Exact Inference

Michael Gutmann

Probabilistic Modelling and Reasoning (INFR11134)
School of Informatics, University of Edinburgh

Spring Semester 2019

Recap

$$p(\mathbf{x}|\mathbf{y}_o) = \frac{\sum_{\mathbf{z}} p(\mathbf{x},\mathbf{y}_o,\mathbf{z})}{\sum_{\mathbf{x},\mathbf{z}} p(\mathbf{x},\mathbf{y}_o,\mathbf{z})}$$

Assume that $\mathbf{x}, \mathbf{y}, \mathbf{z}$ each are $d = 500$ dimensional, and that each element of the vectors can take $K = 10$ values.

- Issue 1: To specify $p(\mathbf{x}, \mathbf{y}, \mathbf{z})$, we need to specify $K^{3d} - 1 = 10^{1500} - 1$ non-negative numbers, which is impossible.

  Topic 1: Representation What reasonably weak assumptions can we make to efficiently represent $p(\mathbf{x}, \mathbf{y}, \mathbf{z})$?
- Directed and undirected graphical models, factor graphs
- Factorisation and independencies

Recap

$$p(\mathbf{x}|\mathbf{y}_o) = \frac{\sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{y}_o, \mathbf{z})}{\sum_{\mathbf{x},\mathbf{z}} p(\mathbf{x}, \mathbf{y}_o, \mathbf{z})}$$

- Issue 2: The sum in the numerator goes over the order of $K^d = 10^{500}$ non-negative numbers and the sum in the denominator over the order of $K^{2d} = 10^{1000}$, which is impossible to compute.

  Topic 2: Exact inference Can we further exploit the assumptions on $p(\mathbf{x}, \mathbf{y}, \mathbf{z})$ to efficiently compute the posterior probability or derived quantities?
- Note: we do not want to introduce new assumptions but exploit those that we made to deal with issue 1.
- Quantities of interest:
  - $p(\mathbf{x}|\mathbf{y}_o)$ (marginal inference)
  - $\mathrm{argmax}_{\mathbf{x}}\, p(\mathbf{x}|\mathbf{y}_o)$ (inference of most probable states)
  - $\mathbb{E}\left[g(\mathbf{x}) \mid \mathbf{y}_o\right]$ for some function g (posterior expectations)

Assumptions

If not otherwise mentioned, we here assume discrete valued random variables whose joint pmf factorises as

$$p(x_1, \dots, x_d) \propto \prod_{i=1}^{m} \phi_i(\mathcal{X}_i),$$

with $\mathcal{X}_i \subseteq \{x_1, \dots, x_d\}$ and $x_i \in \{1, \dots, K\}$.

Note:

- Includes case where (some of) the ϕ_i are conditionals
- The x_i could be categorical taking on maximally K different values.

Program

1. Marginal inference by variable elimination
2. Marginal inference for factor trees (sum-product algorithm)
3. Inference of most probable states

Program

1. Marginal inference by variable elimination
    - Exploiting the factorisation by using the distributive law $ab + ac = a(b + c)$ and by caching computations
    - Variable elimination for general factor graphs
    - Structural changes to the graph due to variable elimination
    - The principles of variable elimination also apply to continuous random variables
2. Marginal inference for factor trees (sum-product algorithm)
3. Inference of most probable states

Example (full factorisation)

- Consider discrete-valued random variables $x_1, x_2, x_3 \in \{1, \ldots, K\}$
- Assume pmf factorises $p(x_1, x_2, x_3) \propto \phi_1(x_1)\phi_2(x_2)\phi_3(x_3)$
- Task: compute $p(x_1 = k)$ for $k \in \{1, \ldots, K\}$
- We can use the sum-rule

  $$p(x_1 = k) = \sum_{x_2, x_3} p(x_1 = k, x_2, x_3)$$

  Sum over K^2 terms for each k (value of x_1).
- Pre-computing $p(x_1, x_2, x_3)$ for all K^3 configurations and then computing the sum is neither necessary nor a good idea
- Exploit factorisation when computing $p(x_1 = k)$.

Example (full factorisation)

$$
\begin{aligned}
&\text{(sum rule)} && p(x_1 = k) = \sum_{x_2, x_3} p(x_1 = k, x_2, x_3) && (1) \\
&\text{(factorisation)} && \propto \sum_{x_2} \sum_{x_3} \phi_1(k)\phi_2(x_2)\phi_3(x_3) && (2) \\
&\text{(distr. law)} && \propto \phi_1(k) \sum_{x_2} \sum_{x_3} \phi_2(x_2)\phi_3(x_3) && (3) \\
&\text{(distr. law)} && \propto \phi_1(k) \left[\sum_{x_2} \phi_2(x_2)\right] \left[\sum_{x_3} \phi_3(x_3)\right] && (4)
\end{aligned}
$$

Example (full factorisation)

$$p(x_1 = k) \propto \phi_1(k) \left[\sum_{x_2} \phi_2(x_2)\right] \left[\sum_{x_3} \phi_3(x_3)\right] \tag{5}$$

What's the point?

- Because of the factorisation (independencies) we don't need to evaluate and store the values of $p(x_1, x_2, x_3)$ for all K^3 configurations of the random variables.
- 2 sums over K numbers vs. 1 sum over K^2 numbers
- Recycling/caching of already computed quantities: we only need to compute

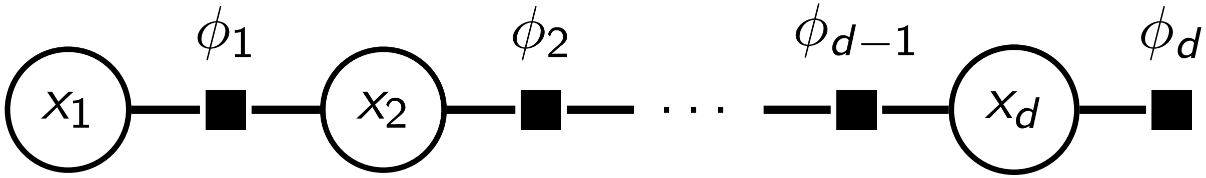
  $$\left[\sum_{x_2} \phi_2(x_2)\right] \left[\sum_{x_3} \phi_3(x_3)\right]$$

  once; the value can be re-used when computing $p(x_1 = k)$ for different k.

Example (chain)

- Assume the pmf factorises as

$$p(x_1, \ldots, x_d) \propto \left[\prod_{i=1}^{d-1} \phi_i(x_i, x_{i+1})\right] \phi_d(x_d)$$



- Task: compute $p(x_1 = k)$ for $k \in \{1, \ldots, K\}$
- Non-scalable approach: Pre-compute $p(x_1, \ldots, x_d)$ for all K^d configurations and then use sum-rule
- Smarter: Exploit factorisation when applying the sum rule

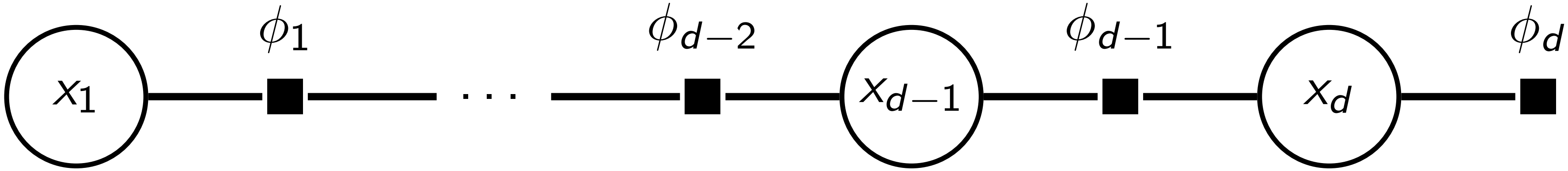
Example (chain)

We have to sum over $x_2, \ldots, x_d$. Let's do x_d first

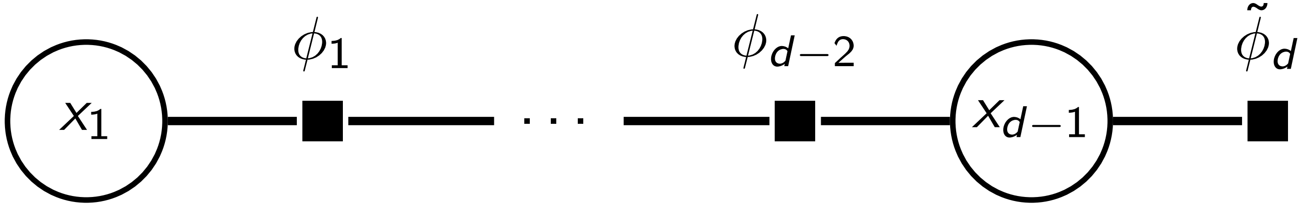
$$
\begin{aligned}
p(x_1, \ldots, x_{d-1}) &= \sum_{x_d} p(x_1, \ldots, x_d) && (6)\\
\text{(factorisation)} &\propto \sum_{x_d} \left[\prod_{i=1}^{d-1} \phi_i(x_i, x_{i+1})\right] \phi_d(x_d) && (7)\\
&\propto \sum_{x_d} \left[\prod_{i=1}^{d-2} \phi_i(x_i, x_{i+1})\right] \phi_{d-1}(x_{d-1}, x_d)\phi_d(x_d) && (8)\\
\text{(by distr. law)} &\propto \left[\prod_{i=1}^{d-2} \phi_i(x_i, x_{i+1})\right] \underbrace{\sum_{x_d} \phi_{d-1}(x_{d-1}, x_d)\phi_d(x_d)}_{\tilde{\phi}_d(x_{d-1}) \quad \text{total cost: } K \cdot K = K^2} && (9)\\
&\propto \left[\prod_{i=1}^{d-2} \phi_i(x_i, x_{i+1})\right] \tilde{\phi}_d(x_{d-1}) && (10)
\end{aligned}
$$

Example (chain)

Factor graph for $p(x_1, \ldots, x_d) \propto \left[\prod_{i=1}^{d-1} \phi_i(x_i, x_{i+1})\right] \phi_d(x_d)$



Factor graph for $p(x_1, \ldots, x_{d-1}) \propto \left[\prod_{i=1}^{d-2} \phi_i(x_i, x_{i+1})\right] \tilde{\phi}_d(x_{d-1})$



Example (chain)

Next, sum over x_{d-1}

$$p(x_1, \ldots, x_{d-2}) = \sum_{x_{d-1}} p(x_1, \ldots, x_{d-1}) \tag{11}$$

$$\text{(factorisation)} \propto \sum_{x_{d-1}} \left[\prod_{i=1}^{d-2} \phi_i(x_i, x_{i+1})\right] \tilde{\phi}_d(x_{d-1}) \tag{12}$$

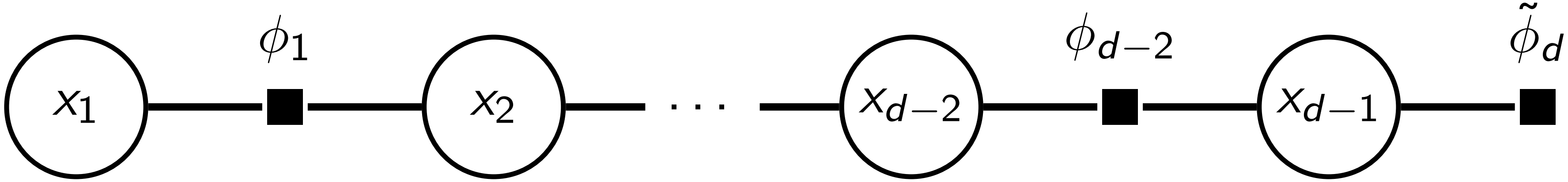
$$\propto \sum_{x_{d-1}} \left[\prod_{i=1}^{d-3} \phi_i(x_i, x_{i+1})\right] \phi_{d-2}(x_{d-2}, x_{d-1})\tilde{\phi}_d(x_{d-1})$$

$$\text{(by distr. law)} \propto \left[\prod_{i=1}^{d-3} \phi_i(x_i, x_{i+1})\right] \underbrace{\sum_{x_{d-1}} \phi_{d-2}(x_{d-2}, x_{d-1})\tilde{\phi}_d(x_{d-1})}_{\tilde{\phi}_{d,d-1}(x_{d-2}) \quad \text{total cost: } K \cdot K = K^2}$$

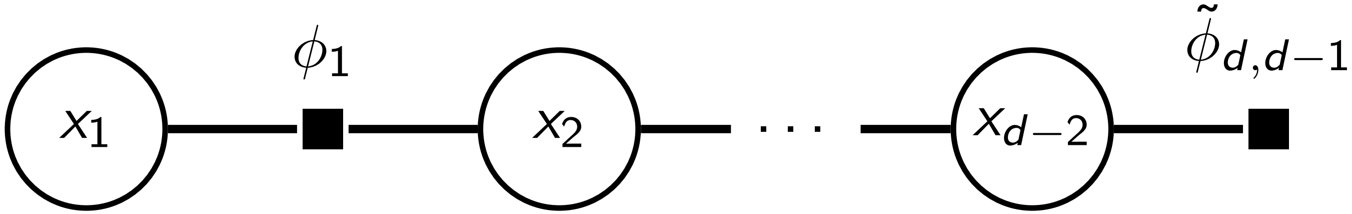
$$\propto \left[\prod_{i=1}^{d-3} \phi_i(x_i, x_{i+1})\right] \tilde{\phi}_{d,d-1}(x_{d-2}) \tag{13}$$

Example (chain)

Factor graph for $p(x_1, \ldots, x_{d-1}) \propto \left[\prod_{i=1}^{d-2} \phi_i(x_i, x_{i+1})\right] \tilde{\phi}_d(x_{d-1})$



Factor graph for
$p(x_1, \ldots, x_{d-2}) \propto \left[\prod_{i=1}^{d-3} \phi_i(x_i, x_{i+1})\right] \tilde{\phi}_{d,d-1}(x_{d-2})$



Example (chain)

- Continue eliminating the last (leaf) variable
- Each time we eliminate a variable, we need to
    - compute $\phi_i(x_i, x_{i+1})$ for all values of x_i and x_{i+1} (matrix with K^2 numbers)
    - sum over K numbers to compute the $\tilde{\phi}(x_i)$ for all K values of x_i (cost: $O(K^2)$)
- To compute $p(x_1 = k)$ we have to eliminate $d - 1$ variables

$\Rightarrow$ Total cost for $p(x_1)$:$O((d-1)K^2) = O(dK^2)$

Example (chain)

- Benefits of exploiting the factorisation
  - Linear growth in number of variables d: in contrast to exponential growth $O(K^d)$ when factorisation is not exploited
  - Recycling/caching: most terms do not depend on x_1 and can be re-used when we compute $p(x_1 = k)$ for different k (e.g. $\tilde{\phi}_d$, $\tilde{\phi}_{d,d-1}$ etc.)
- Chains have the special property that they stay a chain after a leaf variable is eliminated.
- More general factor trees have the same property, which we exploit in the sum-product algorithm.
- First: variable elimination for general factor graphs.

Basic ideas of variable elimination

1. Use the distributive law $ab + ac = a(b + c)$ to exploit the factorisation ($\sum\prod \rightarrow \prod\sum$):
   reduces the overall dimensionality of the domain of the factors in the sum and thereby the computational cost.
2. Recycle/cache results

Variable (bucket) elimination

Example task: Given $p(x_1, \ldots, x_d) \propto \prod_i^m \phi_i(\mathcal{X}_i)$ compute the marginal $p(\mathcal{X}_{\text{target}})$ for some $\mathcal{X}_{\text{target}} \subseteq \{x_1, \ldots, x_d\}$.

- Assume that at iteration k, you have the pmf over $d^k = d - k$ variables $X^k = (x_{i_1}, \ldots, x_{i_{d^k}})$ that factorises as

$$p(X^k) \propto \prod_{i=1}^{m^k} \phi_i^k(\mathcal{X}_i^k)$$

- Decide which variable to eliminate. Call it x^*. ($x^* \in X^k$, $x^* \notin \mathcal{X}_{\text{target}}$)
- Let X^{k+1} be equal to X^k with x^* removed. We have

$$\text{(sum rule)} \qquad p(X^{k+1}) = \sum_{x^*} p(X^k) \tag{14}$$

$$\text{(factorisation)} \qquad \propto \sum_{x^*} \prod_{i=1}^{m^k} \phi_i^k(\mathcal{X}_i^k) \tag{15}$$

Variable elimination (cont.)

$$p(X^{k+1}) \propto \sum_{x^*} \prod_{i:x^* \notin \mathcal{X}_i^k} \phi_i^k(\mathcal{X}_i^k) \prod_{i:x^* \in \mathcal{X}_i^k} \phi_i^k(\mathcal{X}_i^k) \tag{16}$$

$$\text{(distr. law)} \propto \prod_{i:x^* \notin \mathcal{X}_i^k} \phi_i^k(\mathcal{X}_i^k) \underbrace{\sum_{x^*} \prod_{i:x^* \in \mathcal{X}_i^k} \phi_i^k(\mathcal{X}_i^k)}_{\text{new factor } \tilde{\phi}_*} \tag{17}$$

$$\propto \left[\prod_{i:x^* \notin \mathcal{X}_i^k} \phi_i^k(\mathcal{X}_i^k) \right] \tilde{\phi}_*(\tilde{\mathcal{X}}_*) \tag{18}$$

where $\tilde{\mathcal{X}}_*$ is the union of all $\mathcal{X}_i^k$ that contained x^*, with x^* removed

$$\tilde{\mathcal{X}}_* = \bigcup_{i:x^* \in \mathcal{X}_i^k} \left(\mathcal{X}_i^k \setminus x^* \right) \tag{19}$$

Variable elimination (cont.)

- By re-labelling the factors and variables, we obtain

$$p(X^{k+1}) \propto \left[\prod_{i: x^* \notin \mathcal{X}_i^k} \phi_i^k(\mathcal{X}_i^k) \right] \tilde{\phi}_*(\tilde{\mathcal{X}}_*) \tag{20}$$

$$\propto \prod_{i=1}^{m^{k+1}} \phi_i^{k+1}(\mathcal{X}_i^{k+1}), \tag{21}$$

  which has the same form as $p(X^k)$.
- Set $k = k + 1$ and decide which variable x^* to eliminate next.
- To compute $p(\mathcal{X}_{\text{target}})$ stop when $X^k = \mathcal{X}_{\text{target}}$, followed by normalisation.

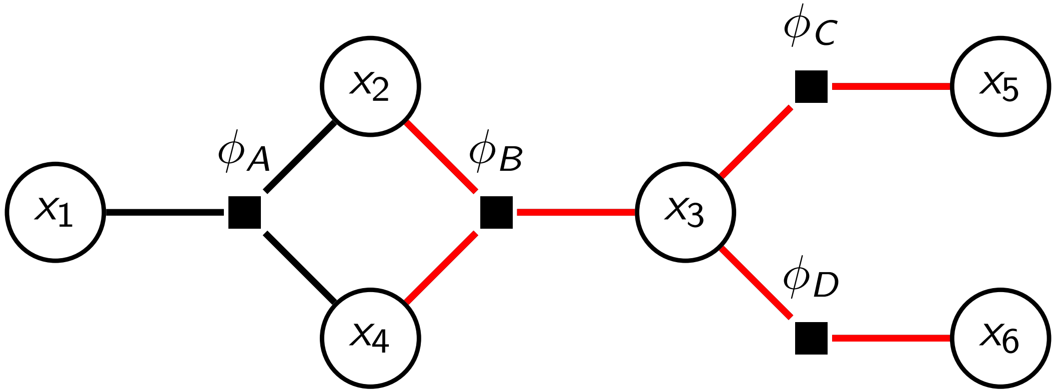
How to choose the elimination variable x^*?

- When we marginalise over x^*, we generate a new factor $\tilde{\phi}_*$ that depends on

$$\tilde{\mathcal{X}}_* = \bigcup_{i: x^* \in \mathcal{X}_i^k} \left(\mathcal{X}_i^k \setminus x^* \right) \tag{22}$$

  This is the set of variables with which x^* shares a factor node in the factor graph ("neighbours").

- Ex.: $p(x_1, \ldots, x_6) \propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\phi_C(x_3, x_5)\phi_D(x_3, x_6)$
  If we eliminated $x^* = x_3$: $\tilde{\mathcal{X}}_* = \{x_2, x_4, x_5, x_6\}$



How to choose the elimination variable x^*?

- When we marginalise over x^*, we generate a new factor $\tilde{\phi}_*$ that depends on

  $$\tilde{\mathcal{X}}_* = \bigcup_{i:x^* \in \mathcal{X}_i^k} \left(\mathcal{X}_i^k \setminus x^*\right) \tag{23}$$

  This is the set of variables with which x^* shares a factor node in the factor graph (“neighbours”).
- If $\tilde{\mathcal{X}}_*$ contains many variables, variable elimination becomes expensive in later iterations (exponential in size of largest $\mathcal{X}^k$).
- Optimal choice of x^* is difficult (for details, see e.g. Koller, Section 9.4, not examinable)
- Heuristic: choose x^* in a greedy way, e.g. the variable with the least number of neighbours in the factor graph (e.g. x_5 or x_6 in the example)

Computing conditionals

- The same approach can be used to compute conditionals.
- Example: Given

  $$p(x_1, \ldots, x_6) \propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\phi_C(x_3, x_5)\phi_D(x_3, x_6)$$

  assume you want to compute $p(x_1|x_3 = \alpha)$
- We can write

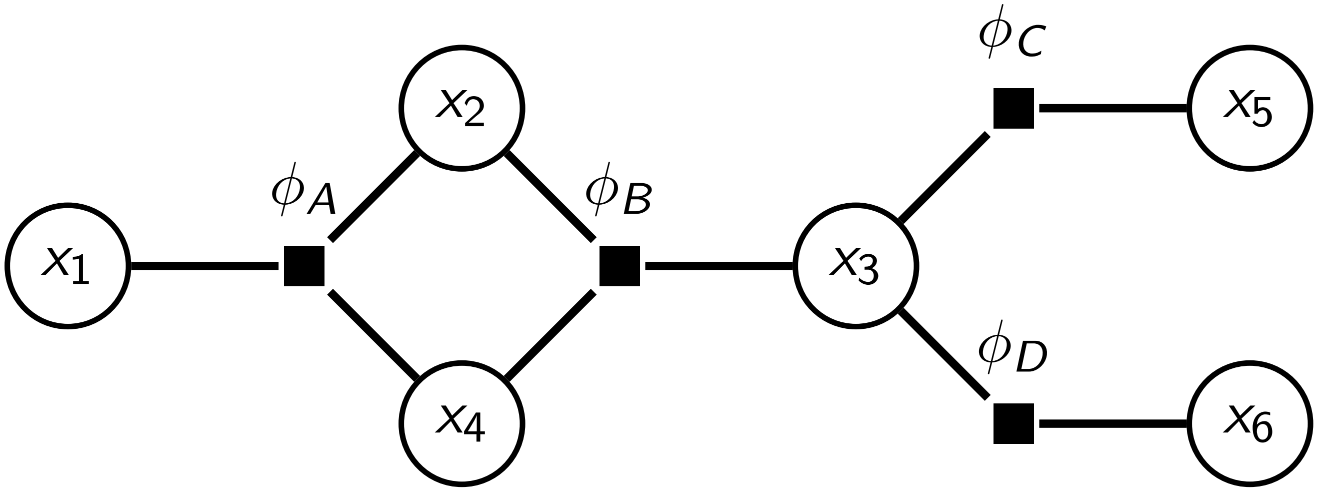
  $$\begin{aligned} p(x_1, x_2, x_4, x_5, x_6|x_3 = \alpha) &\propto p(x_1, x_2, x_3 = \alpha, x_4, x_5, x_6) \\ &\propto \phi_A(x_1, x_2, x_4)\phi_B^\alpha(x_2, x_4)\phi_C^\alpha(x_5)\phi_D^\alpha(x_6) \end{aligned}$$

  and consider $p(x_1, x_2, x_4, x_5, x_6|x_3 = \alpha)$ to be a pdf/pmf $\tilde{p}(x_1, x_2, x_4, x_5, x_6)$ defined up to the proportionality factor.
- We can compute $p(x_1|x_3 = \alpha) = \tilde{p}(x_1)$ by applying variable elimination to $\tilde{p}(x_1, x_2, x_4, x_5, x_6)$.

Example

- Example:

$$p(x_1, \ldots, x_6) \propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\phi_C(x_3, x_5)\phi_D(x_3, x_6)$$



- Task: Compute $p(x_1, x_3)$
- Note the structural changes in the graph during variable elimination

Example (cont)

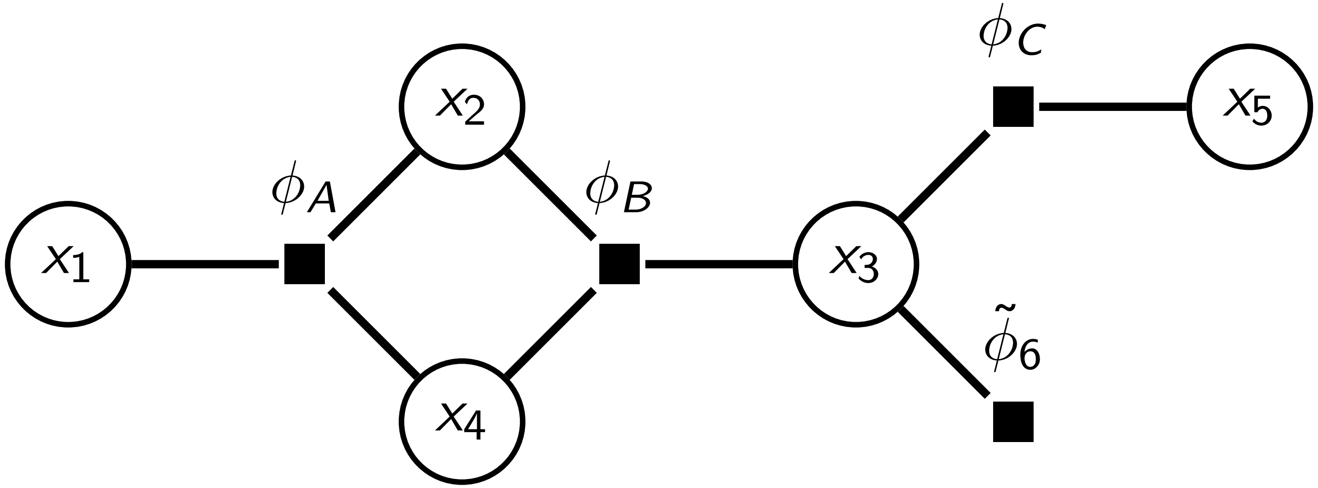
Task: Compute $p(x_1, x_3)$

First eliminate x_6

$$p(x_1, \ldots, x_5) \propto \sum_{x_6} \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\phi_C(x_3, x_5)\phi_D(x_3, x_6)$$

$$\propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\phi_C(x_3, x_5) \sum_{x_6} \phi_D(x_3, x_6)$$

$$\propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\phi_C(x_3, x_5)\tilde{\phi}_6(x_3)$$



Example (cont)

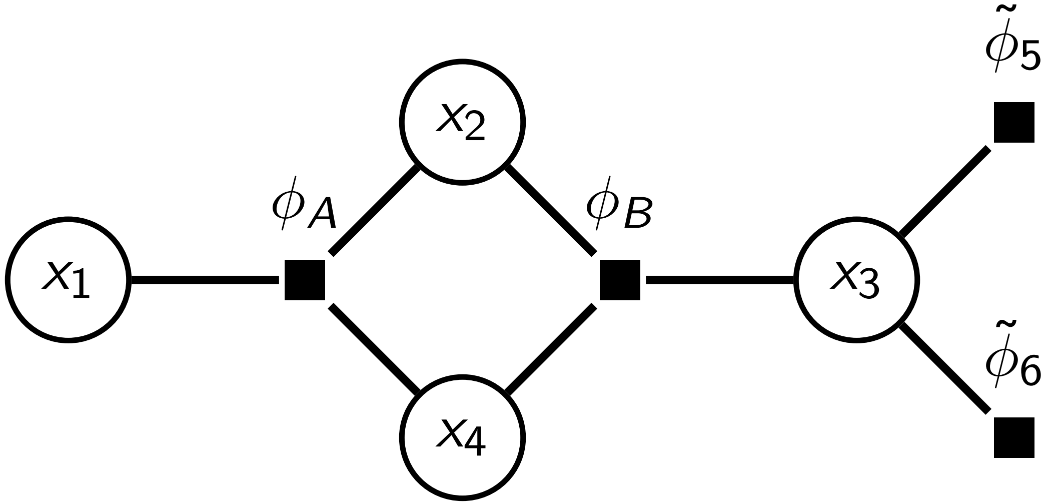
Task: Compute $p(x_1, x_3)$

Eliminate x_5

$$p(x_1, \dots, x_4) \propto \sum_{x_5} \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\phi_C(x_3, x_5)\tilde{\phi}_6(x_3)$$

$$\propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\tilde{\phi}_6(x_3) \sum_{x_5} \phi_C(x_3, x_5)$$

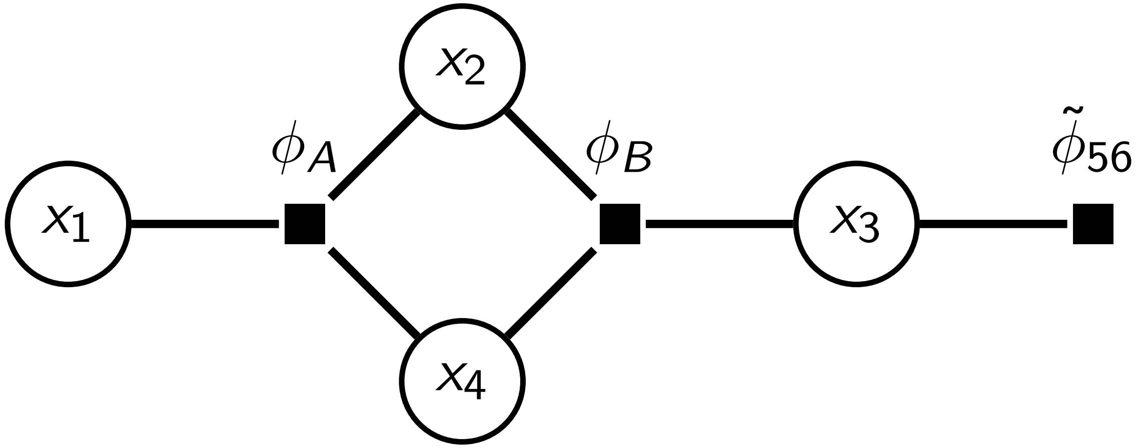
$$\propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\tilde{\phi}_6(x_3)\tilde{\phi}_5(x_3)$$



Example (cont)

Define $\tilde{\phi}_{56}(x_3) = \tilde{\phi}_6(x_3)\tilde{\phi}_5(x_3)$

$$\begin{aligned} p(x_1, \ldots, x_4) &\propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\tilde{\phi}_6(x_3)\tilde{\phi}_5(x_3) \\ &\propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\tilde{\phi}_{56}(x_3) \end{aligned}$$

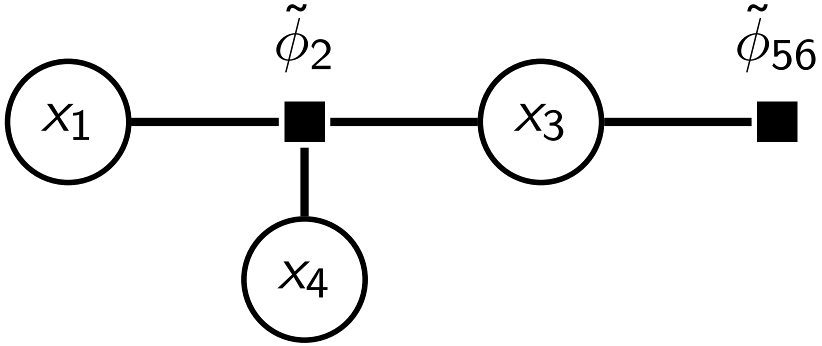


Example (cont)

Task: Compute $p(x_1, x_3)$

Eliminate x_2

$$\begin{aligned} p(x_1, x_3, x_4) &\propto \sum_{x_2} \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\tilde{\phi}_{56}(x_3) \\ &\propto \tilde{\phi}_{56}(x_3) \sum_{x_2} \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4) \\ &\propto \tilde{\phi}_{56}(x_3)\tilde{\phi}_2(x_1, x_3, x_4) \end{aligned}$$

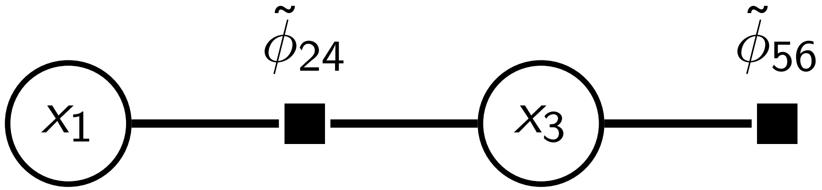


Example (cont)

Task: Compute $p(x_1, x_3)$

Eliminate x_4

$$\begin{aligned} p(x_1, x_3) &\propto \sum_{x_4} \tilde{\phi}_{56}(x_3)\tilde{\phi}_2(x_1, x_3, x_4) \\ &\propto \tilde{\phi}_{56}(x_3) \sum_{x_4} \tilde{\phi}_2(x_1, x_3, x_4) \\ &\propto \tilde{\phi}_{56}(x_3)\tilde{\phi}_{24}(x_1, x_3) \end{aligned}$$



Normalisation:

$$p(x_1, x_3) = \frac{\tilde{\phi}_{56}(x_3)\tilde{\phi}_{24}(x_1, x_3)}{\sum_{x_1, x_3} \tilde{\phi}_{56}(x_3)\tilde{\phi}_{24}(x_1, x_3)}$$

Structural changes in the graph during variable elimination

- Eliminated leaf-variable and factor node
  → factor node
- Factors node depending on the same variables
  → single factor node
- Factor nodes between neighbours of the target variable
  → single factor node connecting all neighbours

What if we have continuous random variables?

- Conceptually, all stays the same but we replace sums with integrals
  - Simplifications due to distributive law remain valid
  - Caching of results remains valid
- In special cases, integral can be computed in closed form (e.g. Gaussian family)
- If not: need for approximations (see later)
- Approximations are also needed for discrete random variables with high-dimensional range (if K is large).

Program

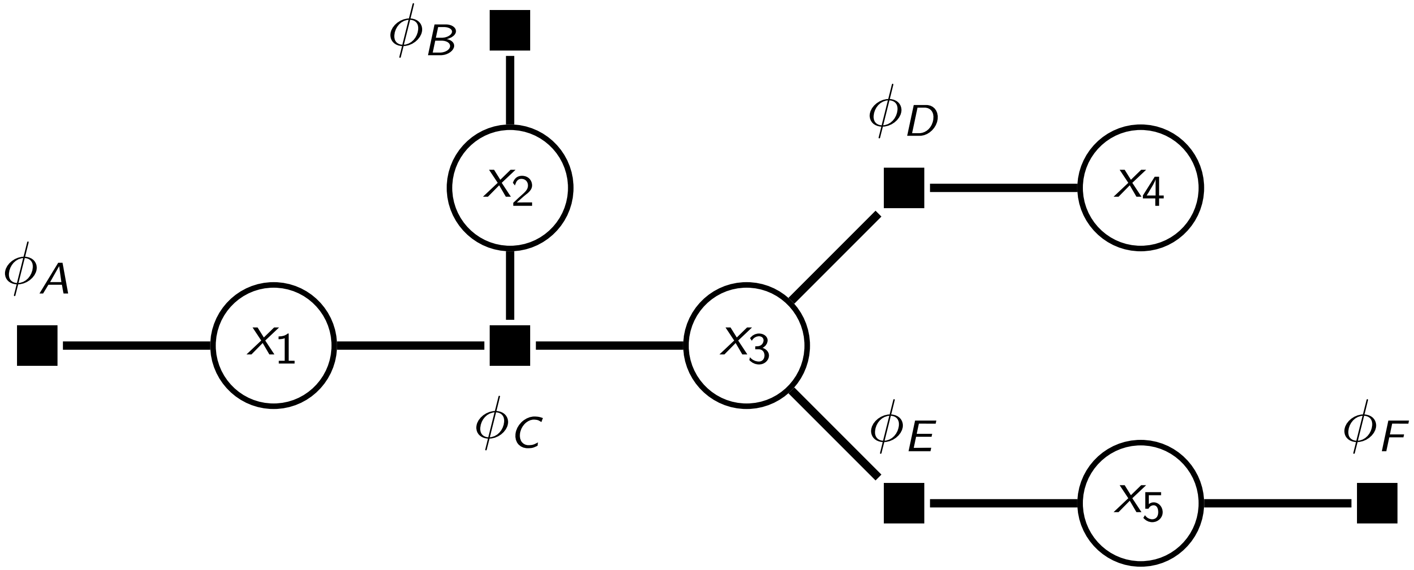
1. Marginal inference by variable elimination
   - Exploiting the factorisation by using the distributive law $ab + ac = a(b + c)$ and by caching computations
   - Variable elimination for general factor graphs
   - Structural changes to the graph due to variable elimination
   - The principles of variable elimination also apply to continuous random variables
2. Marginal inference for factor trees (sum-product algorithm)
3. Inference of most probable states

Program

1. Marginal inference by variable elimination
2. Marginal inference for factor trees (sum-product algorithm)
    - Factor trees
    - Sum-product algorithm = variable elimination for factor trees
    - Messages = effective factors
    - The rules for sum-product message passing
3. Inference of most probable states

Factor trees

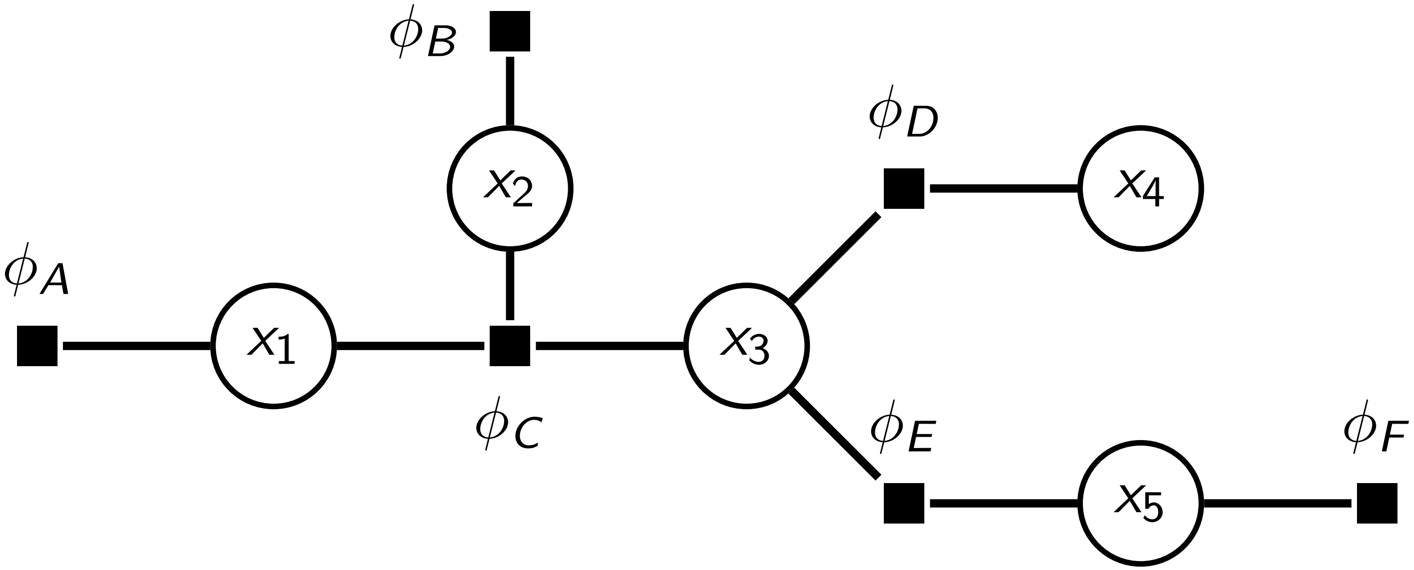
- We next consider the class of models (pmfs/pdfs) for which the factor graph is a tree
- Tree: graph where there is only one path connecting any two nodes (no loops!)
- Chain is an example of a factor tree.
- Useful property: the factor tree obtained after summing out a leaf variable is still a factor tree.



Variable elimination for factor trees

Task: Compute $p(x_1)$ for

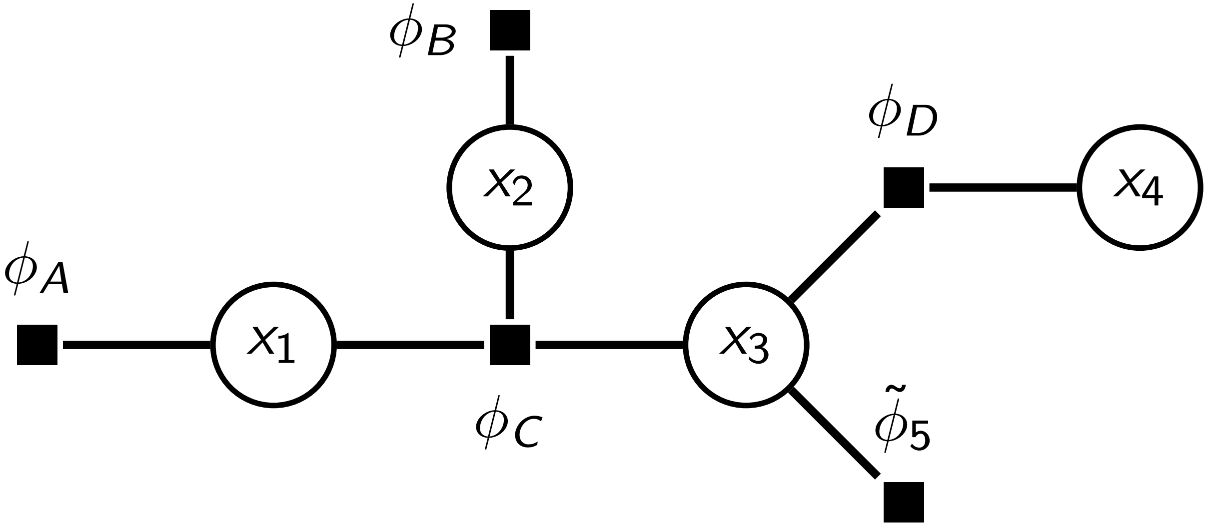
$$p(x_1, \ldots, x_5) \propto \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\phi_D(x_3, x_4)\phi_E(x_3, x_5)\phi_F(x_5)$$



Sum out leaf-variable x_5

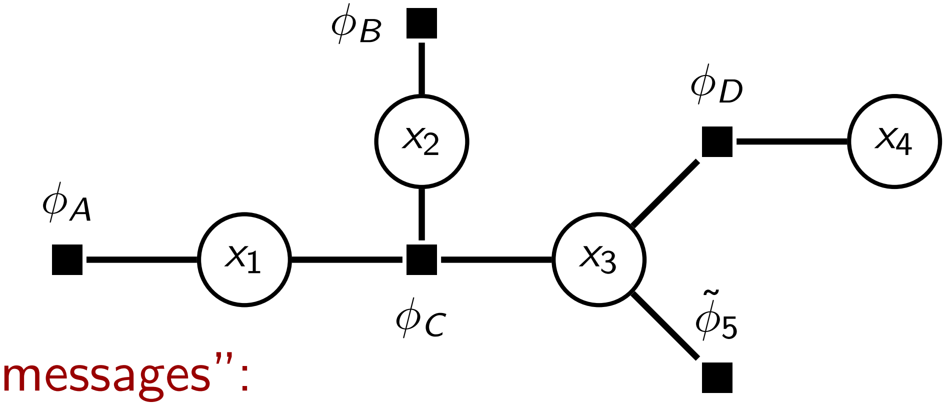
Task: Compute $p(x_1)$

$$\begin{aligned}
p(x_1, \ldots, x_4) &= \sum_{x_5} p(x_1, \ldots, x_5) \\
&\propto \sum_{x_5} \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\phi_D(x_3, x_4)\phi_E(x_3, x_5)\phi_F(x_5) \\
&\propto \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\phi_D(x_3, x_4) \sum_{x_5} \phi_E(x_3, x_5)\phi_F(x_5) \\
&\propto \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\phi_D(x_3, x_4)\tilde{\phi}_5(x_3)
\end{aligned}$$

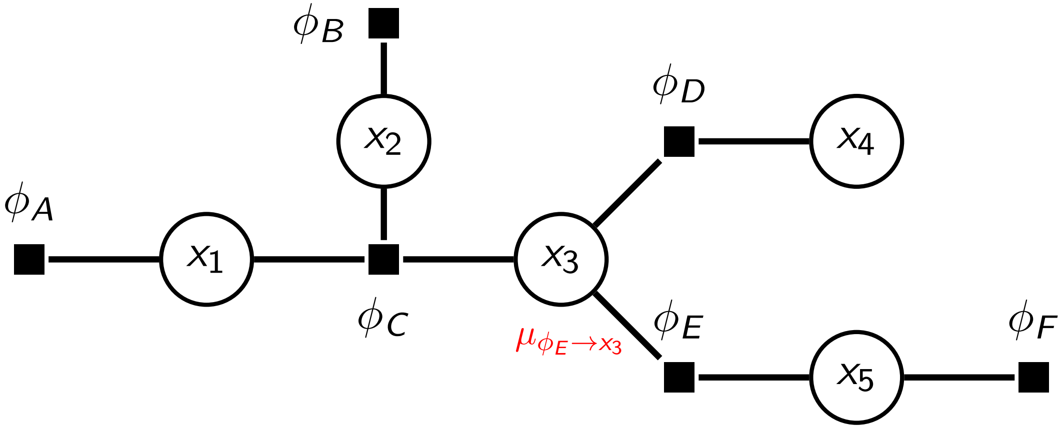


Visualising the computation

Graph with transformed factors:



Graph with "messages":



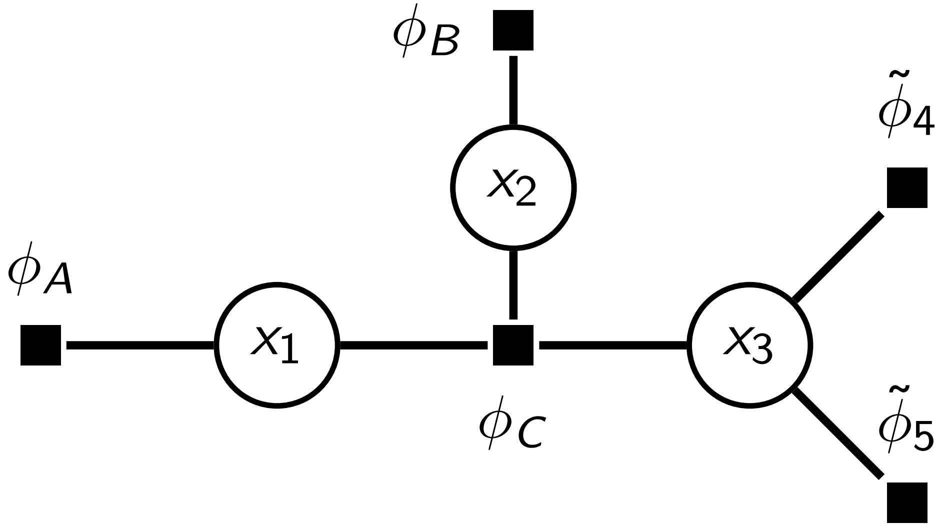
Message: $\mu_{\phi_E \to x_3}(x_3) = \tilde{\phi}_5(x_3) = \sum_{x_5} \phi_E(x_3, x_5)\phi_F(x_5)$

Effective factor for x_3 if all variables in the subtree attached to ϕ_E are eliminated (subtree does *not* include x_3)

Sum out leaf-variable x_4

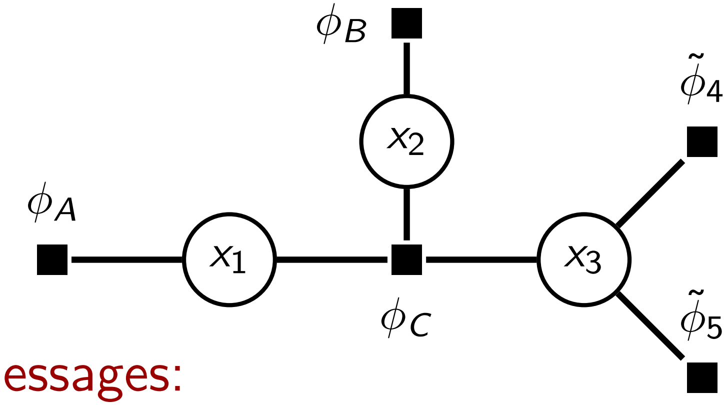
Task: Compute $p(x_1)$

$$\begin{aligned}
p(x_1, \ldots, x_3) &= \sum_{x_4} p(x_1, \ldots, x_4) \\
&\propto \sum_{x_4} \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\phi_D(x_3, x_4)\tilde{\phi}_5(x_3) \\
&\propto \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\tilde{\phi}_5(x_3) \sum_{x_4} \phi_D(x_3, x_4) \\
&\propto \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\tilde{\phi}_5(x_3)\tilde{\phi}_4(x_3)
\end{aligned}$$

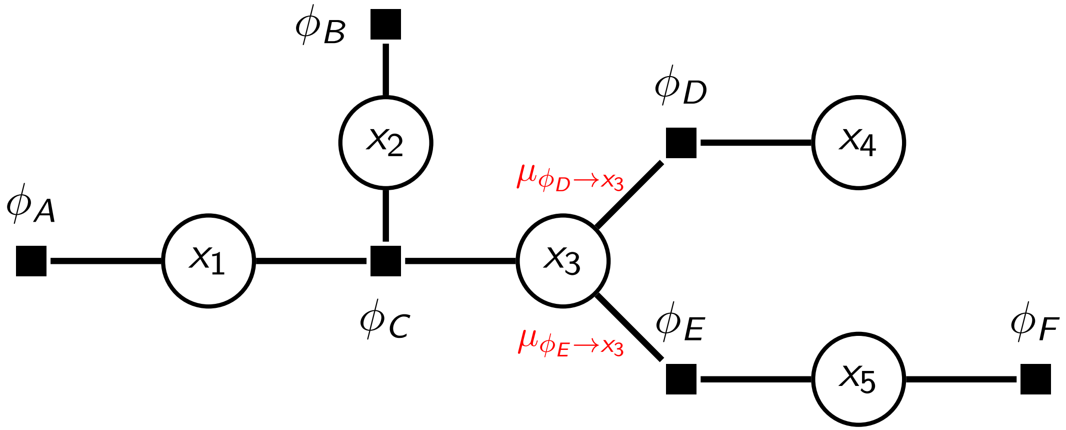


Visualising the computation

Graph with transformed factors:



Graph with messages:



Message: $\mu_{\phi_D \to x_3}(x_3) = \tilde{\phi}_4(x_3) = \sum_{x_4} \phi_D(x_3, x_4)$

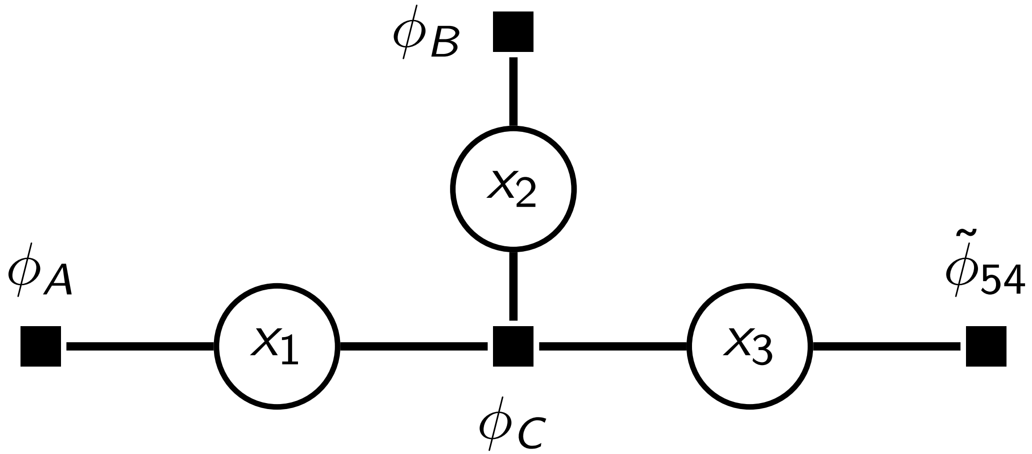
Effective factor for x_3 if all variables in the subtree attached to ϕ_D are eliminated (subtree does *not* include x_3)

Simplify by multiplying factors with common domain

Task: Compute $p(x_1)$

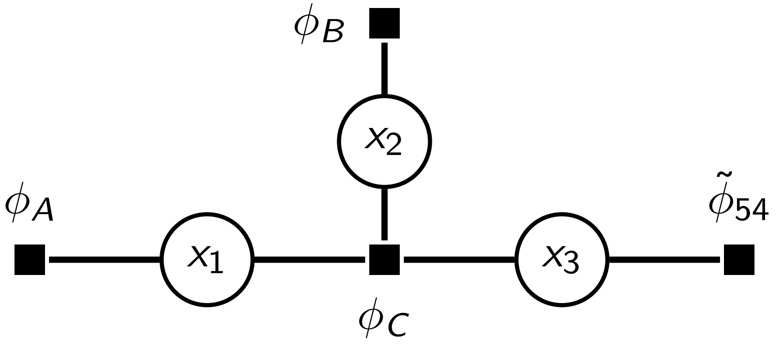
$$p(x_1, \ldots, x_3) \propto \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\underbrace{\tilde{\phi}_5(x_3)\tilde{\phi}_4(x_3)}_{\tilde{\phi}_{54}(x_3)}$$

$$\propto \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\tilde{\phi}_{54}(x_3)$$

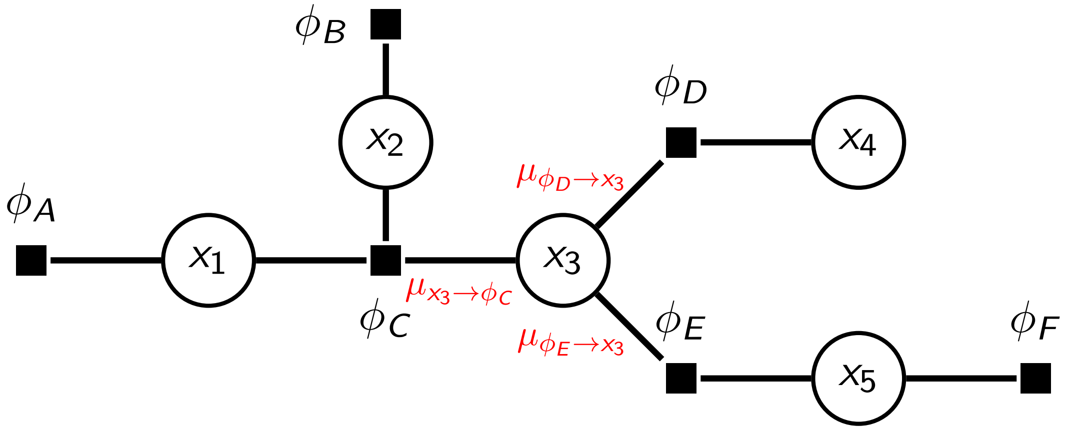


Visualising the computation

Graph with transformed factors:



Graph with messages:



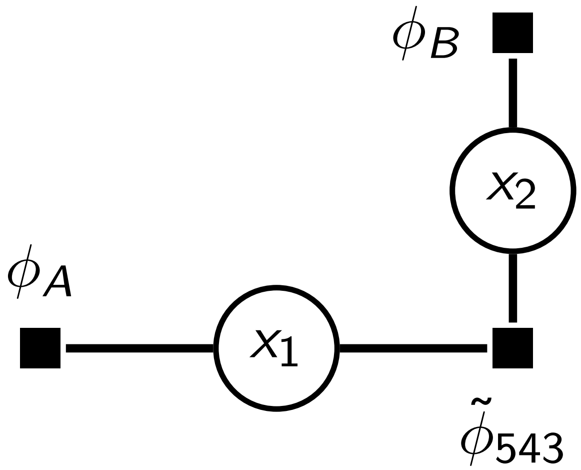
Message: $\mu_{x_3 \to \phi_C}(x_3) = \tilde{\phi}_{54}(x_3) = \tilde{\phi}_4(x_3)\tilde{\phi}_5(x_3) = \mu_{\phi_D \to x_3}(x_3)\mu_{\phi_E \to x_3}(x_3)$

Effective factor for x_3 if all variables in the subtrees attached to x_3 are eliminated (subtrees do *not* include ϕ_c)

Sum out leaf-variable x_3

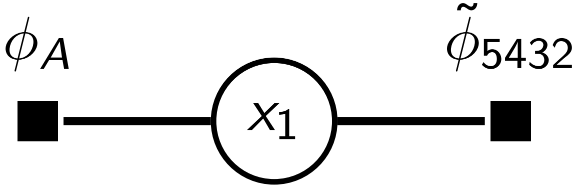
Task: Compute $p(x_1)$

$$\begin{aligned}
p(x_1, x_2) &= \sum_{x_3} p(x_1, x_2, x_3) \\
&\propto \sum_{x_3} \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\tilde{\phi}_{54}(x_3) \\
&\propto \phi_A(x_1)\phi_B(x_2) \sum_{x_3} \phi_C(x_1, x_2, x_3)\tilde{\phi}_{54}(x_3) \\
&\propto \phi_A(x_1)\phi_B(x_2)\tilde{\phi}_{543}(x_1, x_2)
\end{aligned}$$



Sum out leaf-variable x_2 and normalise

$$
\begin{aligned}
p(x_1) = \sum_{x_2} p(x_1, x_2) &\propto \sum_{x_2} \phi_A(x_1)\phi_B(x_2)\tilde{\phi}_{543}(x_1, x_2) \\
&\propto \phi_A(x_1) \sum_{x_2} \phi_B(x_2)\tilde{\phi}_{543}(x_1, x_2) \\
&\propto \phi_A(x_1)\tilde{\phi}_{5432}(x_1)
\end{aligned}
$$



$$
p(x_1) = \frac{\phi_A(x_1)\tilde{\phi}_{5432}(x_1)}{\sum_{x_1} \phi_A(x_1)\tilde{\phi}_{5432}(x_1)}
$$

Alternative: sum out both x_2 and x_3

Since

$$\begin{aligned}\tilde{\phi}_{5432}(x_1) &= \sum_{x_2} \phi_B(x_2)\tilde{\phi}_{543}(x_1, x_2) \\ &= \sum_{x_2} \phi_B(x_2) \sum_{x_3} \phi_C(x_1, x_2, x_3)\tilde{\phi}_{54}(x_3) \\ &= \sum_{x_2, x_3} \phi_C(x_1, x_2, x_3)\phi_B(x_2)\tilde{\phi}_{54}(x_3)\end{aligned}$$

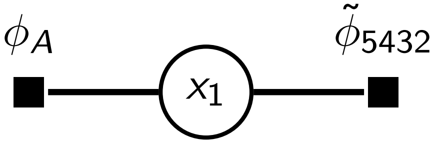
we obtain the same result by first summing out x_2 and then x_3, or both at the same time.

In any case:

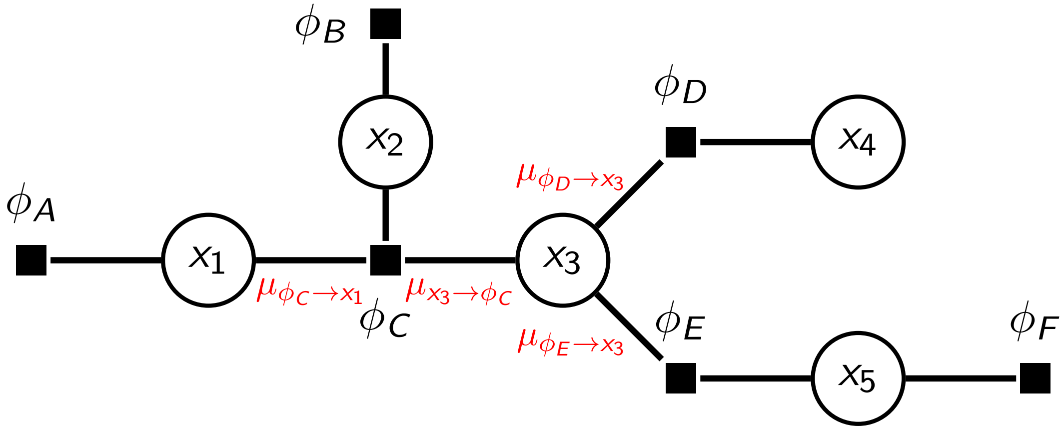
$$p(x_1) \propto \phi_A(x_1) \sum_{x_2, x_3} \phi_C(x_1, x_2, x_3)\phi_B(x_2)\tilde{\phi}_{54}(x_3)$$

Visualising the computation

Graph with transformed factors:



Graph with messages:



Message:

$$\mu_{\phi_C \to x_1}(x_1) = \tilde{\phi}_{5432}(x_1) = \sum_{x_2, x_3} \phi_C(x_1, x_2, x_3)\phi_B(x_2)\mu_{x_3 \to \phi_C}(x_3)$$

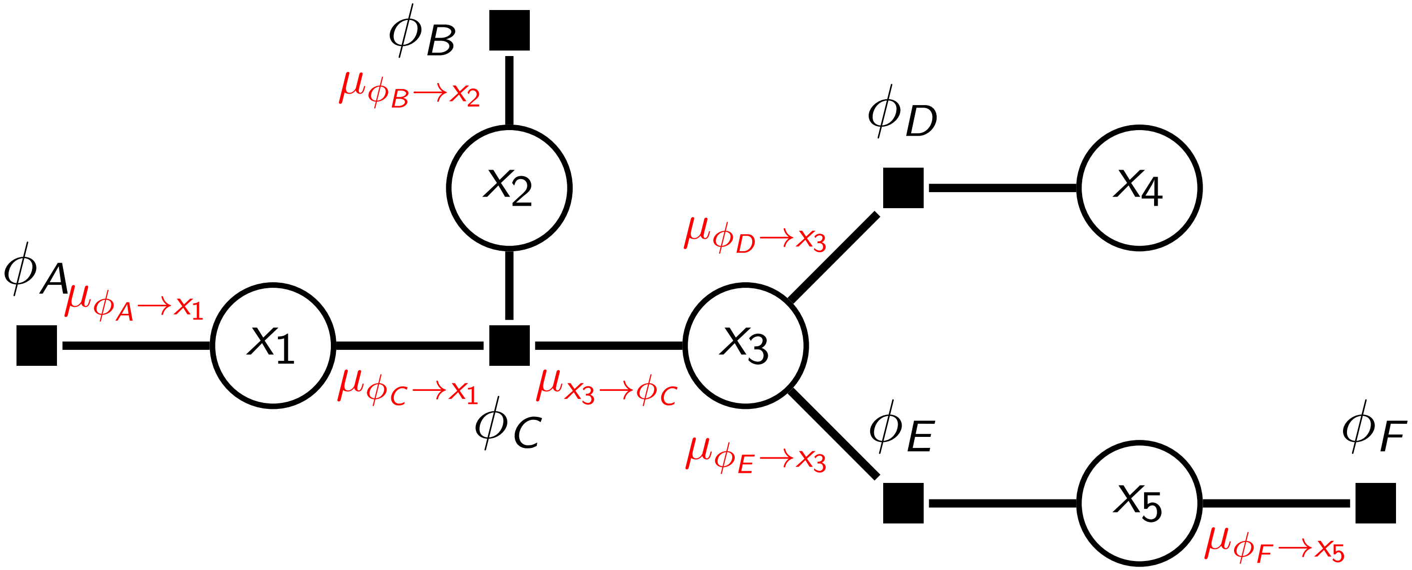
Effective factor for x_1 if all variables in the subtrees attached to ϕ_C are eliminated (subtrees do *not* include x_1)

Representing leaf-factors with messages

Since there are no variables "behind" the leaf-factors, all leaf-factors define effective factors themselves:

$$\mu_{\phi_A \to x_1}(x_1) = \phi_A(x_1)$$
$$\mu_{\phi_B \to x_2}(x_2) = \phi_B(x_2)$$
$$\mu_{\phi_F \to x_5}(x_5) = \phi_F(x_5)$$

We then obtain



Variables with single incoming messages copy the message

We had

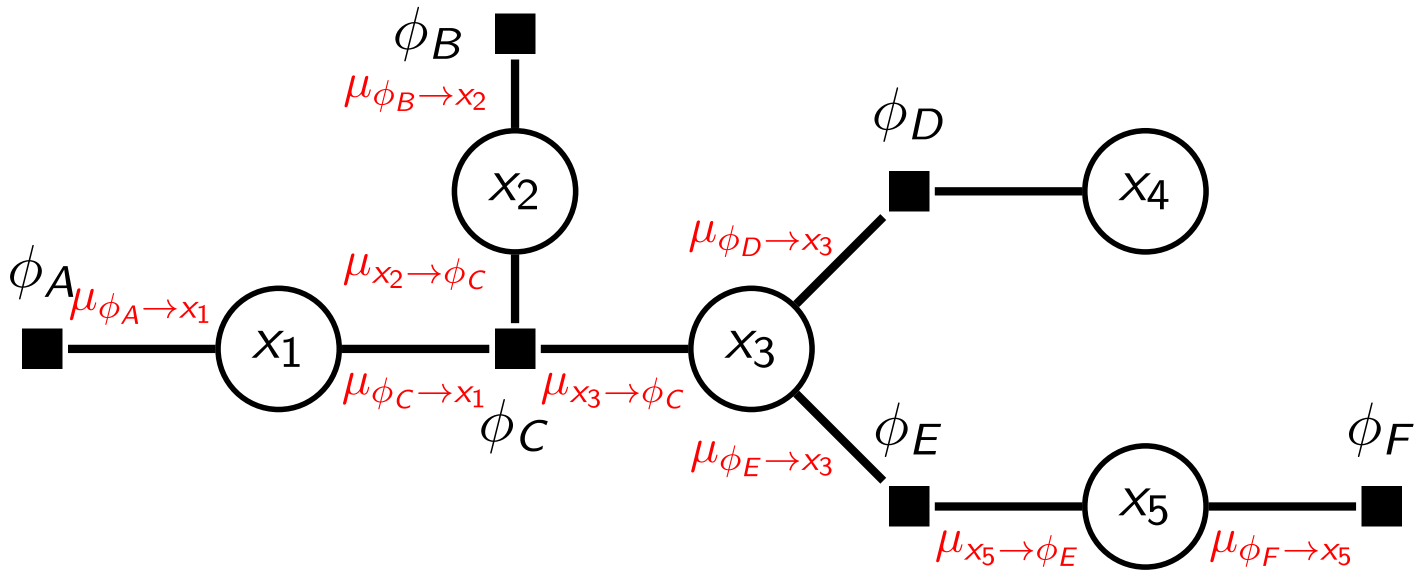
$$\mu_{x_3 \to \phi_C}(x_3) = \mu_{\phi_D \to x_3}(x_3)\mu_{\phi_E \to x_3}(x_3)$$

which corresponded to simplifying the factorisation by multiplying effective factors defined on the same domain. Special cases:

$$\mu_{x_5 \to \phi_E}(x_5) = \mu_{\phi_F \to x_5}(x_5)$$

$$\mu_{x_2 \to \phi_C}(x_2) = \mu_{\phi_B \to x_2}(x_2)$$

We then obtain



Messages from leaf variable nodes

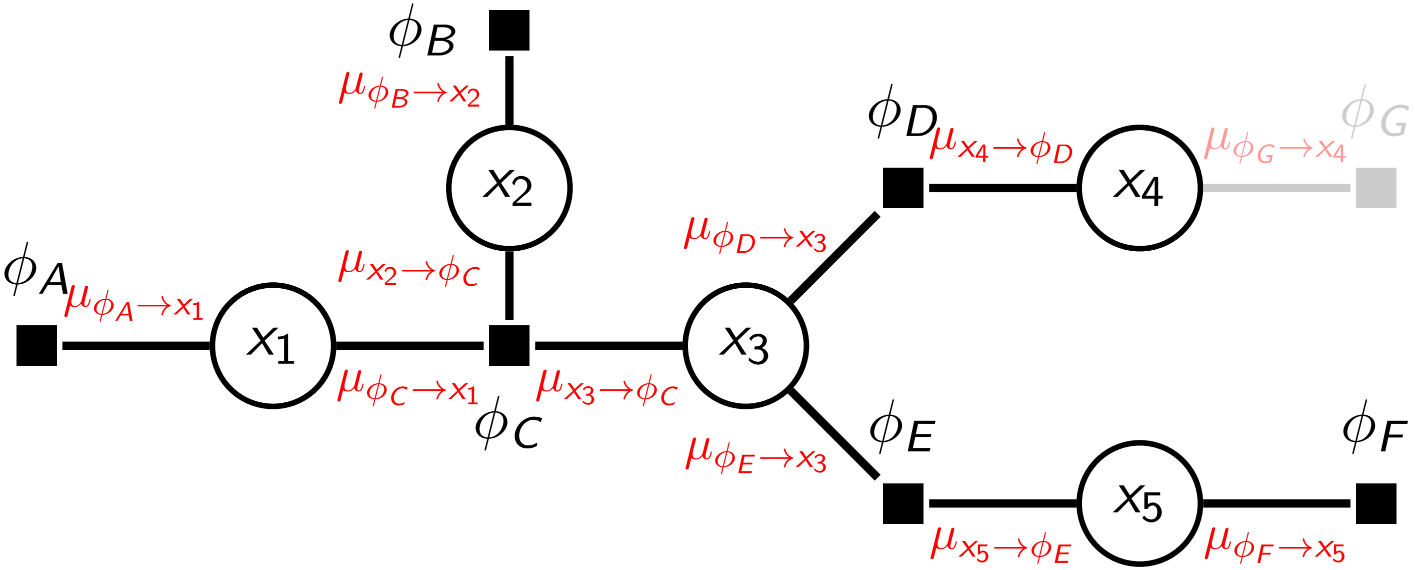
What about x_4? We can consider

$$p(x_1, \ldots, x_5) \propto \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\phi_D(x_3, x_4)\phi_E(x_3, x_5)\phi_F(x_5)$$

to include an additional factor $\phi_G(x_4) = 1$. We can thus set

$$\begin{aligned} \mu_{\phi_G \to x_4}(x_4) &= 1 \\ \mu_{x_4 \to \phi_D}(x_4) &= \mu_{\phi_G \to x_4}(x_4) = 1 \end{aligned}$$

Graph:

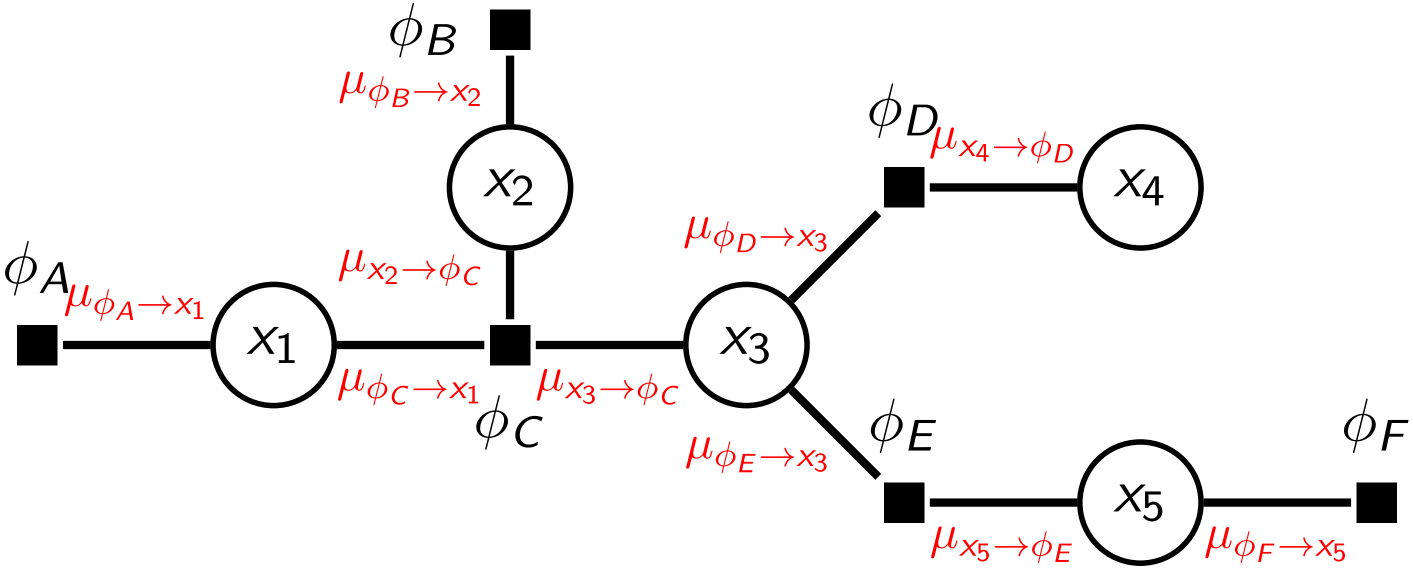


Single marginal from messages

We have seen that

$$\begin{aligned} p(x_1) &\propto \phi_A(x_1)\tilde{\phi}_{5432}(x_1) \\ &\propto \mu_{\phi_A \to x_1}(x_1)\mu_{\phi_C \to x_1}(x_1) \end{aligned}$$

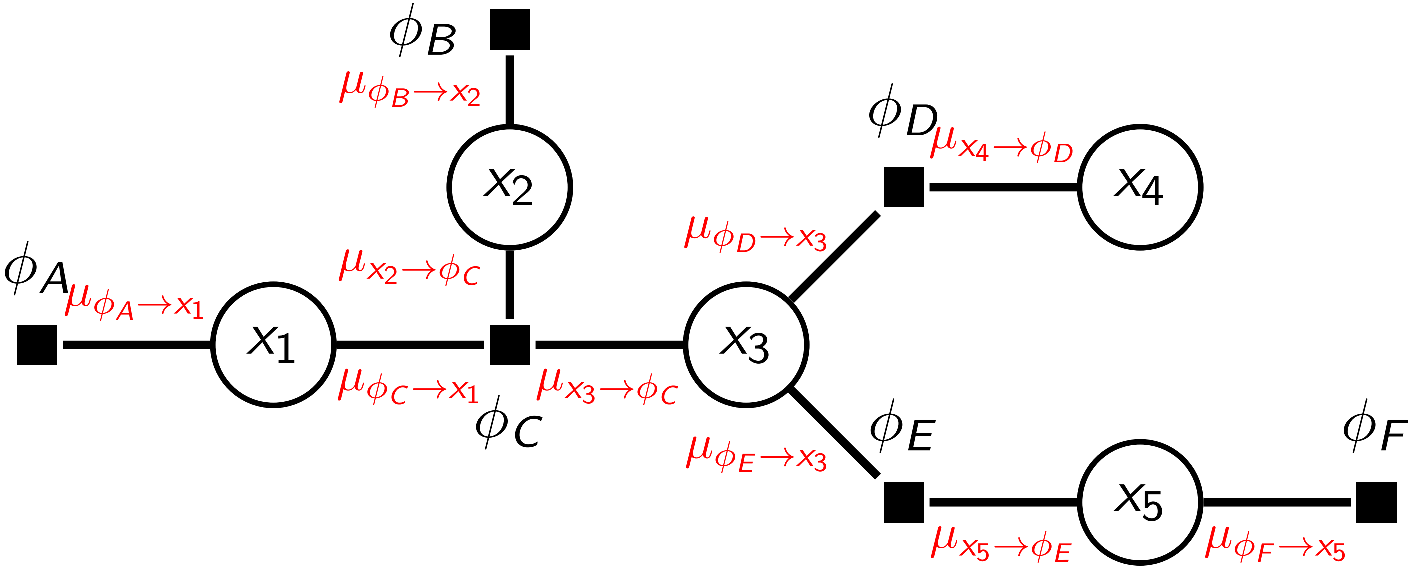
Marginal is proportional to the product of the incoming messages.



Single marginal from messages

Cost (due to properties of variable elimination):

- Linear in number of variables d, exponential in maximal number of variables attached to a factor node.
- Recycling: most messages do not depend on x_1 and can be re-used for computing $p(x_1)$ for any value of x_1 (as well as for computing the marginal distribution of other variables, see next slides)

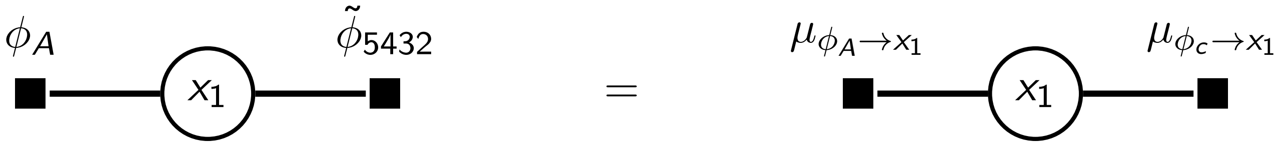


Further marginals from messages

- We have seen that

$$\begin{aligned} p(x_1) &\propto \phi_A(x_1)\tilde{\phi}_{5432}(x_1) \\ &\propto \mu_{\phi_A \to x_1}(x_1)\mu_{\phi_C \to x_1}(x_1) \end{aligned}$$

- Remember: Messages are effective factors

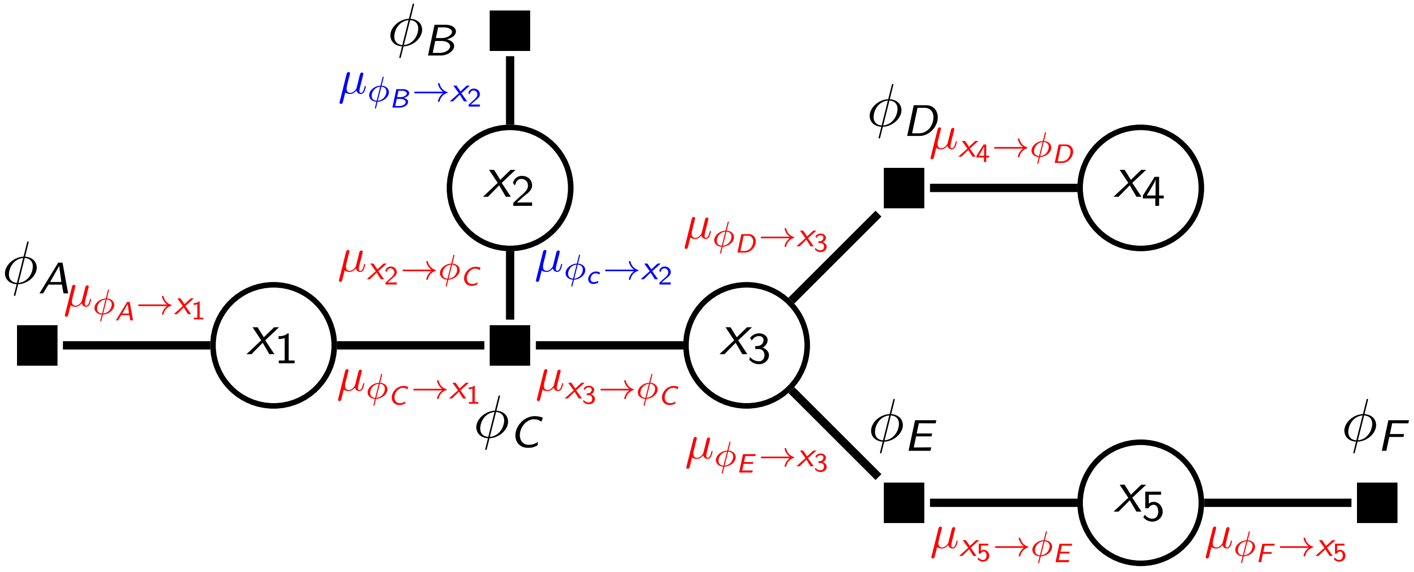


- This correspondence allows us to write down the marginal for other variables too. All we need are the incoming messages.

Further marginals from messages

- Example: For $p(x_2)$ we need $\mu_{\phi_B \to x_2}$ and $\mu_{\phi_C \to x_2}$
- $\mu_{\phi_B \to x_2}$ is known but $\mu_{\phi_C \to x_2}$ needs to be computed
- $\mu_{\phi_C \to x_2}$ corresponds to effective factor for x_2 if all variables of the subtrees attached to ϕ_c are eliminated.
- Can be computed from previously computed factors:

$$\mu_{\phi_A \to x_1} \qquad \text{and} \qquad \mu_{x_3 \to \phi_C}$$



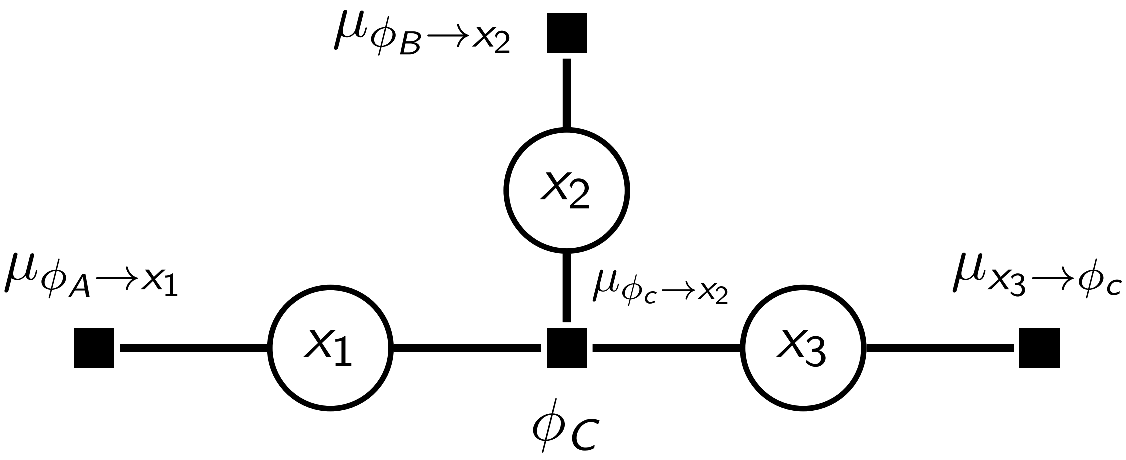
Further marginals from messages

- By definition of the messages, and their correspondence to effective factors, we have

$$p(x_1, x_2, x_3) \propto \phi_C(x_1, x_2, x_3)\mu_{\phi_A \to x_1}(x_1)\mu_{\phi_B \to x_2}(x_2)\mu_{x_3 \to \phi_C}(x_3)$$

- Eliminating x_1 and x_3 gives

$$\begin{aligned} p(x_2) &\propto \mu_{\phi_B \to x_2}(x_2) \sum_{x_1, x_3} \phi_c(x_1, x_2, x_3)\mu_{x_3 \to \phi_C}(x_3)\mu_{\phi_A \to x_1}(x_1) \\ &\propto \mu_{\phi_B \to x_2}(x_2)\mu_{\phi_C \to x_2}(x_2) \end{aligned}$$



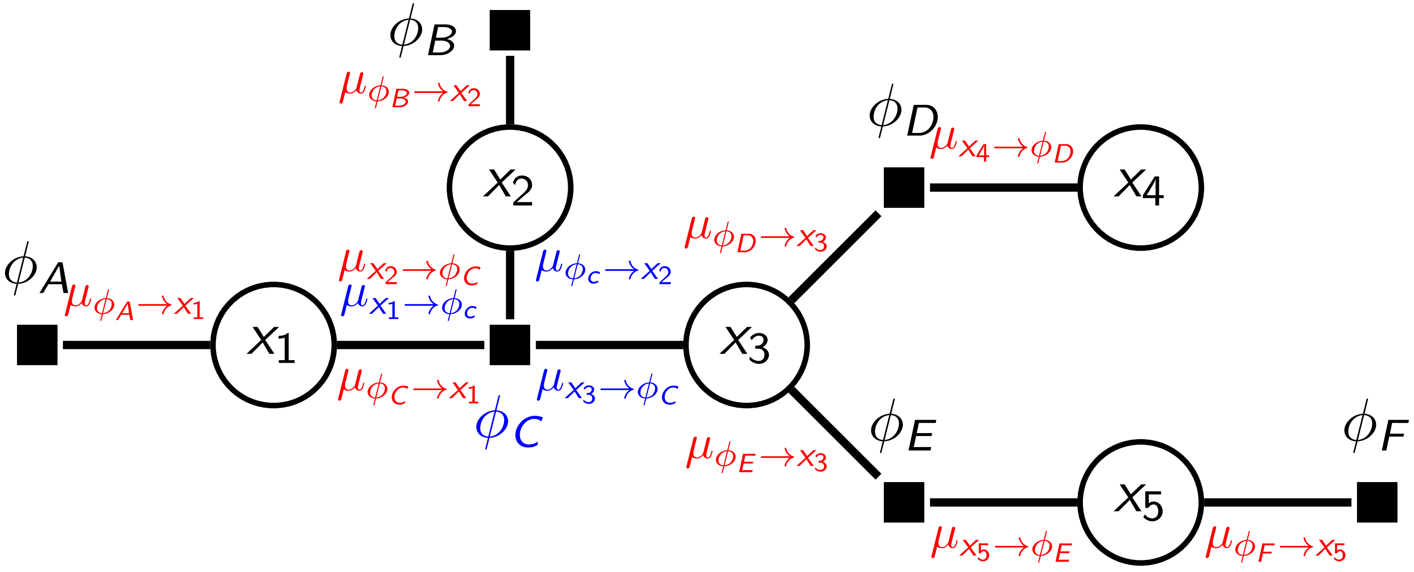
Further marginals from messages

We had

$$\mu_{\phi_C \to x_2}(x_2) = \sum_{x_1, x_3} \phi_c(x_1, x_2, x_3) \mu_{x_3 \to \phi_C}(x_3) \mu_{\phi_A \to x_1}(x_1)$$

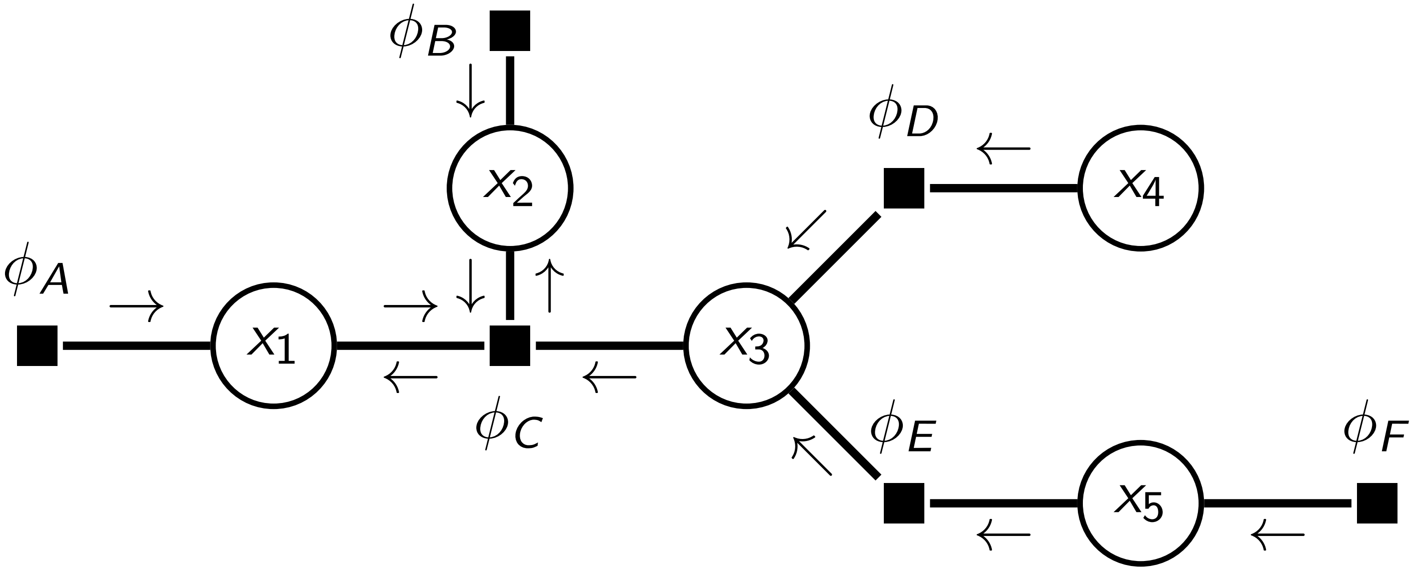
Introducing variable to factor message $\mu_{x_1 \to \phi_c} = \mu_{\phi_A \to x_1} = \phi_A$

$$\mu_{\phi_C \to x_2}(x_2) = \sum_{x_1, x_3} \phi_c(x_1, x_2, x_3) \mu_{x_3 \to \phi_C}(x_3) \mu_{x_1 \to \phi_c}(x_1)$$



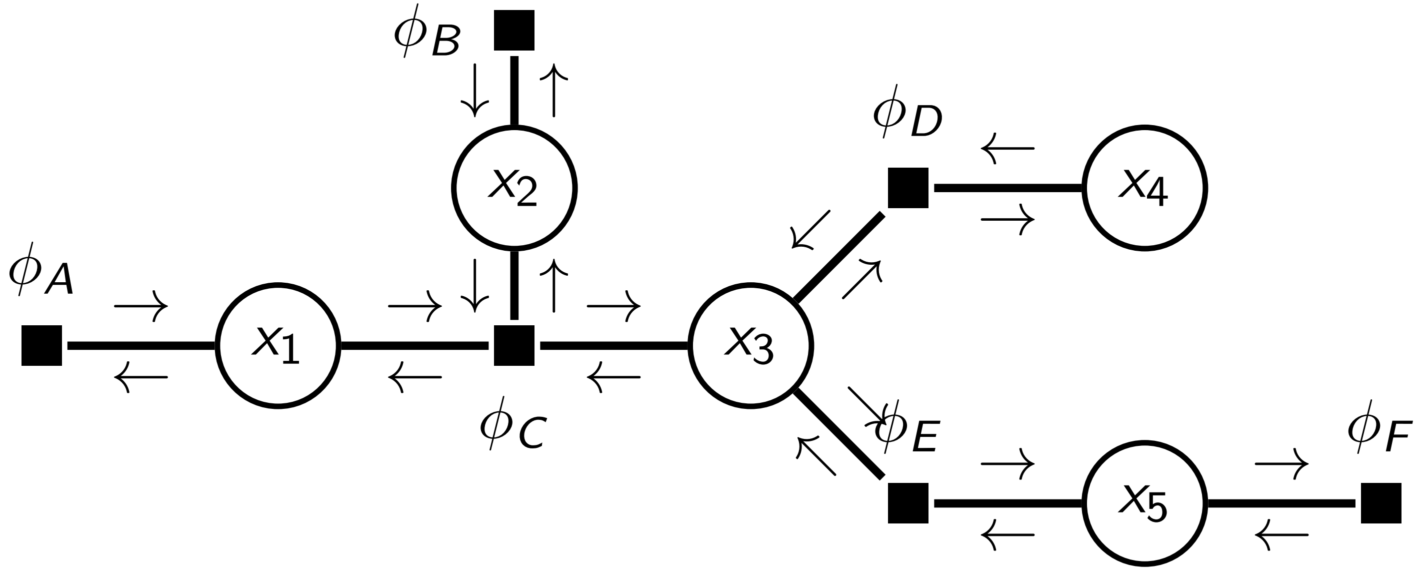
Using arrows to indicate the messages

Less cluttered representation using arrows for the messages



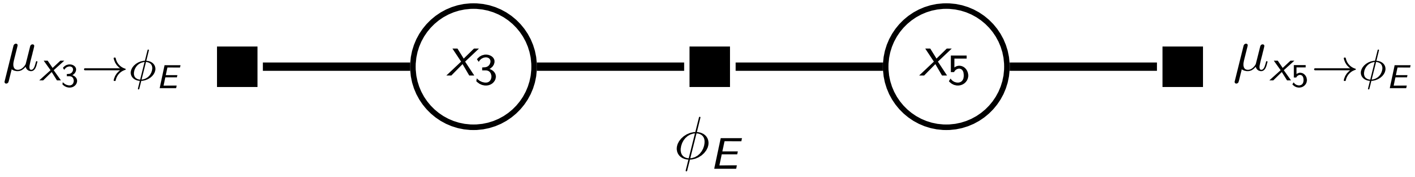
All (univariate) marginals from messages

- We can use the messages to compute the marginals of all variables in the graph.
- For the marginal of a variable x we need to know the incoming messages $\mu_{\phi_i \to x}$ from all factor nodes ϕ_i connected to x.
- This means that if each edge has a message in both directions, we can compute the marginals of all variables in the graph.



Joint distributions from messages

- The correspondence between messages and effective factors allows us to find the joint distribution for variables connected to the same factor node (neighbours).
- For example, we can compute $p(x_3, x_5)$ from messages
- The messages $\mu_{x_3 \to \phi_E}$ and $\mu_{x_5 \to \phi_E}$ correspond to effective factors attached to x_3 and x_5, respectively.



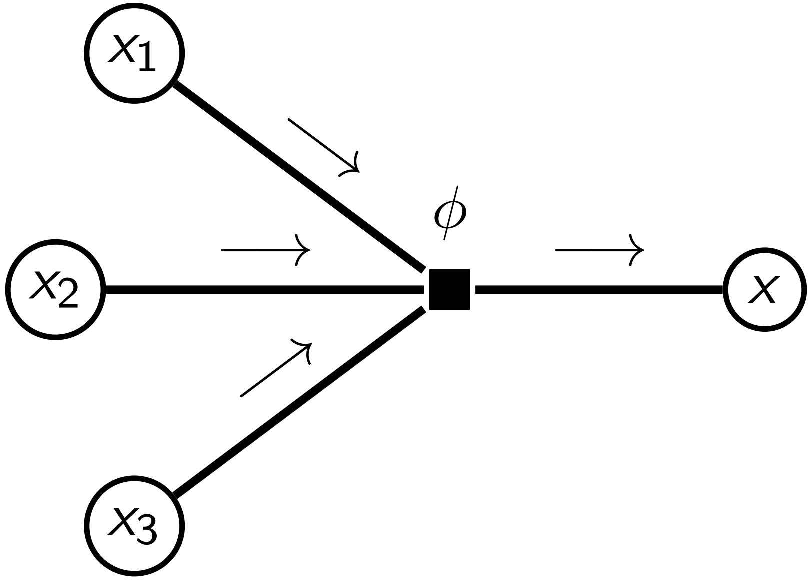
- Factor graph corresponds to

$$p(x_3, x_5) \propto \phi_E(x_3, x_5)\mu_{x_3 \to \phi_E}(x_3)\mu_{x_5 \to \phi_E}(x_5)$$

“Rules” of message passing: factor to variable messages

Note: The rules come from the fact that messages correspond to effective factors obtained after marginalisation.

$$\mu_{\phi \to x}(x) = \sum_{x_1, \ldots, x_j} \phi(x_1, \ldots, x_j, x) \prod_{i=1}^{j} \mu_{x_i \to \phi}(x_i)$$

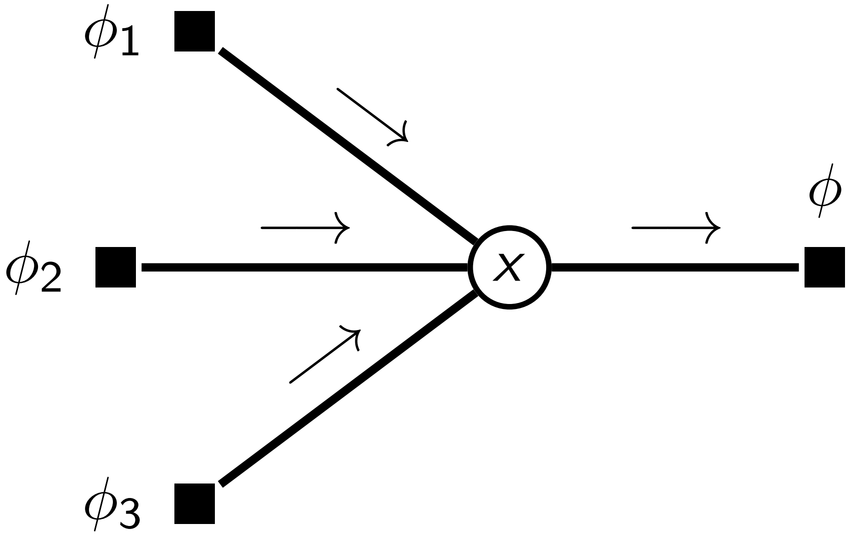


Rule corresponds to eliminating variables $x_1, \ldots, x_j$

"Rules" of message passing: variable to factor messages

Note: The rules come from the fact that messages correspond to effective factors obtained after marginalisation.

$$\mu_{x\to\phi}(x) = \prod_{i=1}^{j} \mu_{\phi_i\to x}(x)$$

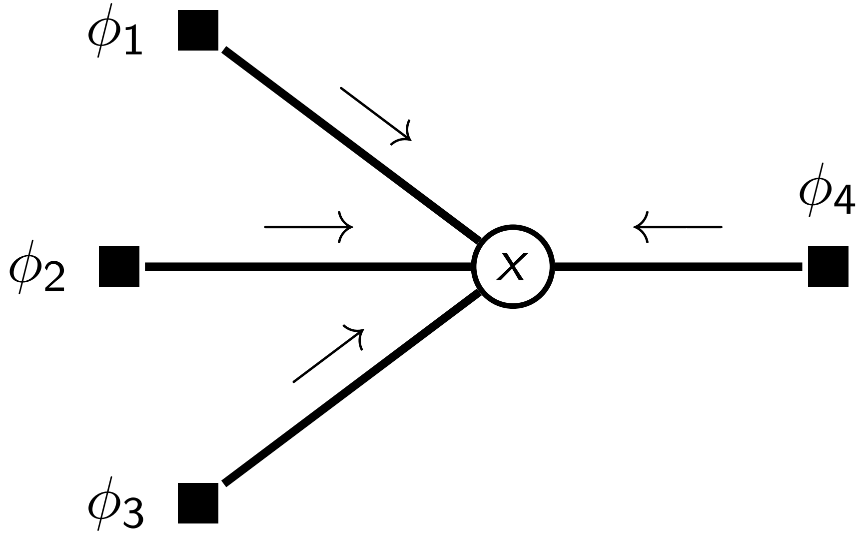


Rule corresponds to simplifying the factorisation by multiplying effective factors defined on the same domain.

"Rules" of message passing: univariate marginals

Note: The rules come from the fact that messages correspond to effective factors obtained after marginalisation.

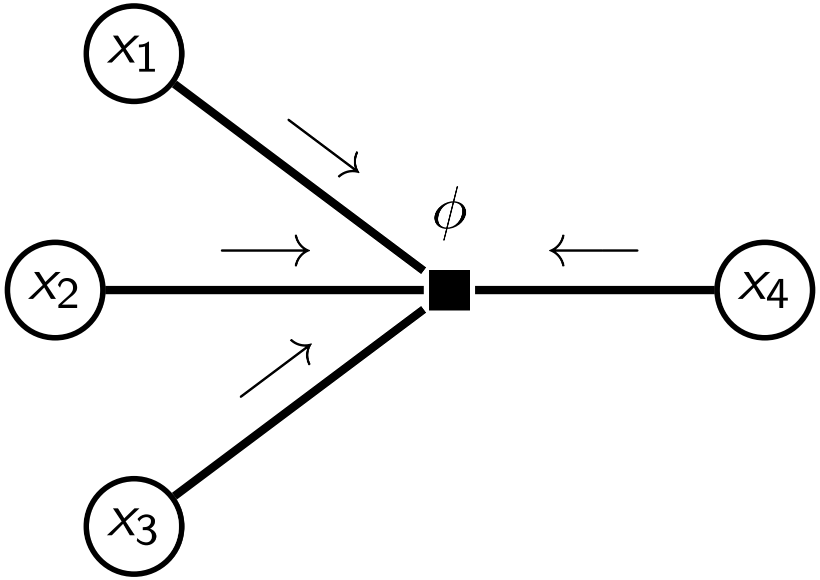
$$p(x) \propto \prod_{i=1}^{j} \mu_{\phi_i \to x}(x)$$



“Rules” of message passing: joint marginals

Note: The rules come from the fact that messages correspond to effective factors obtained after marginalisation.

$$p(x_1, \ldots, x_j) \propto \phi(x_1, \ldots, x_j) \prod_{i=1}^{j} \mu_{x_i \to \phi}(x_i)$$



A word about numerics

In practice, it is better to work with log-messages (see Barber's paragraph on "log messages", p86)

Other names for the sum-product algorithm

- Other names for the sum-product algorithm include
    - sum-product message passing
    - message passing
    - belief propagation
- Whatever the name: it is variable elimination applied to factor trees

Key advantages of the sum-product algorithm

Assume $p(x_1, \ldots, x_d) \propto \prod_{i=1}^{m} \phi_i(\mathcal{X}_i)$, with $\mathcal{X}_i \subseteq \{x_1, \ldots, x_d\}$, can be represented as a factor tree.

- The sum-product algorithm allows us to compute
  - *all* univariate marginals $p(x_i)$.
  - *all* joint distributions $p(\mathcal{X}_i)$ for the variables $\mathcal{X}_i$ that are part of the same factor ϕ_i.
- Cost: If variables can take maximally K values and there are maximally M elements in the $\mathcal{X}_i$: $O(2dK^M) = O(dK^M)$

Applicability of the sum-product algorithm

- Factor graph must be a tree
- Can be used to compute conditionals (same argument as for variable elimination)
- May be used for continuous random variables (same caveats as for variable elimination)
- Same ideas can be used to compute $\operatorname{argmax}_{\mathbf{x}} p(\mathbf{x})$

If the factor graph is not a tree

- Use variable elimination
- Group variables together so that the factor graph becomes a tree (for details, see Chapter 6 in Barber, or Section V in Kschischang et al, *Factor Graphs and the Sum-Product Algorithm*, 2001; not examinable)
- Pretend the factor graph is a tree and use message passing (loopy belief propagation; not examinable)
- Can you condition on some variables so that the conditional is a tree? Message passing can then be used to solve part of the inference problem.
  Example: $p(x_1, x_2, x_3, x_4)$ is not a tree but $p(x_1, x_2, x_3 | x_4)$ is.
  Use law of total probability

  $$p(x_1) = \sum_{x_4} \underbrace{\sum_{x_2, x_3} p(x_1, x_2, x_3 | x_4)}_{\text{by message passing}} p(x_4)$$

  (see Barber Section 5.3.2, "Loop-cut conditioning")

Summary

1. Marginal inference by variable elimination
    - Exploiting the factorisation by using the distributive law $ab + ac = a(b + c)$ and by caching computations
    - Variable elimination for general factor graphs
    - Structural changes to the graph due to variable elimination
    - The principles of variable elimination also apply to continuous random variables

2. Marginal inference for factor trees (sum-product algorithm)
    - Factor trees
    - Sum-product algorithm = variable elimination for factor trees
    - Messages = effective factors
    - The rules for sum-product message passing

Program

1. Marginal inference by variable elimination
2. Marginal inference for factor trees (sum-product algorithm)
3. Inference of most probable states
   - Maximisers of the marginals $\neq$ maximiser of joint
   - We can use the distributive law $\max(ab, ac) = a \max(b, c)$ to exploit the factorisation
   - Max-product algorithm and back-tracking

Other inference tasks

- So far: given a joint distribution $p(\mathbf{x})$, find marginals or conditionals over variables
- Other common inference task:
  - Find a setting of the variables that maximises $p(\mathbf{x})$, i.e.

$$\underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{x})$$

  - Find the corresponding value of $p(\mathbf{x})$, i.e.

$$\max_{\mathbf{x}} p(\mathbf{x})$$

- Note: the $\operatorname{argmax}_{\mathbf{x}} p(\mathbf{x})$ task here includes $\operatorname{argmax}_{\mathbf{x}} \tilde{p}(\mathbf{x}|\mathbf{y}_o)$, which is known as maximum a-posteriori (MAP) estimation or inference.

Maximisers of the marginals $\neq$ maximiser of joint

- The sum-product algorithm gives us the univariate marginals $p(x_i)$ for all variables $x_1, \ldots, x_d$.
- But the vector with the $\operatorname{argmax}_{x_i} p(x_i)$, $x_1, \ldots, x_d$, is not the same as $\operatorname{argmax}_{\mathbf{x}} p(\mathbf{x})$
- Example (Bishop Table 8.1):

x_1	x_2	$p(x_1, x_2)$
0	0	0.3
1	**0**	0.4
0	1	0.3
1	1	0.0

x_1	$p(x_1)$
0	0.6
1	0.4

x_2	$p(x_2)$
0	0.7
1	0.3

Using the distributive law to exploit the factorisation

- For marginal inference, we relied on the distributive law

$$ab + ac = a(b + c)$$

$$\text{sum}(ab, ac) = a\,\text{sum}(b, c)$$

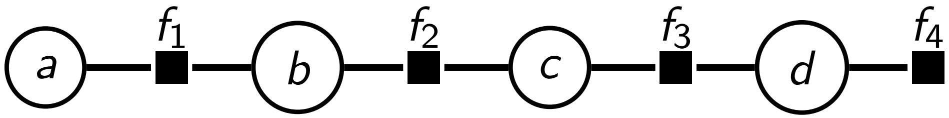
- For finding the most probable state, use similarly

$$\max(ab, ac) = a \max(b, c)$$

Example (chain)

(Based on a slide courtesy of David Barber)

$$p(a, b, c, d) \propto f_1(a, b) f_2(b, c) f_3(c, d) f_4(d)$$



For marginal inference, we had

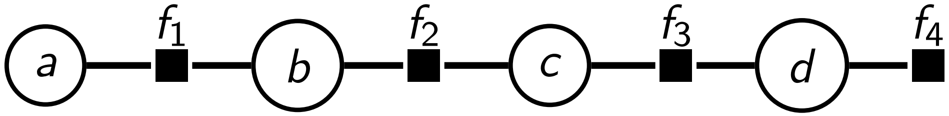
$$p(a) \propto \sum_b \sum_c \sum_d f_1(a, b) f_2(b, c) f_3(c, d) f_4(d)$$

$$\propto \underbrace{\sum_b f_1(a, b) \underbrace{\left[\sum_c f_2(b, c) \underbrace{\left[\sum_d f_3(c, d) f_4(d) \right]}_{\mu_{f_3 \to c} = \mu_{c \to f_2}} \right]}_{\mu_{f_2 \to b} = \mu_{b \to f_1}}}_{\mu_{f_1 \to a}}$$

Example (chain)

(Based on a slide courtesy of David Barber)

$$p(a,b,c,d) \propto f_1(a,b) f_2(b,c) f_3(c,d) f_4(d)$$



For finding $\max p(a,b,c,d)$:

$$\max_{a,b,c,d} p(a,b,c,d) = \frac{1}{Z} \max_a \max_b \max_c \max_d f_1(a,b) f_2(b,c) f_3(c,d) f_4(d)$$

$$= \frac{1}{Z} \max_a \underbrace{\max_b f_1(a,b) \underbrace{\left[\max_c f_2(b,c) \underbrace{\left[\max_d f_3(c,d) f_4(d) \right]}_{\gamma_{f_3 \to c} = \gamma_{c \to f_2}} \right]}_{\gamma_{f_2 \to b} = \gamma_{b \to f_1}}}_{\gamma_{f_1 \to a}}$$

As before for the sum-product algorithm, the $\gamma_{\to}$ denote messages

Example (chain)

$$\max_{a,b,c,d} p(a,b,c,d) = \frac{1}{Z} \max_a \underbrace{\max_b f_1(a,b) \underbrace{\left[\max_c f_2(b,c) \underbrace{\left[\max_d f_3(c,d) f_4(d) \right]}_{\gamma_{f_3 \to c}(c) = \mu_{c \to f_2}(c)} \right]}_{\gamma_{f_2 \to b}(b) = \gamma_{b \to f_1}(b)}}_{\gamma_{f_1 \to a}(a)}$$

How to compute $\mathrm{argmax}\, p(a,b,c,d)$?

Example (chain)

$$\max_{a,b,c,d} p(a,b,c,d) = \frac{1}{Z} \max_a \underbrace{\max_b f_1(a,b) \underbrace{\left[\max_c f_2(b,c) \underbrace{\left[\max_d f_3(c,d) f_4(d) \right]}_{\gamma_{f_3 \to c}(c) = \mu_{c \to f_2}(c)} \right]}_{\gamma_{f_2 \to b}(b) = \gamma_{b \to f_1}(b)}}_{\gamma_{f_1 \to a}(a)}$$

Consider $\max_d f_3(c,d) f_4(d)$:

- This is an optimisation problem that needs to be solved for all values of c.
- Maximiser $d^* = \operatorname{argmax}_d f_3(c,d) f_4(d)$ depends on c:

$$d^*(c) = \operatorname*{argmax}_d f_3(c,d) f_4(d)$$

- $d^*(c)$ is a function (look-up table) that returns the optimal value for d for any value of c.

Example (chain)

$$\max_{a,b,c,d} p(a,b,c,d) = \frac{1}{Z} \max_a \underbrace{\max_b f_1(a,b) \underbrace{\left[\max_c f_2(b,c) \underbrace{\left[\max_d f_3(c,d) f_4(d) \right]}_{\gamma_{f_3 \to c}(c) = \mu_{c \to f_2}(c)} \right]}_{\gamma_{f_2 \to b}(b) = \gamma_{b \to f_1}(b)}}_{\gamma_{f_1 \to a}(a)}$$

In addition to $d^*(c) = \mathrm{argmax}_d\, f_3(c,d) f_4(d)$, we further have:

$$c^*(b) = \underset{c}{\mathrm{argmax}}\, f_2(b,c)\gamma_{c \to f_2}(c)$$

$$b^*(a) = \underset{b}{\mathrm{argmax}}\, f_1(a,b)\gamma_{b \to f_1}(b)$$

$$\hat{a} = \underset{a}{\mathrm{argmax}}\, \gamma_{f_1 \to a}(a)$$

After $\hat{a}$ has been computed, we can compute $\mathrm{argmax}\, p(a,b,c,d)$ via $\hat{b} = b^*(\hat{a})$, $\hat{c} = c^*(\hat{b})$, and $\hat{d} = d^*(\hat{c})$ ("back-tracking")

Max-product algorithm

- The above example for a chain extends to general factor graphs (like in variable elimination)
- max takes the place of $\sum$
- For factor trees: sum-product algorithm becomes max-product algorithm with corresponding rules of how to compute the corresponding messages (see Barber, Section 5.2.1)
- Messages for the max-product algorithm are called max-product messages.
- For numerical stability, it is better to implement the algorithms using log messages: max-product algorithm becomes max-sum algorithm (see Bishop, 8.4.5)

Program recap

1. Marginal inference by variable elimination
    - Exploiting the factorisation by using the distributive law $ab + ac = a(b + c)$ and by caching computations
    - Variable elimination for general factor graphs
    - Structural changes to the graph due to variable elimination
    - The principles of variable elimination also apply to continuous random variables
2. Marginal inference for factor trees (sum-product algorithm)
    - Factor trees
    - Sum-product algorithm $=$ variable elimination for factor trees
    - Messages $=$ effective factors
    - The rules for sum-product message passing
3. Inference of most probable states
    - Maximisers of the marginals $\neq$ maximiser of joint
    - We can use the distributive law $\max(ab, ac) = a \max(b, c)$ to exploit the factorisation
    - Max-product algorithm and back-tracking