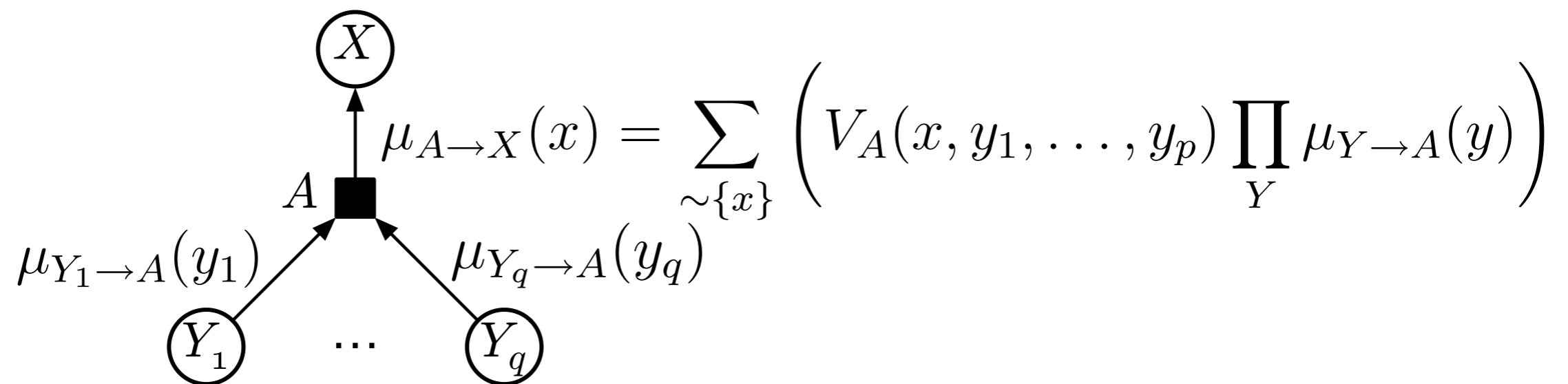
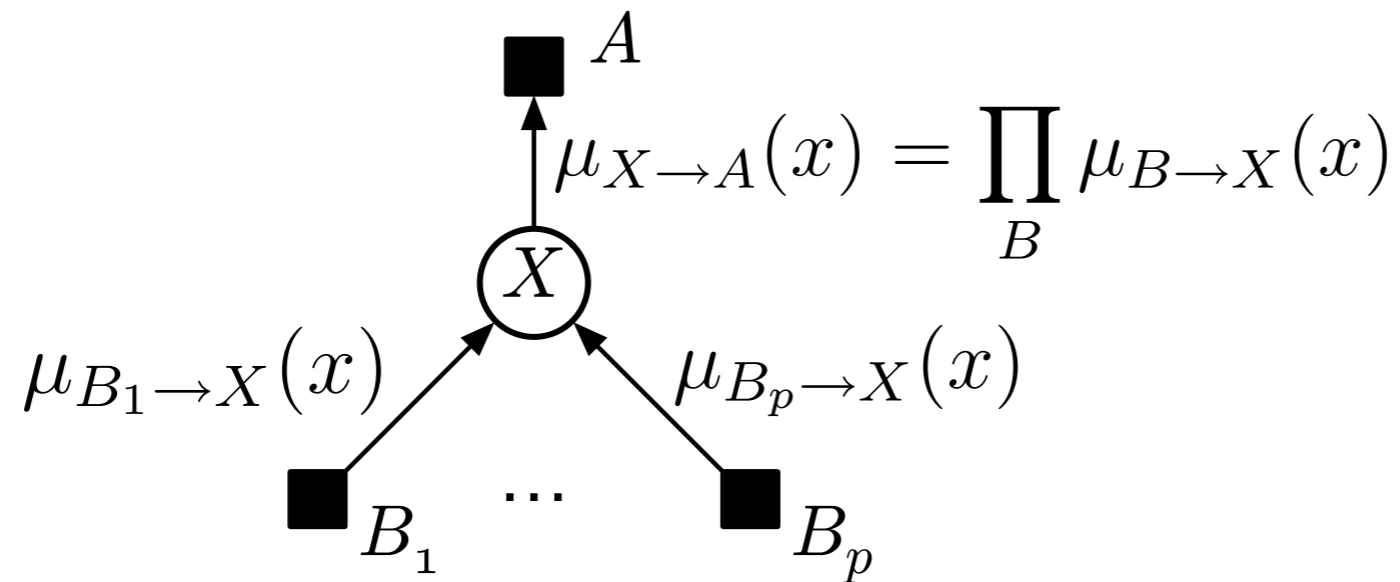
$$P(\mathbf{x}) = V_A(x_1)V_B(x_2)V_C(x_1, x_2, x_3)V_D(x_3, x_4)V_E(x_3, x_5)$$

$$P(x_1) = V_A(x_1) \left( \sum_{x_2} V_B(x_2) \left( \sum_{x_3} V_C(x_1, x_2, x_3) \left( \sum_{x_4} V_D(x_3, x_4) \right) \left( \sum_{x_5} V_E(x_3, x_5) \right) \right) \right)$$



$$P(x_1) = V_A(x_1) \times \sum_{\sim\{x_1\}} \left( V_B(x_2) \times V_C(x_1, x_2, x_3) \times \left( \sum_{\sim\{x_3\}} V_D(x_3, x_4) \right) \times \left( \sum_{\sim\{x_3\}} V_E(x_3, x_5) \right) \right)$$

# Message Propagation



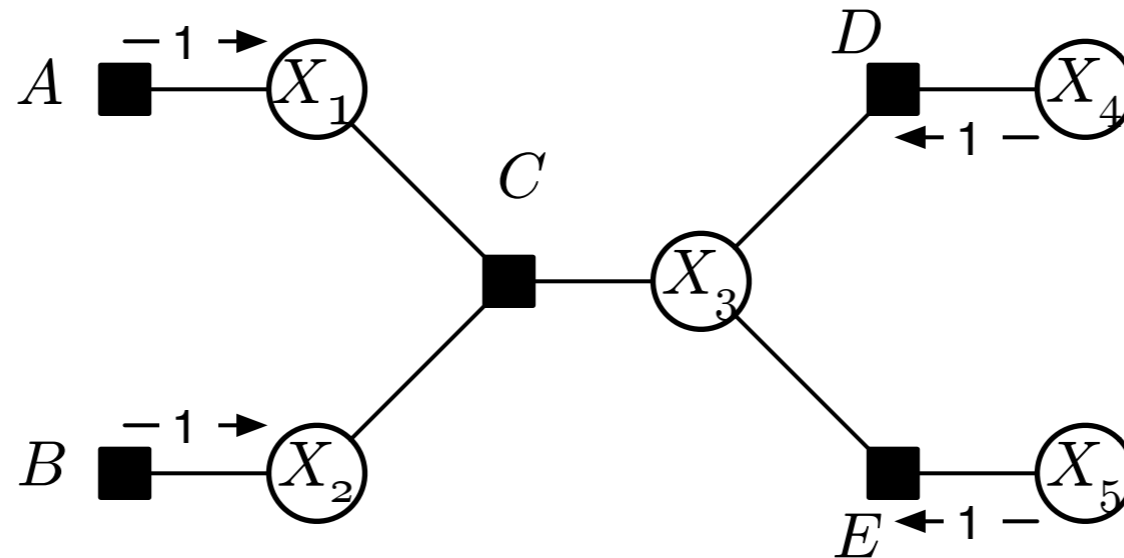
# All the Marginals

- Often, we want to compute several (all) marginals
- We want to share as much as possible of the computation
- Main idea: propagate messages in both directions
- Example you already know: the forward-backward algorithm

# Sum-Product Algorithm

- For each edge  $e$  leaving vertex  $v$ , if  $v$  has received messages on all other edges, send the appropriate message out on  $e$ .
- A leaf  $v$  has a single edge  $e$ , so it can send out a message on  $e$  immediately.
- Terminate when every edge has sent messages in both directions.
- The marginal for  $v$  is the product of all messages incoming into  $v$ .

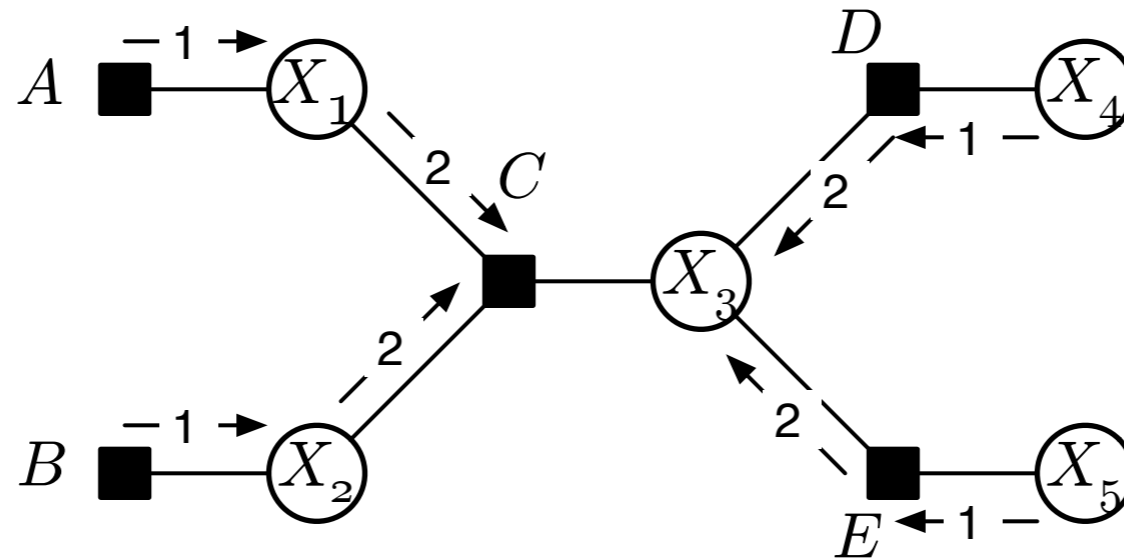
# Running the Algorithm



①

$$\begin{aligned}\mu_{A \rightarrow X_1}(x_1) &= \sum_{\sim\{x_1\}} V_A(x_1) = V_A(x_1) \\ \mu_{B \rightarrow X_2}(x_2) &= \sum_{\sim\{x_2\}} V_B(x_2) = V_B(x_2) \\ \mu_{X_4 \rightarrow D}(x_4) &= 1 \\ \mu_{X_5 \rightarrow E}(x_5) &= 1\end{aligned}$$

# Running the Algorithm



②

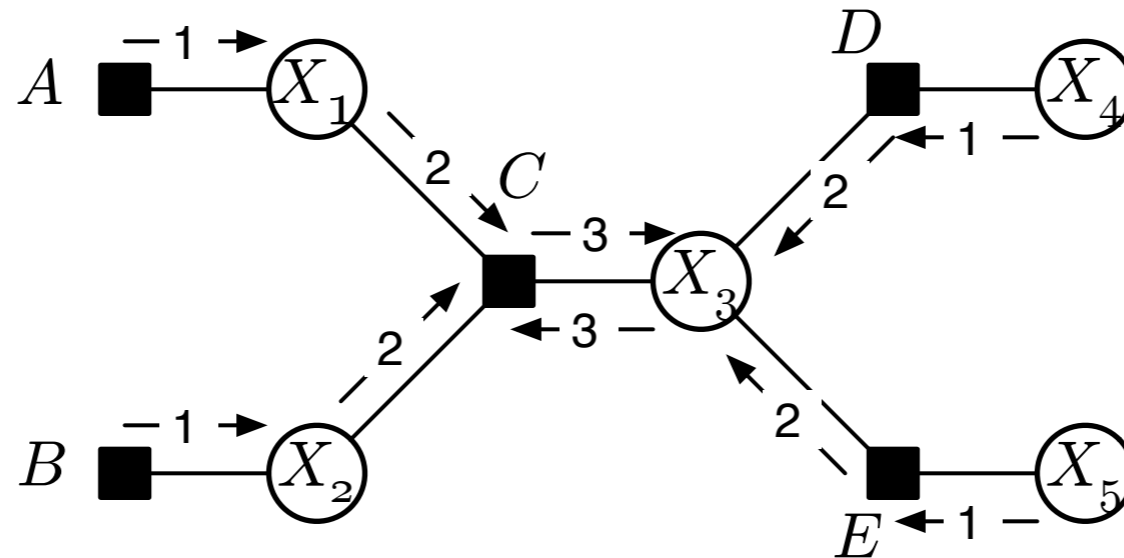
$$\mu_{X_1 \rightarrow C}(x_1) = \mu_{A \rightarrow X_1}(x_1)$$

$$\mu_{X_2 \rightarrow C}(x_2) = \mu_{B \rightarrow X_2}(x_2)$$

$$\mu_{D \rightarrow X_3}(x_3) = \sum_{\sim\{x_3\}} V_D(x_3, x_4) \mu_{X_4 \rightarrow D}(x_4)$$

$$\mu_{E \rightarrow X_3}(x_3) = \sum_{\sim\{x_3\}} V_E(x_3, x_5) \mu_{X_5 \rightarrow D}(x_5)$$

# Running the Algorithm

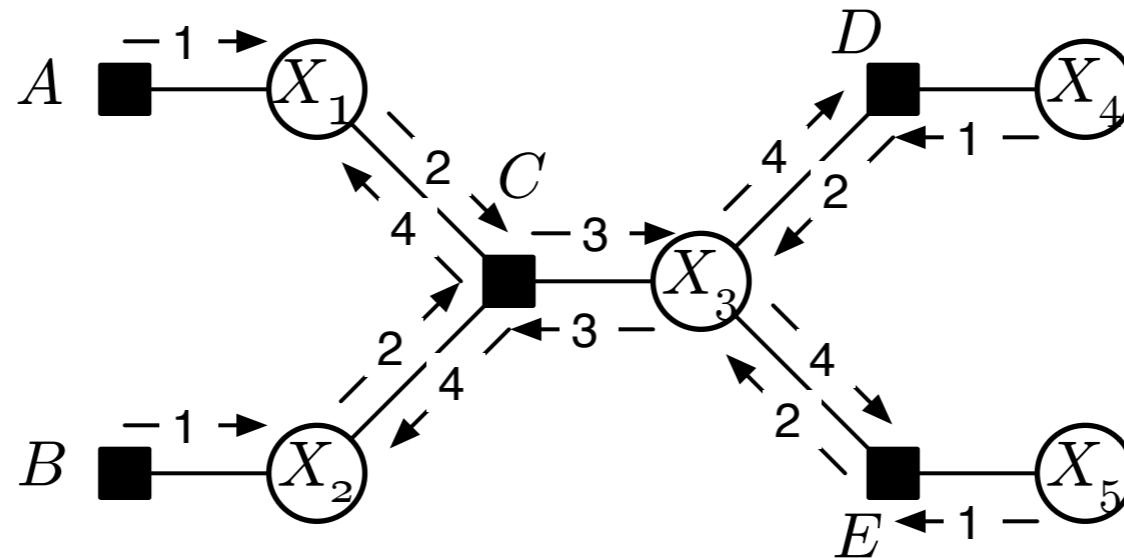


③

$$\mu_{C \rightarrow X_3}(x_3) = \sum_{\sim\{x_3\}} V_C(x_1, x_2, x_3) \mu_{X_1 \rightarrow C}(x_1) \mu_{X_2 \rightarrow C}(x_2)$$

$$\mu_{X_3 \rightarrow C}(x_3) = \mu_{D \rightarrow X_3}(x_3) \mu_{E \rightarrow X_3}(x_3)$$

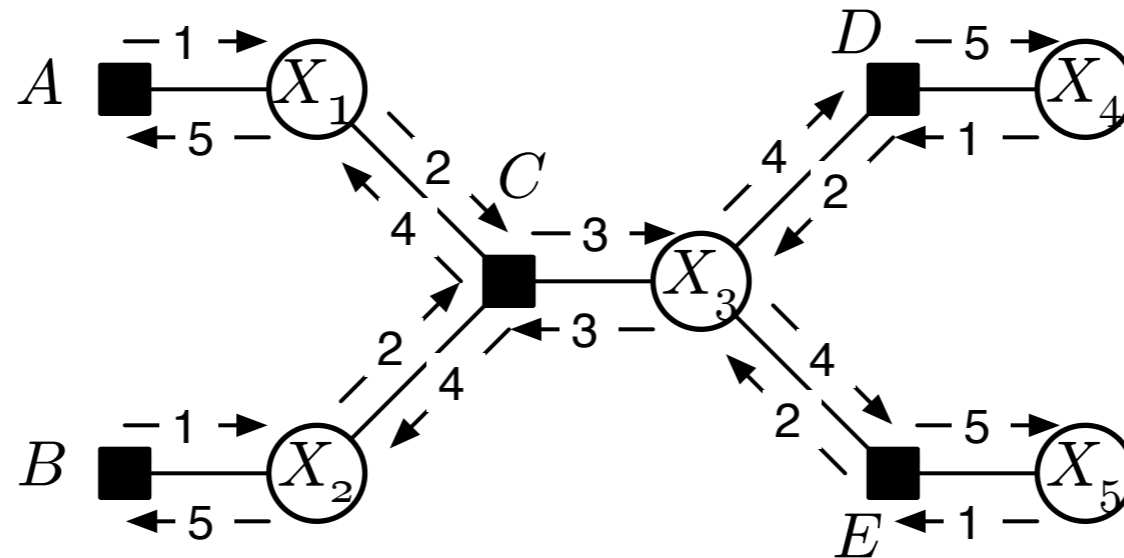
# Running the Algorithm



④

$$\begin{aligned} \mu_{C \rightarrow X_1}(x_1) &= \sum_{\sim\{x_1\}} V_C(x_1, x_2, x_3) \mu_{X_2 \rightarrow C}(x_2) \mu_{X_3 \rightarrow C}(x_3) \\ \mu_{C \rightarrow X_2}(x_2) &= \sum_{\sim\{x_1\}} V_C(x_1, x_2, x_3) \mu_{X_1 \rightarrow C}(x_1) \mu_{X_3 \rightarrow C}(x_3) \\ \mu_{X_3 \rightarrow D}(x_3) &= \mu_{C \rightarrow X_3}(x_3) \mu_{E \rightarrow X_3}(x_3) \\ \mu_{X_3 \rightarrow E}(x_3) &= \mu_{C \rightarrow X_3}(x_3) \mu_{D \rightarrow X_3}(x_3) \end{aligned}$$

# Running the Algorithm



⑤

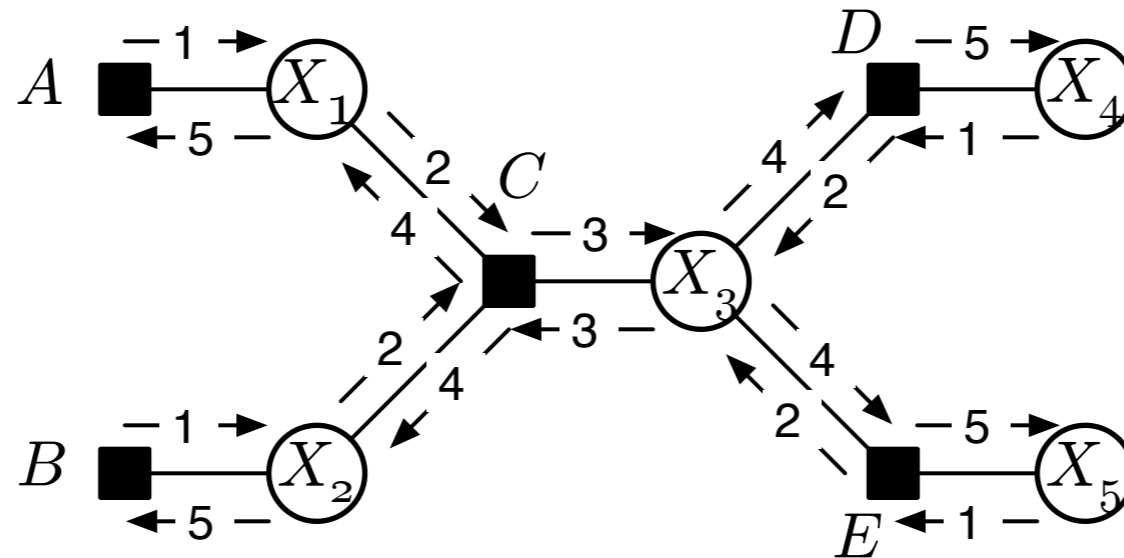
$$\mu_{X_1 \rightarrow A}(x_1) = \mu_{C \rightarrow X_1}(x_1)$$

$$\mu_{X_2 \rightarrow B}(x_2) = \mu_{C \rightarrow X_2}(x_2)$$

$$\mu_{D \rightarrow X_4}(x_4) = \sum_{\sim\{x_4\}} V_D(x_3, x_4) \mu_{X_3 \rightarrow D}(x_3)$$

$$\mu_{E \rightarrow X_5}(x_5) = \sum_{\sim\{x_5\}} V_E(x_3, x_5) \mu_{X_3 \rightarrow E}(x_3)$$

# Extracting Marginals



$$\begin{aligned}
 P(X_1 = x_1) &= \mu_{A \rightarrow X_1}(x_1) \mu_{C \rightarrow X_1}(x_1) \\
 P(X_2 = x_2) &= \mu_{B \rightarrow X_2}(x_2) \mu_{C \rightarrow X_2}(x_2) \\
 P(X_3 = x_3) &= \mu_{C \rightarrow X_3}(x_3) \mu_{D \rightarrow X_3}(x_3) \mu_{E \rightarrow X_3}(x_3) \\
 P(X_4 = x_4) &= \mu_{D \rightarrow X_4}(x_4) \\
 P(X_5 = x_5) &= \mu_{E \rightarrow X_5}(x_5)
 \end{aligned}$$

# Semirings

- We've only used
  - associativity and commutativity of  $+$
  - associativity of  $\times$
  - distributivity of  $\times$  with respect to  $+$
- Message passing and the sum-product algorithm apply to any *semiring*
- In particular, the min-sum semiring
  - Applying to  $-\log$  probabilities, we get the Viterbi algorithm

# Dealing with Cycles

- The sum-product algorithm is only valid for factor trees
- Extending to general factor graphs:
  - *exact inference*: transform factor graph into factor tree
  - *approximate inference*
    - loopy propagation
    - variational methods
    - relaxing marginal constraints

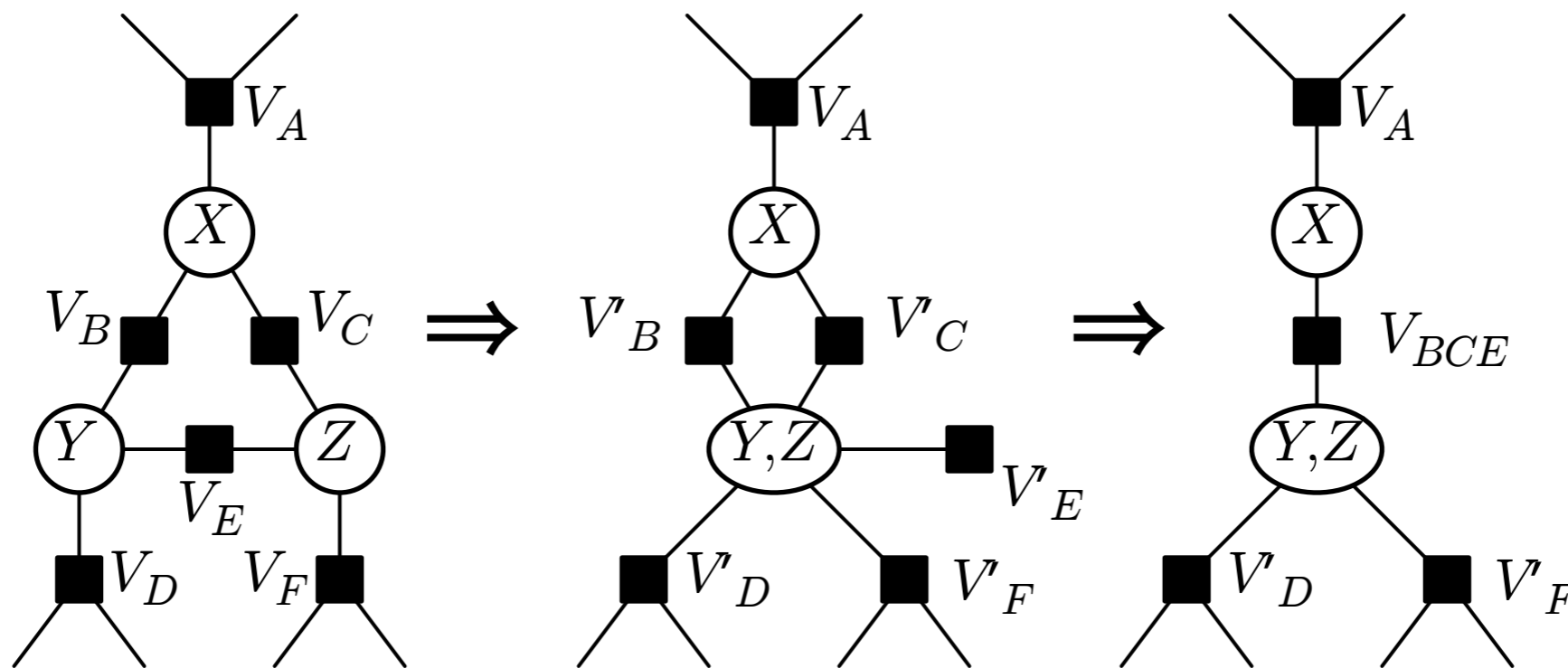
# Exact Inference

- Sum-product algorithm on factor trees is
- $O(k^{m-1}e)$ 
  - $e$ : number of edges
  - $m$ : maximum degree of a factor
  - $k$ : number of possible variable values
- Transformations make a tree from a graph
  - At the expense of making  $k$  larger
  - Finding the optimal transformation is intractable
  - Useful heuristics exist

# Clustering

- Merge variables and factors to remove cycles

$$Y = y, Z = z \equiv W = (y, z)$$



$$V'_B(x, (y, z)) = V_B(x, y)$$

$$V'_C(x, (y, z)) = V_C(x, z)$$

$$V'_D((y, z), \dots) = V_D(y, \dots)$$

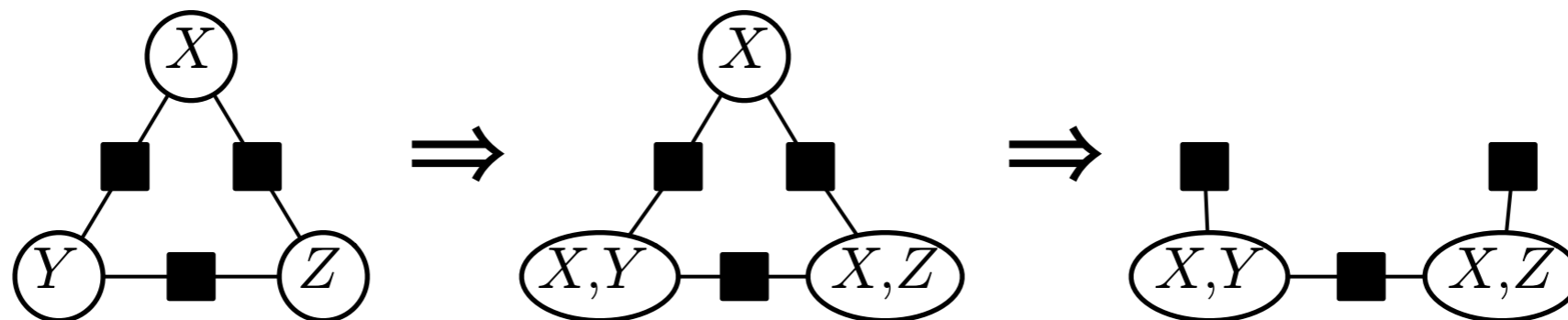
$$V'_E((y, z)) = V_E(y, z)$$

$$V'_F((y, z), \dots) = V_F(z, \dots)$$

$$V_{BCE}(x, w) = V'_B(x, w)V'_C(x, w)V'_E(w)$$

# Stretching

- Extend variables along paths
- Remove redundant factors
- *Spanning trees*
  - stretch variables so that vertices that use the variable form subtrees
  - edges not in tree become redundant



# Loopy Propagation

- Pretend that the graph is a tree and use standard sum-product messages
- Messages: *updatable* values on edges
  - When message on  $e$  to  $v$  is updated, messages leaving  $v$  on other edges are ready to be updated
  - Initialize with dummy 1 messages
- Message-passing schedules: any order that ensures eventual message updates
- Termination: update count, message stabilization