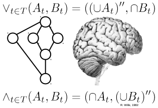
Bayesian networks: efficient marginalization

Bayesian Statistics and Machine Learning

Dominik Endres

November 12, 2015

Philipps Universität Marburg

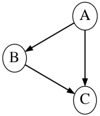


Outline

Summary: Bayesian networks

A type of probabilistic graphical model which expresses conditional (in)dependence relationships.

Random variables	Bayesian networks
Random variables A, B, C	Nodes of a graph
Conditional (in)dependence	(No) directed edges
Chain rule decomposition	directed acyclic graph (DAG)
Typical correspondences	
Observed variable	bottom node
Hidden variable	node towards top



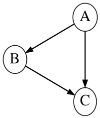
A Bayesian network with 3 random variables A,B,C.

Question: what can we do with Bayesian networks?

Summary: Bayesian networks

A type of probabilistic graphical model which expresses conditional (in)dependence relationships.

Random variables	**Bayesian networks**
Random variables A, B, C	Nodes of a graph
Conditional (in)dependence	(No) directed edges
Chain rule decomposition	directed acyclic graph (DAG)
Typical correspondences	
Observed variable	bottom node
Hidden variable	node towards top



A Bayesian network with 3 random variables A,B,C.

Question: what can we do with Bayesian networks?

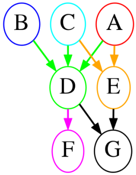
Inference

Our knowledge about the random variables of interest is expressed in their joint probability distribution.

A Bayesian network describes *relationships* between the random variables which appear in it.

Question: how does our knowledge about (a subset) of the random variables change, if learn something about (another subset) the random variables?

Important case:



W learn/observe the *value* of some random variables, say $G = g$ and $F = f$

We would like to know how that changes our beliefs about B, i.e. $P(B|f, g)$.

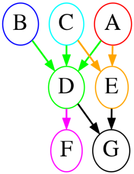
Since B is above G, this is an instance of **inference**.

$\approx$ Inference is going for the *effects* to the *causes*.

Inference

Our knowledge about the random variables of interest is expressed in their joint probability distribution.
A Bayesian network describes *relationships* between the random variables which appear in it.
Question: how does our knowledge about (a subset) of the random variables change, if learn something about (another subset) the random variables?

Important case:

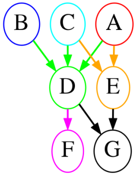


W learn/observe the *value* of some random variables, say $G = g$ and $F = f$
We would like to know how that changes our beliefs about B, i.e. $P(B|f, g)$.
Since B is above G, this is an instance of **inference**.
$\approx$ Inference is going for the *effects* to the *causes*.

Inference

Our knowledge about the random variables of interest is expressed in their joint probability distribution.
A Bayesian network describes *relationships* between the random variables which appear in it.
Question: how does our knowledge about (a subset) of the random variables change, if learn something about (another subset) the random variables?

Important case:



W learn/observe the *value* of some random variables, say $G = g$ and $F = f$
We would like to know how that changes our beliefs about B, i.e. $P(B|f, g)$.
Since B is above G, this is an instance of **inference**.
$\approx$ Inference is going for the *effects* to the *causes*.

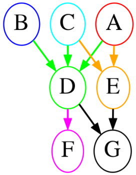
Inference

Our knowledge about the random variables of interest is expressed in their joint probability distribution.

A Bayesian network describes *relationships* between the random variables which appear in it.

Question: how does our knowledge about (a subset) of the random variables change, if learn something about (another subset) the random variables?

Important case:



W learn/observe the *value* of some random variables, say $G = g$ and $F = f$

We would like to know how that changes our beliefs about B, i.e. $P(B|f, g)$.

Since B is above G, this is an instance of **inference**.

$\approx$ Inference is going for the *effects* to the *causes*.

Inference: examples

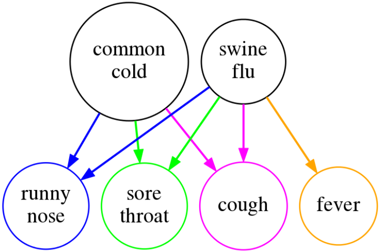
Typical **inference** scenario:

- *Observe* values of observable nodes (lower in the network).
- *Infer* (marginal) probabilities of hidden nodes, typically the ancestors of the observable nodes.

	Bayesian network	Examples	
		die rolls	medical diagnosis
Observe	observable nodes	die roll outcomes	symptoms
Infer	hidden nodes	die fairness	diseases

Inference: swine flu vs. common cold

Question: given observable symptoms, what are the probabilities of suffering from swine flu vs. the common cold?

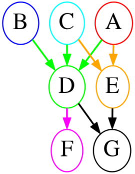


Intuition from the graph: if a fever is observed, swine flu should become more probable, whereas a runny nose should make both the common cold and the swine flu more probable.

Inference: observed nodes

Inference: observe some nodes (lower down) in the network, compute marginal distribution of nodes higher up in the network given these observations.

Graphical notation: observed nodes are shaded or colored.

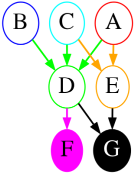


observe

$\Rightarrow$

$G = g$

$F = f$



Marginals: $P(A)$ $\qquad P(A|f, g) = \frac{P(A, f, g)}{P(f, g)}$

$P(B, C)$ $\qquad P(B, C|f, g) = \frac{P(B, C, f, g)}{P(f, g)}$

Inference: marginalization and conditioning

To compute $P(A|f, g)$, we need

$$P(A|f, g) = \frac{P(A, f, g)}{P(f, g)}$$

i.e. the marginals

$$\begin{aligned} P(A, f, g) &= \sum_{\sim\{A,F,G\}} P(A, B, C, D, E, F = f, G = g) \\ P(f, g) &= \sum_{\sim\{F,G\}} P(A, B, C, D, E, F = f, G = g) \end{aligned}$$

where $\sum_{\sim\{A,F,G\}}$ indicates the 'not-sum' or *summary*, which is the sum over all random variables except for $\{A, F, G\}$.

$\Rightarrow$ To do efficient inference, we need efficient marginalization methods.

Efficient: small number of operations for evaluation

Inference: marginalization and conditioning

To compute $P(A|f,g)$, we need

$$P(A|f,g) = \frac{P(A,f,g)}{P(f,g)}$$

i.e. the marginals

$$\begin{aligned} P(A,f,g) &= \sum_{\sim\{A,F,G\}} P(A,B,C,D,E,F=f,G=g) \\ P(f,g) &= \sum_{\sim\{F,G\}} P(A,B,C,D,E,F=f,G=g) \end{aligned}$$

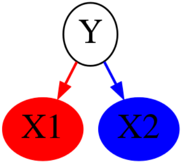
where $\sum_{\sim\{A,F,G\}}$ indicates the 'not-sum' or *summary*, which is the sum over all random variables except for $\{A,F,G\}$.

$\Rightarrow$ To do efficient inference, we need efficient marginalization methods.

Efficient: small number of operations for evaluation.

Example: die is rolled twice

A die is rolled twice. We don't know if the die is fair or loaded.
Random variables: X_1, X_2: value of 1st and 2nd roll, Y: fairness.
The rolls are observed, we'd like to compute posterior probability of fairness:

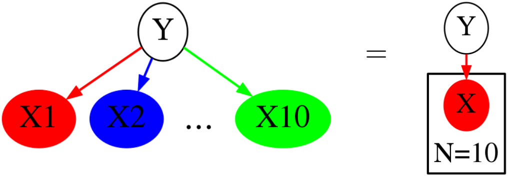


$$P(Y|x_1, x_2)$$

Repeated observations: plates

A die is rolled N times. We don't know if the die is fair or loaded. Random variables: $X_1, \ldots, X_N$: value of rolls, Y: fairness.

If a (part of) a graph is enclosed in a rectangle, usually with a solid border, then this part is repeated N times. This graph element is called a **plate**.



Plates are a useful shorthand notation for i.i.d. random variables.

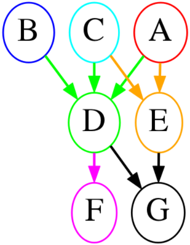
Prediction

Prediction comes in 3 flavours:

- *Marginal*: Compute the marginal probability of some node, e.g. $P(F)$.
  - Answers: what does the model (i.e. the Bayes net) predict for observable variables?
- *Conditional on ancestors*: Compute the conditional probability of some (observable) node, given a subset of its ancestors. E.g. $P(G|A, B)$.
  - The inverse of inference: going from the causes to the effects.
- *Conditional on other observations*: e.g. prediction future events like $P(X_3|x_1, x_2)$.

Prediction: marginal

Question: what does the model predict for $P(F)$?
E.g. how frequent is a symptom in the population?
Answer: marginalize all other random vars. out of the joint distribution:



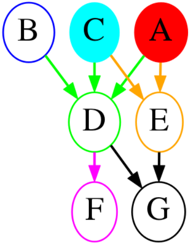
$\Rightarrow$ efficient marginalization is important if this type of prediction is to be done efficiently.

Prediction: conditional on ancestors

Question: what does the model predict for $P(F)$, given that we know $A = a$ and $C = c$?

E.g. how probable is a symptom given that the patient has diseases a and c?

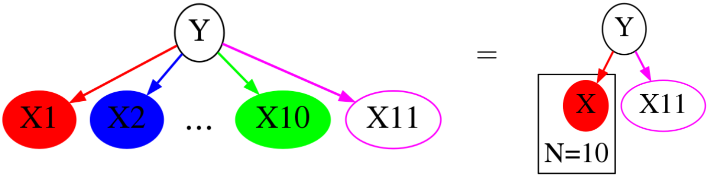
Answer: compute $P(F|a, c) = \frac{P(F,a,c)}{P(a,c)}$.



$\Rightarrow$ efficient marginalization is important if this type of prediction is to be done efficiently.

Prediction: conditional other (previous) observations

Question: what does the model predict for $P(X_{11})$, given that we know $x_1, \dots, x_{10}$?
Answer: compute $P(X_{N+1}|x_1, \dots, x_N) = \frac{P(X_{11}, x_1, \dots, x_{10})}{P(x_1, \dots, x_{10})}$.



$\Rightarrow$ efficient marginalization is important if this type of prediction is to be done efficiently.

Efficient marginalization for inference and prediction

Both inference and prediction are done via marginalization (and conditioning).

Marginalization can be computationally costly: assume M random variables, each of which can take on K different values. Then evaluating

$$P(X_1) = \sum_{\sim X_1} P(X_1, \ldots, X_M)$$

requires $\mathcal{O}(K^M)$ operations.

$\Rightarrow$ **Infeasible** to compute even for moderate values of M.

But we have not used the structural information in the graph yet.

Can *conditional independence* statements be used to reduce the computational effort?

Efficient marginalization for inference and prediction

Both inference and prediction are done via marginalization (and conditioning).
Marginalization can be computationally costly: assume M random variables, each of which can take on K different values. Then evaluating

$$P(X_1) = \sum_{\sim X_1} P(X_1, \ldots, X_M)$$

requires $\mathcal{O}(K^M)$ operations.
$\Rightarrow$ **Infeasible** to compute even for moderate values of M.
But we have not used the structural information in the graph yet. Can *conditional independence* statements be used to reduce the computational effort?

Efficient marginalization for inference and prediction

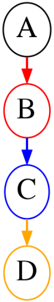
Both inference and prediction are done via marginalization (and conditioning).
Marginalization can be computationally costly: assume M random variables, each of which can take on K different values. Then evaluating

$$P(X_1) = \sum_{\sim X_1} P(X_1, \ldots, X_M)$$

requires $\mathcal{O}(K^M)$ operations.
$\Rightarrow$ **Infeasible** to compute even for moderate values of M.
But we have not used the structural information in the graph yet.
Can *conditional independence* statements be used to reduce the computational effort?

Efficient marginalization using conditional independence

Introductory example: (Markov) chain of 4 random variables.
We would like to compute $P(D)$.



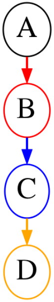
Chain rule decomposition:
$P(A, B, C, D) = P(A)P(B|A)P(C|A, B)P(D|A, B, C)$

Use conditional independence info. from graph:
$P(A, B, C, D) = P(A)P(B|A)P(C|B)P(D|C)$

Marginalize:
$P(D) = \sum_A \sum_B \sum_C P(A)P(B|A)P(C|B)P(D|C)$

Efficient marginalization using conditional independence

Introductory example: (Markov) chain of 4 random variables.
We would like to compute $P(D)$.



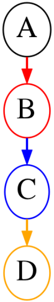
Chain rule decomposition:
$P(A, B, C, D) = P(A)P(B|A)P(C|A, B)P(D|A, B, C)$

Use conditional independence info. from graph:
$P(A, B, C, D) = P(A)P(B|A)P(C|B)P(D|C)$

Marginalize:
$P(D) = \sum_A \sum_B \sum_C P(A)P(B|A)P(C|B)P(D|C)$

Efficient marginalization using conditional independence

Introductory example: (Markov) chain of 4 random variables.
We would like to compute $P(D)$.



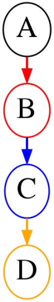
Chain rule decomposition:
$P(A, B, C, D) = P(A)P(B|A)P(C|A, B)P(D|A, B, C)$

Use conditional independence info. from graph:
$P(A, B, C, D) = P(A)P(B|A)P(C|B)P(D|C)$

Marginalize:
$P(D) = \sum_A \sum_B \sum_C P(A)P(B|A)P(C|B)P(D|C)$

Efficient marginalization using conditional independence

Introductory example: (Markov) chain of 4 random variables.
We would like to compute $P(D)$.



Chain rule decomposition:
$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$

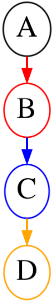
Use conditional independence info. from graph:
$P(A,B,C,D) = P(A)P(B|A)P(C|B)P(D|C)$

Marginalize:
$P(D) = \sum_A \sum_B \sum_C P(A)P(B|A)P(C|B)P(D|C)$

Efficient marginalization: direct evaluation

Reminder:
Assume A, B, C, D can take on K different values.

Question: how much computational effort can we save using distributivity?



Marginalize:

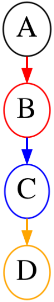
$$P(D) = \sum_A \sum_B \sum_C P(A) P(B|A) P(C|B) P(D|C)$$

Doing sums directly: K^4 calculations.
For chain of length M: K^M calculations.

Efficient marginalization: direct evaluation

Reminder:
Assume A, B, C, D can take on K different values.

Question: how much computational effort can we save using distributivity?



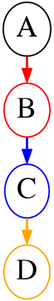
Marginalize:

$$P(D) = \sum_A \sum_B \sum_C P(A) P(B|A) P(C|B) P(D|C)$$

Doing sums directly: K^4 calculations.
For chain of length M: K^M calculations.

Efficient marginalization: distributivity

Question: how much computational effort can we save using distributivity?



Marginalize:
$P(D) = \sum_A \sum_B \sum_C P(A)P(B|A)P(C|B)P(D|C)$

Reminder: distributivity

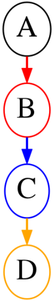
$\forall a, b, c \in \mathbb{R} : a \cdot b + a \cdot c = a \cdot (b + c)$

Use distributivity to 'push in' the sums as far as possible. Push each sum past all factors which do not depend on summand:
$P(D) = \sum_C P(D|C) \sum_B P(C|B) \sum_A P(B|A)P(A)$

Efficient marginalization: distributivity

Question: how much computational effort can we save using distributivity?

Marginalize:
$P(D) = \sum_A \sum_B \sum_C P(A) P(B|A) P(C|B) P(D|C)$



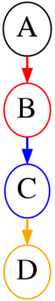
Reminder: distributivity

$\forall a, b, c \in \mathbb{R} : a \cdot b + a \cdot c = a \cdot (b + c)$

Use distributivity to 'push in' the sums as far as possible. Push each sum past all factors which do not depend on summand:
$P(D) = \sum_C P(D|C) \sum_B P(C|B) \sum_A P(B|A) P(A)$

Efficient marginalization: distributivity

Question: how much computational effort can we save using distributivity?



Marginalize:

$$P(D) = \sum_A \sum_B \sum_C P(A)P(B|A)P(C|B)P(D|C)$$

Reminder: distributivity

$\forall a, b, c \in \mathbb{R} : a \cdot b + a \cdot c = a \cdot (b + c)$

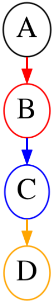
Use distributivity to 'push in' the sums as far as possible. Push each sum past all factors which do not depend on summand:

$$P(D) = \sum_C P(D|C) \sum_B P(C|B) \sum_A P(B|A)P(A)$$

Efficient marginalization: direct vs. distributive

Reminder:
Assume A, B, C, D can take on K different values.
Direct evaluation of marginal $P(D)$ takes $\mathcal{O}(K^M)$ operations.

Question: how much computational effort can we save using distributivity?



Use distributivity, then marginalize:

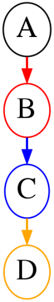
$$P(D) = \sum_C P(D|C) \sum_B P(C|B) \sum_A P(B|A)P(A)$$

For each value of B, we compute a sum of length K over values of A, yielding $P(B)$.
Repeat for sums over B and C.
$\Rightarrow$ $3K^2$ calculcations per value of D.
$\Rightarrow$ For a chain of length M: $\mathcal{O}(MK^2)$ calculations.

Efficient marginalization: direct vs. distributive

Reminder:
Assume A, B, C, D can take on K different values.
Direct evaluation of marginal $P(D)$ takes $\mathcal{O}(K^M)$ operations.

Question: how much computational effort can we save using distributivity?



Use distributivity, then marginalize:

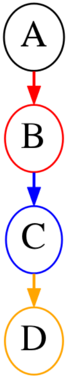
$$P(D) = \sum_C P(D|C) \sum_B P(C|B) \sum_A P(B|A) P(A)$$

For each value of B, we compute a sum of length K over values of A, yielding $P(B)$.
Repeat for sums over B and C.
$\Rightarrow 3K^2$ calculcations per value of D.
$\Rightarrow$ For a chain of length M: $\mathcal{O}(MK^2)$ calculations.

Computing other marginals 'on the way'

Reminder:
$P(A, B, C, D) = P(A)P(B|A)P(C|B)P(D|C)$ $P(D) = \sum_A P(A) \sum_B P(B|A) \sum_C P(B|C)P(D|C)$

When evaluating $P(D)$, other marginals can be computed 'on the way there': we know $P(A)$,

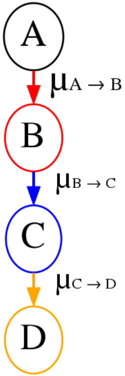


$$
\begin{aligned}
P(B) &= \sum_A P(B|A)P(A) \\
P(C) &= \sum_B P(C|B) \sum_A P(B|A)P(A) \\
&= \sum_B P(C|B)P(B) \\
P(D) &= \sum_C P(D|C) \sum_B P(C|B) \sum_A P(B|A)P(A) \\
&= \sum_C P(D|C)P(C)
\end{aligned}
$$

Message-passing interpretation

Reminder:
$P(B) = \sum_A P(A)P(B|A)$
$P(C) = \sum_A P(A) \sum_B P(B|A)P(C|B) = \sum_B P(B)P(C|B)$
$P(D) = \sum_A P(A) \sum_B P(B|A) \sum_C P(C|B)P(D|C) = \sum_C P(D|C)P(C)$

The evaluation of $P(D)$ can be interpreted as node-local computation and message passing between the nodes:



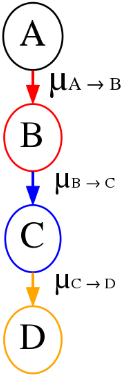
Let $\mu_{X \to Y}$ be the message sent from node X to Y. The locally available information at Y is the conditional $P(Y|X)$. Then

$$
\begin{aligned}
\mu_{A \to B} &:= P(A) \\
\mu_{B \to C} &:= \sum_A \mu_{A \to B} P(B|A) = P(B) \\
\mu_{C \to D} &:= \sum_B \mu_{B \to C} P(C|B) = P(C) \\
P(D) &= \sum_C \mu_{C \to D} P(D|C)
\end{aligned}
$$

Message-passing interpretation

Reminder:
$P(B) = \sum_A P(A)P(B|A)$
$P(C) = \sum_A P(A) \sum_B P(B|A)P(C|B) = \sum_B P(B)P(C|B)$
$P(D) = \sum_A P(A) \sum_B P(B|A) \sum_C P(C|B)P(D|C) = \sum_C P(D|C)P(C)$

The evaluation of $P(D)$ can be interpreted as node-local computation and message passing between the nodes:



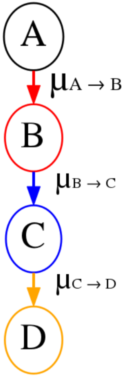
Let $\mu_{X \to Y}$ be the message sent from node X to Y. The locally available information at Y is the conditional $P(Y|X)$. Then

$$
\begin{aligned}
\mu_{A \to B} &:= P(A) \\
\mu_{B \to C} &:= \sum_A \mu_{A \to B} P(B|A) = P(B) \\
\mu_{C \to D} &:= \sum_B \mu_{B \to C} P(C|B) = P(C) \\
P(D) &= \sum_C \mu_{C \to D} P(D|C)
\end{aligned}
$$

Message-passing interpretation

Reminder:
$P(B) = \sum_A P(A)P(B|A)$
$P(C) = \sum_A P(A) \sum_B P(B|A)P(C|B) = \sum_B P(B)P(C|B)$
$P(D) = \sum_A P(A) \sum_B P(B|A) \sum_C P(C|B)P(D|C) = \sum_C P(D|C)P(C)$

The evaluation of $P(D)$ can be interpreted as node-local computation and message passing between the nodes:



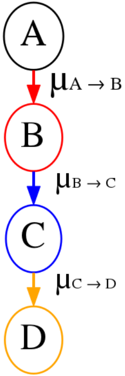
Let $\mu_{X \to Y}$ be the message sent from node X to Y. The locally available information at Y is the conditional $P(Y|X)$. Then

$$
\begin{aligned}
\mu_{A \to B} &:= P(A) \\
\mu_{B \to C} &:= \sum_A \mu_{A \to B} P(B|A) = P(B) \\
\mu_{C \to D} &:= \sum_B \mu_{B \to C} P(C|B) = P(C) \\
P(D) &= \sum_C \mu_{C \to D} P(D|C)
\end{aligned}
$$

Message-passing interpretation

Reminder:
pa_X: parent(s) of node X.
$\mu_{X \to Y}$ is the message passed from X to Y. Locally available information at X: $P(X|\mathrm{pa}_X)$.

The evaluation of $P(D)$ can be interpreted as node-local computation and message passing between the nodes:



Let ch_X be the child of X.
Message-passing scheme:

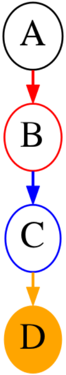
$$\mu_{X \to \mathrm{ch}_X} := \sum_{\mathrm{pa}_X} \mu_{\mathrm{pa}_X \to X} P(X|\mathrm{pa}_X) = P(X)$$

$\Rightarrow$ to evaluate node marginals, messages are passed *forward* along the graph.

Computing joint probabilities for conditionals

Reminder:
$P(A, B, C, D, E) = P(A)P(B|A)P(C|B)P(D|C)$
$\mu_{X \to Y} := \sum_{\mathrm{pa}_X} \mu_{\mathrm{pa}_X \to X} P(X|\mathrm{pa}_X) = P(X)$

Assume $D = d$ is observed. For inference, we need conditionals like $P(C|D = d) = \frac{P(C,d)}{P(d)}$ etc.



$$P(C,d) = \underbrace{P(d|C)}_{:=\mu_{C\leftarrow D}} \underbrace{\sum_B P(C|B) \sum_A P(B|A)P(A)}_{\mu_{C\to D}=P(C)}$$

$$P(B,d) = \underbrace{\sum_C P(d|C)P(C|B)}_{:=\mu_{B\leftarrow C}} \underbrace{\sum_A P(B|A)P(A)}_{\mu_{B\to C}=P(B)}$$

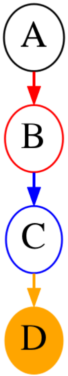
where

$$\mu_{B\leftarrow C} = \sum_C P(C|B)\mu_{C\leftarrow D}$$

Computing joint probabilities for conditionals

Reminder:
$P(A, B, C, D, E) = P(A)P(B|A)P(C|B)P(D|C)$
$\mu_{X \to Y} := \sum_{\mathrm{pa}_X} \mu_{\mathrm{pa}_X \to X} P(X|\mathrm{pa}_X) = P(X)$

Assume $D = d$ is observed. For inference, we need conditionals like $P(C|D = d) = \frac{P(C,d)}{P(d)}$ etc.



$$P(C,d) = \underbrace{P(d|C)}_{:=\mu_{C \leftarrow D}} \underbrace{\sum_B P(C|B) \sum_A P(B|A)P(A)}_{\mu_{C \to D} = P(C)}$$

$$P(B,d) = \underbrace{\sum_C P(d|C)P(C|B)}_{:=\mu_{B \leftarrow C}} \underbrace{\sum_A P(B|A)P(A)}_{\mu_{B \to C} = P(B)}$$

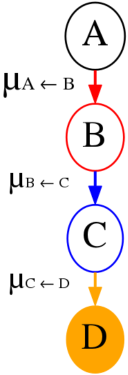
where

$$\mu_{B \leftarrow C} = \sum_C P(C|B)\mu_{C \leftarrow D}$$

Computing joint probabilities by backward message passing

Reminder:
$P(A,B,C,D,E) = P(A)P(B|A)P(C|B)P(D|C)$
$\mu_{X\to Y} := \sum_{\mathrm{pa}_X} \mu_{\mathrm{pa}_X \to X} P(X|\mathrm{pa}_X) = P(X)$
$P(C,d) = \mu_{C\to D} P(d|C)$
$P(B,d) = \mu_{B\to C} \sum_C P(C|B)P(d|C)$ ch_Y is the child of node Y, pa_Y is the parent. *Leaf nodes* have no child(ren).

We want to evaluate $P(C,d)$ etc. for conditioning.



Let $L = l$ the observation of the leaf node.
Backwards message-passing scheme:

$$\mu_{\mathrm{pa}_X \leftarrow X} := \sum_X P(X|\mathrm{pa}_X)\mu_{X\leftarrow \mathrm{ch}_X}$$

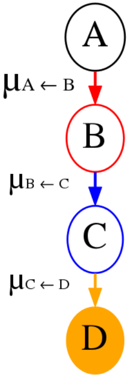
$\Rightarrow$ to evaluate the joint probabilities, messages are passed *backwards* along the graph, then compute

$$P(X, L = l) = \mu_{X\to \mathrm{ch}_X} \times \mu_{X\leftarrow \mathrm{ch}_X}$$

Computing joint probabilities by backward message passing

Reminder:
$P(A,B,C,D,E) = P(A)P(B|A)P(C|B)P(D|C)$
$\mu_{X\to Y} := \sum_{\mathrm{pa}_X} \mu_{\mathrm{pa}_X \to X} P(X|\mathrm{pa}_X) = P(X)$
$P(C,d) = \mu_{C\to D} P(d|C)$
$P(B,d) = \mu_{B\to C} \sum_C P(C|B)P(d|C)$ ch_Y is the child of node Y, pa_Y is the parent. *Leaf nodes* have no child(ren).

We want to evaluate $P(C,d)$ etc. for conditioning.



Let $L = l$ the observation of the leaf node.
Backwards message-passing scheme:

$$\mu_{\mathrm{pa}_X \leftarrow X} := \sum_X P(X|\mathrm{pa}_X)\mu_{X\leftarrow \mathrm{ch}_X}$$

$\Rightarrow$ to evaluate the joint probabilities, messages are passed *backwards* along the graph, then compute

$$P(X, L=l) = \mu_{X\to \mathrm{ch}_X} \times \mu_{X\leftarrow \mathrm{ch}_X}$$

Summary: conditioning on the leaf node of a (Markov) chain

- The message-passing scheme which we have derived is a.k.a. the *forward-backward* algorithm for Markov chains.
- Messages are computed from *locally available* information.
- Forward messages:

$$\mu_{X \to \mathrm{ch}_X} := \sum_{\mathrm{pa}_X} \mu_{\mathrm{pa}_X \to X} P(X|\mathrm{pa}_X)$$

  - The root node R sends $\mu_{R \to \mathrm{ch}_R} = P(R)$.
- Backward message:

$$\mu_{\mathrm{pa}_X \leftarrow X} := \sum_{X} P(X|\mathrm{pa}_X) \mu_{X \leftarrow \mathrm{ch}_X}$$

  - The leaf node L sends $\mu_{\mathrm{pa}_L \leftarrow L} = P(l|\mathrm{pa}_L)$.

Summary: conditioning on the leaf node of a (Markov) chain

- The message-passing scheme which we have derived is a.k.a. the *forward-backward* algorithm for Markov chains.
- Messages are computed from *locally available* information.
- Forward messages:

$$\mu_{X \to \mathrm{ch}_X} := \sum_{\mathrm{pa}_X} \mu_{\mathrm{pa}_X \to X} P(X|\mathrm{pa}_X)$$

  - The root node R sends $\mu_{R \to \mathrm{ch}_R} = P(R)$.
- Backward message:

$$\mu_{\mathrm{pa}_X \leftarrow X} := \sum_{X} P(X|\mathrm{pa}_X) \mu_{X \leftarrow \mathrm{ch}_X}$$

  - The leaf node L sends $\mu_{\mathrm{pa}_L \leftarrow L} = P(l|\mathrm{pa}_L)$.

Summary: conditioning on the leaf node of a (Markov) chain

- Marginals $P(X) = \mu_{X \to \mathrm{ch}_X}$
- Joint $P(X, L = l) = \mu_{X \to \mathrm{ch}_X} \times \mu_{X \leftarrow \mathrm{ch}_X}$
- All information available at all nodes after 2 passes through the chain!

Generalization to non-chain structured graphs

- We will now generalize the message-passing scheme to more general graphs.
- This can be done by transforming a Bayesian network into a *factor graph*.
- A factor graph is useful for representing functions which can be decomposed into factors.
  - Each factor depends on a subset of the arguments of the function.
- Facilitates the exploitation of distributivity.
- Marginalizations are done with the **sum-product** algorithm.
- Examples: Bayesian networks, Markov random fields, Markov chains, Hidden Markov models, Viterbi algorithm, Turbo codes, Fast Fourier transform etc....

Generalization to non-chain structured graphs

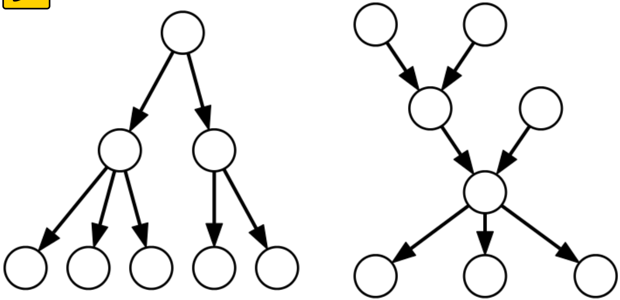
- We will now generalize the message-passing scheme to more general graphs.
- This can be done by transforming a Bayesian network into a *factor graph*.
- A factor graph is useful for representing functions which can be decomposed into factors.
  - Each factor depends on a subset of the arguments of the function.
- Facilitates the exploitation of distributivity.
- Marginalizations are done with the **sum-product** algorithm.
- Examples: Bayesian networks, Markov random fields, Markov chains, Hidden Markov models, Viterbi algorithm, Turbo codes, Fast Fourier transform etc....

Generalization to non-chain structured graphs

- We will now generalize the message-passing scheme to more general graphs.
- This can be done by transforming a Bayesian network into a *factor graph*.
- A factor graph is useful for representing functions which can be decomposed into factors.
  - Each factor depends on a subset of the arguments of the function.
- Facilitates the exploitation of distributivity.
- Marginalizations are done with the **sum-product** algorithm.
- Examples: Bayesian networks, Markov random fields, Markov chains, Hidden Markov models, Viterbi algorithm, Turbo codes, Fast Fourier transform etc....

Trees and polytrees

Question: which types of Bayesian networks can be marginalized efficiently (after conversion to a factor graph)? **Answer**: Singly connected graphs: exactly one path between any pair of nodes (ignoring the arrows). Trees and polytrees.



Factor graphs

Let $\mathbf{X} = \{X_1, \ldots, X_N\}$ be the arguments of a function $q(X_1, \ldots, X_N)$. Let the sets $\mathbf{s}_1, \ldots, \mathbf{s}_K$ be subsets of $\mathbf{X}$, and the functions $f_1, \ldots, f_K$ depend only on the arguments in the corresponding $\mathbf{s}_i$. The f_i are a factorization of q if

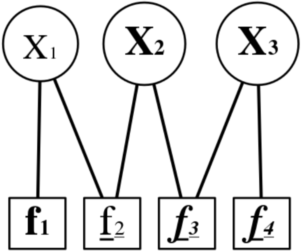
$$q(\mathbf{X}) = \prod_{i=1}^{K} f_i(\mathbf{s}_i)$$

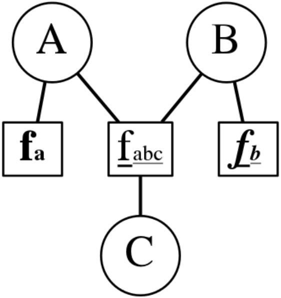
To construct a factor graph from a factorization,

- draw one variable node (circle) for each variable X_j,
- draw one factor node (square) for each factor f_i,
- if $j \in s_i$, connect the variable node of X_j with the factor node of f_i.

Example: factor graphs

$f_1(X_1)\, f_2(X_1, X_2)\, f_3(X_2, X_3)\, f_4(X_3)$ $\quad$ $f_a(A)\, f_b(B)\, f_{abc}(A, B, C)$

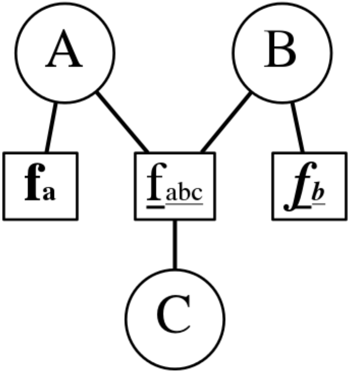




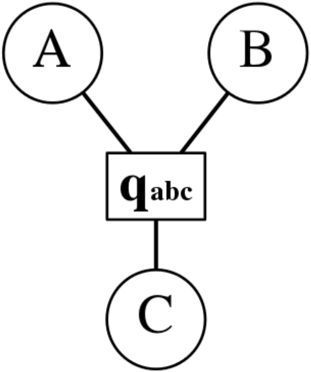
Factor graphs are *bipartite*: factors are only connected to variables, and vice versa!

Factorizations are not unique

$$f_a(A)\, f_b(B)\, f_{abc}(A, B, C)$$



$$q_{abc}(A, B, C) =$$
$$= f_a(A)\, f_b(B)\, f_{abc}(A, B, C)$$

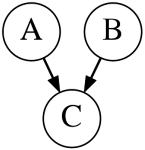


Translating Bayesian Nets into factor graphs

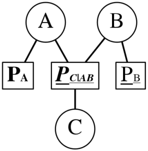
Bayesian network	**factor graph**
Node	Variable node
(Un)conditional distribution	Factor node
Origin node of directed edge	Edge from variable to factor
Target node of directed edge	Edge from factor to variable

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

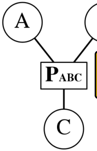
Bay. net



A factor graph



Another factor graph



Marginalization on factor graphs

Reminder:
$\mathbf{X} = X_1, \ldots, X_N$ are the arguments of $q(X_1, \ldots, X_N)$.
$\mathbf{s}_1, \ldots, \mathbf{s}_K \subseteq \mathbf{X}$, functions $f_1, \ldots, f_K$ depend only on the arguments in the corresponding $\mathbf{s}_i$.
$q(\mathbf{X}) = \prod_{i=1}^{K} f_i(\mathbf{s}_i)$

Given a factorization of a function

$$q(\mathbf{X}) = \prod_i f_i(\mathbf{s}_i)$$

we want to

- obtain an efficient, exact algorithm for computing marginals.
- If several marginals are desired, re-use as much of the computation as possible.
- We would also prefer a local algorithm (to keep the calculations simpler).

Marginalization on factor graphs, contd.

Reminder:
$\mathbf{X} = X_1, \ldots, X_N$ are the arguments of $q(X_1, \ldots, X_N)$.
$\mathbf{s}_1, \ldots, \mathbf{s}_K \subseteq \mathbf{X}$, functions $f_1, \ldots, f_K$ depend only on the arguments in the corresponding $\mathbf{s}_i$.
$q(\mathbf{X}) = \prod_{i=1}^{K} f_i(\mathbf{s}_i)$

Assume we wanted to compute the marginal (function) of $Y \in \mathbf{X}$.
The marginal will be evaluated at the corresponding variable node:

$$f(Y) = \sum_{\sim Y} q(\mathbf{X}) = \sum_{\sim Y} \prod_{i} f_i(\mathbf{s}_i)$$

Because the graph is singly connected, we can divide the factors f_i into *disjoint* sets according to which neighbouring factor node of Y they appear in or connect to.
Likewise, we can divide the variables into *disjoint* sets sets according to which neighbouring factor node of Y they appear in or connect to.

Marginalization on factor graphs, contd.

Reminder:
$\mathbf{X} = X_1, \ldots, X_N$ are the arguments of $q(X_1, \ldots, X_N)$.
$\mathbf{s}_1, \ldots, \mathbf{s}_K \subseteq \mathbf{X}$, functions $f_1, \ldots, f_K$ depend only on the arguments in the corresponding $\mathbf{s}_i$.
$q(\mathbf{X}) = \prod_{i=1}^{K} f_i(\mathbf{s}_i)$

Assume we wanted to compute the marginal (function) of $Y \in \mathbf{X}$. The marginal will be evaluated at the corresponding variable node:

$$f(Y) = \sum_{\sim Y} q(\mathbf{X}) = \sum_{\sim Y} \prod_i f_i(\mathbf{s}_i)$$

Because the graph is singly connected, we can divide the factors f_i into *disjoint* sets according to which neighbouring factor node of Y they appear in or connect to.
Likewise, we can divide the variables into *disjoint* sets sets according to which neighbouring factor node of Y they appear in or connect to.

Marginalization on factor graphs, contd.

Reminder:
$\mathbf{X} = X_1, \ldots, X_N$ are the arguments of $q(X_1, \ldots, X_N)$.
$\mathbf{s}_1, \ldots, \mathbf{s}_K \subseteq \mathbf{X}$, functions $f_1, \ldots, f_K$ depend only on the arguments in the corresponding $\mathbf{s}_i$.
$q(\mathbf{X}) = \prod_{i=1}^{K} f_i(\mathbf{s}_i)$

Assume we wanted to compute the marginal (function) of $Y \in \mathbf{X}$.
The marginal will be evaluated at the corresponding variable node:

$$f(Y) = \sum_{\sim Y} q(\mathbf{X}) = \sum_{\sim Y} \prod_i f_i(\mathbf{s}_i)$$

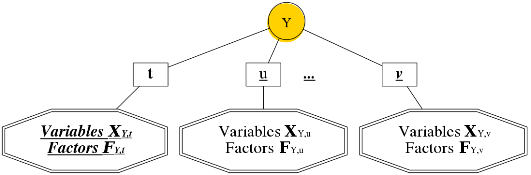
Because the graph is singly connected, we can divide the factors f_i into *disjoint* sets according to which neighbouring factor node of Y they appear in or connect to.
Likewise, we can divide the variables into *disjoint* sets sets according to which neighbouring factor node of Y they appear in or connect to.

Dividing factors and variables into disjoint sets

Reminder:
We want to compute $f(Y) = \sum_{\sim Y} q(\mathbf{X}) = \sum_{\sim Y} \prod_i f_i(\mathbf{s}_i)$ at the node of Y.

Let

- ne_Y be the set of factor nodes neighbouring Y,
- $\mathbf{X}_{Y,t}$ and $\mathbf{F}_{Y,t}$ be the subset of variables and factors which connect to Y via factor node t,
- $F_t(Y, \mathbf{X}_{Y,t}) = \prod_{f \in \mathbf{F}_{Y,t}} f(\mathbf{s}_f)$, where $\mathbf{s}_f \subseteq \{Y\} \cup \mathbf{X}_{Y,t}$.

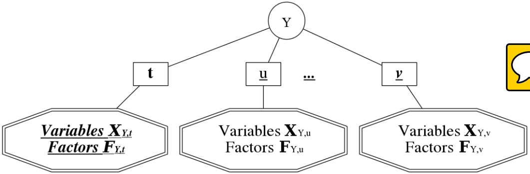


Dividing factors and variables into disjoint sets, contd.

Reminder:
We want to compute $f(Y) = \sum_{\sim Y} q(\mathbf{X}) = \sum_{\sim Y} \prod_i f_i(\mathbf{s}_i)$ at the node of Y.
ne_Y is the set of factor nodes neighbouring variable node Y.
$\mathbf{X}_{Y,t}$ and $\mathbf{F}_{Y,t}$ be the subset of variables and factors which connect to Y via factor node t,
$F_t(Y, \mathbf{X}_{Y,t}) = \prod_{f \in \mathbf{F}_{Y,t}} f(\mathbf{s}_f)$, where $\mathbf{s}_f \subseteq \{Y\} \cup \mathbf{X}_{Y,t}$.

Because the graph is singly connected (i.e. an undirected tree with Y as a possible root):

- $\forall t, u \in \mathrm{ne}_Y : t \neq u \Rightarrow \mathbf{X}_{Y,t} \cap \mathbf{X}_{Y,u} = \emptyset$ and $\mathbf{F}_{Y,t} \cap \mathbf{F}_{Y,u} = \emptyset$.
- $\forall t, u \in \mathrm{ne}_Y, X \in \mathbf{X}_{Y,t} : t \neq u \Rightarrow X \notin \mathbf{X}_{Y,u}$,
  i.e. $F_u(Y, \mathbf{X}_{Y,u})$ does not depend on $X \in \mathbf{X}_{Y,t}$.
- $Y \cup \bigcup_{t \in \mathrm{ne}_Y} \mathbf{X}_{Y,t} = \mathbf{X}$ and $q(\mathbf{X}) = \prod_{t \in \mathrm{ne}_Y} F_t(Y, \mathbf{X}_{Y,t})$



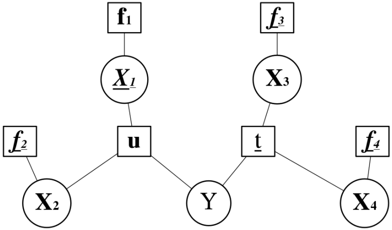
Example: decomposing the product by neighbours

Reminder:

ne_Y are the (factor) neighbours of (variable) node Y.
$\mathbf{X}_{Y,t}$ are variables (except Y) which connect to Y via factor node $t \in \mathrm{ne}_Y$.
$\forall t \in \mathrm{ne}_Y : F_t(Y, \mathbf{X}_{Y,t})$ product of all factors which connect to Y via t.

Example:

- $\mathbf{X} = \{X_1, \ldots, X_4, Y\}$, $q(\mathbf{X}) = t(X_3, X_4, Y)u(X_1, X_2, Y) \prod_i f_i(X_i)$.
- $\mathrm{ne}_Y = \{t, u\}$, $\mathbf{X}_{Y,t} = \{X_3, X_4\}$, $\mathbf{X}_{Y,u} = \{X_1, X_2\}$.
- $F_t(Y, X_3, X_4) = f_3(X_3)f_4(X_4)t(X_3, X_4, Y)$.
- $F_u(Y, X_1, X_2) = f_1(X_1)f_2(X_2)u(X_1, X_1, Y)$.
- $q(\mathbf{X}) = F_t(Y, X_3, X_4)F_u(Y, X_1, X_2)$



Sending messages from factors to variables

Reminder:
ne_Y are the (factor) neighbours of (variable) node Y.
$\mathbf{X}_{Y,t}$ are variables which connect to Y via factor node $t \in \mathrm{ne}_Y$.
$\forall t \in \mathrm{ne}_Y : F_t(Y, \mathbf{X}_{Y,t})$ product of all factors which connect to Y via t.

Let $\mathrm{ne}_Y = \{t_1, \ldots, t_N\}$. Rewrite the sum over all other variables and use distributivity:

$$\begin{aligned} f(Y) &= \sum_{\sim Y} \prod_{t \in \mathrm{ne}_Y} F_t(Y, \mathbf{X}_{Y,t}) \\ &= \sum_{X \in \mathbf{X}_{Y,t_1}} \cdots \sum_{X \in \mathbf{X}_{Y,t_N}} \prod_{t \in \mathrm{ne}_Y} F_t(Y, \mathbf{X}_{Y,t}) \\ &= \prod_{t \in \mathrm{ne}_Y} \left[\sum_{\mathbf{X}_{Y,t}} F_t(Y, \mathbf{X}_{Y,t}) \right] =: \prod_{t \in \mathrm{ne}_Y} \mu_{t \to Y}(Y) \end{aligned}$$

Definition: message from factor node t to variable node Y is

$$\mu_{t \to Y}(Y) := \sum_{\mathbf{X}_{Y,t}} F_t(Y, \mathbf{X}_{Y,t})$$

Sending messages from variables to factors

Reminder:
ne_Y are the (factor) neighbours of (variable) node Y.
$\mathbf{X}_{Y,t}$ are the variables which connect to Y through factor node $t \in \mathrm{ne}_Y$.
$\forall t \in \mathrm{ne}_Y : F_t(Y, \mathbf{X}_{Y,t})$ product of all factors which connect to Y via t.
$\mu_{t\to Y}(Y) := \sum_{\mathbf{X}_{Y,t}} F_t(Y, \mathbf{X}_{Y,t})$ is the message from $t \in \mathrm{ne}_Y$ to Y.

So far: $f(Y) = \prod_{t\in \mathrm{ne}_Y} \mu_{t\to Y}(Y)$.
Question: how do we compute the messages $\mu_{t\to Y}(Y)$?
Answer: the $F_t(Y, \mathbf{X}_{Y,t})$ are factor (sub)graphs, which can be decomposed.

- We now look at the graph with t as the root.
- Denote $\mathrm{ne}_t = \{Y, Z_1, \ldots, Z_M\}$ the (variable) neighbours of t.
- Let $\mathbf{X}_{t,Z}$ be the set of all variables which are connected to t via Z.
- Let $\mathbf{F}_{t,Z}$ be the set of all factors which are connected to t via Z.

Sending messages from variables to factors

Reminder:
ne_Y are the (factor) neighbours of (variable) node Y.
$\mathbf{X}_{Y,t}$ are the variables which connect to Y through factor node $t \in \mathrm{ne}_Y$.
$\forall t \in \mathrm{ne}_Y : F_t(Y, \mathbf{X}_{Y,t})$ product of all factors which connect to Y via t.
$\mu_{t \to Y}(Y) := \sum_{\mathbf{X}_{Y,t}} F_t(Y, \mathbf{X}_{Y,t})$ is the message from $t \in \mathrm{ne}_Y$ to Y.

So far: $f(Y) = \prod_{t \in \mathrm{ne}_Y} \mu_{t \to Y}(Y)$.
Question: how do we compute the messages $\mu_{t \to Y}(Y)$?
Answer: the $F_t(Y, \mathbf{X}_{Y,t})$ are factor (sub)graphs, which can be decomposed.

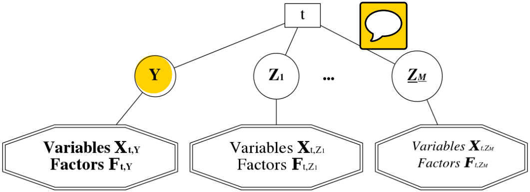
- We now look at the graph with t as the root.
- Denote $\mathrm{ne}_t = \{Y, Z_1, \ldots, Z_M\}$ the (variable) neighbours of t.
- Let $\mathbf{X}_{t,Z}$ be the set of all variables which are connected to t via Z.
- Let $\mathbf{F}_{t,Z}$ be the set of all factors which are connected to t via Z.

Sending messages from variables to factors, contd.

Reminder:
$ne_t = \{Y, Z_1, \ldots, Z_M\}$ are the (variable) neighbours of (factor) node t.
$\mathbf{X}_{t,Z}$ and $\mathbf{F}_{t,Z}$ are the sets of all variables and factors which are connected to t via Z.
$\mu_{t\to Y}(Y) := \sum_{\mathbf{X}_{Y,t}} F_t(Y, \mathbf{X}_{Y,t})$ is the message from $t \in ne_Y$ to Y.
$F_t(Y, \mathbf{X}_{Y,t})$ product of all factors which connect to Y via t.

Let $G_Z(Z, \mathbf{X}_{t,Z}) = \prod_{f\in \mathbf{F}_{t,Z}} f(Z, \mathbf{X}_{t,Z})$ be the product of all factors that connect to t via Z. Then

$$F_t(Y, \mathbf{X}_{Y,t}) = t(Y, Z_1, \ldots, Z_M) G_{Z_1}(Z_1, \mathbf{X}_{t,Z_1}) \ldots G_{Z_M}(Z_M, \mathbf{X}_{t,Z_M})$$



Sending messages from variables to factors contd.

Reminder:

ne_Y are the (factor) neighbours of (variable) node Y. ne_t are the (variable) neighbours of (factor) node t.
$\mathbf{X}_{Y,t}$ are the variables which connect to Y via factor node $t \in \mathrm{ne}_Y$.
$\mathbf{X}_{t,Z}$ are the variables which connect to t via Z.
$\forall t \in \mathrm{ne}_Y : F_t(Y, \mathbf{X}_{Y,t})$ product of all factors which connect to Y via t.
$\mu_{t \to Y}(Y) := \sum_{\mathbf{X}_{Y,t}} F_t(Y, \mathbf{X}_{Y,t})$ is the message from $t \in \mathrm{ne}_Y$ to Y.
Product of all factors that connect to t via Z: $G_Z(Z, \mathbf{X}_{t,Z}) = \prod_{f \in \mathbf{F}_{t,Z}} f(Z, \mathbf{X}_{t,Z})$.

Thus we can write the messages from factors to variables as:

$$
\begin{aligned}
\mu_{t \to Y}(Y) &= \sum_{\mathbf{X}_{Y,t}} F_t(Y, \mathbf{X}_{Y,t}) \\
&= \sum_{Z_1} \cdots \sum_{Z_M} t(Y, Z_1, \ldots, Z_M) \prod_{Z \in \mathrm{ne}_t \setminus Y} \left[\sum_{\mathbf{X}_{t,Z}} G_Z(Z, \mathbf{X}_{t,Z}) \right] \\
&= \sum_{Z_1} \cdots \sum_{Z_M} t(Y, Z_1, \ldots, Z_M) \prod_{Z \in \mathrm{ne}_t \setminus Y} \mu_{Z \to t}(Z)
\end{aligned}
$$

where we have defined the message from variable Z to factor t as

$$\mu_{Z \to t}(Z) := \sum_{\mathbf{X}_{t,Z}} G_Z(Z, \mathbf{X}_{t,Z}).$$

Sending messages from factors to variables

Reminder:
$\mathbf{X}_{t,Z}$ are the variables which connect to t via Z.
Product of all factors that connect to t via Z: $G_Z(Z, \mathbf{X}_{t,Z}) = \prod_{f \in \mathbf{F}_{t,Z}} f(Z, \mathbf{X}_{t,Z})$.

We found:

$$\mu_{Z \to t}(Z) := \sum_{\mathbf{X}_{t,Z}} G_Z(Z, \mathbf{X}_{t,Z}).$$

Question: how do we compute $G_Z(Z, \mathbf{X}_{t,Z})$?
Answer: $G_Z(Z, \mathbf{X}_{t,Z})$ can be decomposed into factors which can be represented by messages from factors to variables:

$$G_Z(Z, \mathbf{X}_{t,Z}) = \prod_{l \in \mathrm{ne}_Z \setminus t} F_l(Z, \mathbf{X}_{Z,l})$$

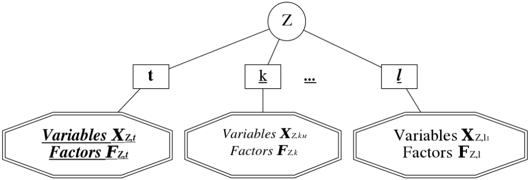
where $\mathbf{X}_{Z,l}$ contains all variables connected to Z through factor l.

Sending messages from factors to variables, contd.

Reminder:
Product of all factors that connect to t via Z: $G_Z(Z, \mathbf{X}_{t,Z}) = \prod_{f \in \mathbf{F}_{t,Z}} f(Z, \mathbf{X}_{t,Z})$.

$F_l(Z, \mathbf{X}_{Z,l})$ is the product of all factors which connect to Z via l.
We want to evaluate:

$$G_Z(Z, \mathbf{X}_{t,Z}) = \prod_{l \in \mathrm{ne}_Z \setminus t} F_l(Z, \mathbf{X}_{Z,l})$$



Sending messages from factors to variables, contd.

Reminder:
$\mathbf{X}_{t,Z}$ contains all variables connected to t via Z.
$\mathbf{X}_{Z,l}$ contains all variables connected to Z through factor l.
$\mu_{l\to Z}(Z) := \sum_{\mathbf{X}_{Z,l}} F_l(Z, \mathbf{X}_{Z,l})$
Product of all factors that connect to t via Z: $G_Z(Z, \mathbf{X}_{t,Z}) = \prod_{f\in \mathbf{F}_{t,Z}} f(Z, \mathbf{X}_{t,Z})$.

Thus we find for the messages from variables to factors:

$$
\begin{aligned}
\mu_{Z\to t}(Z) &= \sum_{\mathbf{X}_{t,Z}} G_Z(Z, \mathbf{X}_{t,Z}) \\
&= \sum_{\mathbf{X}_{t,Z}} \prod_{l\in \mathrm{ne}_Z\setminus t} F_l(Z, \mathbf{X}_{Z,l}) \\
&= \prod_{l\in \mathrm{ne}_Z\setminus t} \sum_{\mathbf{X}_{Z,l}} F_l(Z, \mathbf{X}_{Z,l}) \\
&= \prod_{l\in \mathrm{ne}_Z\setminus t} \mu_{l\to Z}(Z)
\end{aligned}
$$

$\Rightarrow$ to send a message to a factor, multiply incoming messages from other factors.

Starting the message-passing iteration

Reminder:
Message from factor t to variable Y: $\mu_{t \to Y}(Y) = \sum_{Z_1} \dots \sum_{Z_M} t(Y, Z_1, \dots, Z_M) \prod_{Z \in \mathrm{ne}_t \setminus Y} \mu_{Z \to t}(Z)$
Message from variable Y to factor t: $\mu_{Y \to t}(Y) = \prod_{l \in \mathrm{ne}_Y \setminus t} \mu_{l \to Y}(Y)$

- To send a message from a variable Y to a factor t, we need the messages sent to Y from all $\mathrm{ne}_Y \setminus \{t\}$.
- To send a message from a factor t to a variable Y, we need the messages sent to t from all $\mathrm{ne}_t \setminus \{Y\}$.

Question: where do we start the message-passing iteration?

Answer: at leaf nodes, since they do not need to wait for incoming messages.

Starting the message-passing iteration

Reminder:
Message from factor t to variable Y: $\mu_{t \to Y}(Y) = \sum_{Z_1} \dots \sum_{Z_M} t(Y, Z_1, \dots, Z_M) \prod_{Z \in \mathrm{ne}_t \setminus Y} \mu_{Z \to t}(Z)$
Message from variable Y to factor t: $\mu_{Y \to t}(Y) = \prod_{l \in \mathrm{ne}_Y \setminus t} \mu_{l \to Y}(Y)$

- To send a message from a variable Y to a factor t, we need the messages sent to Y from all $\mathrm{ne}_Y \setminus \{t\}$.
- To send a message from a factor t to a variable Y, we need the messages sent to t from all $\mathrm{ne}_t \setminus \{Y\}$.

Question: where do we start the message-passing iteration?
Answer: at leaf nodes, since they do not need to wait for incoming messages.

Messages from leaf nodes

Reminder:
Message from factor t to variable Y: $\mu_{t \to Y}(Y) = \sum_{Z_1} \dots \sum_{Z_M} t(Y, Z_1, \dots, Z_M) \prod_{Z \in \mathrm{ne}_t \setminus Y} \mu_{Z \to t}(Z)$
Message from variable Y to factor t: $\mu_{Y \to t}(Y) = \prod_{l \in \mathrm{ne}_Y \setminus t} \mu_{l \to Y}(Y)$

Generally, messages from factors t to variables Y are given by

$$\mu_{t \to Y}(Y) = \sum_{Z_1} \dots \sum_{Z_M} t(Y, Z_1, \dots, Z_M) \prod_{Z \in \mathrm{ne}_t \setminus Y} \mu_{Z \to t}(Z)$$

where the $Z_i \in \mathrm{ne}_t \setminus Y$. If t is a factor leaf node, then it has (at most) one neighbour, say variable Y. Thus $\mathrm{ne}_t \setminus Y = \emptyset$ and hence

$$\mu_{t \to Y}(Y) = t(Y).$$

Likewise, if V is a variable leaf node, with V appearing only in factor r:

$$\mu_{V \to r}(V) = \prod_{l \in \mathrm{ne}_V \setminus r} \mu_{l \to V}(V) = 1$$

Messages from leaf nodes

Reminder:
Message from factor t to variable Y: $\mu_{t\to Y}(Y) = \sum_{Z_1} \dots \sum_{Z_M} t(Y, Z_1, \dots, Z_M) \prod_{Z \in \mathrm{ne}_t \setminus Y} \mu_{Z\to t}(Z)$
Message from variable Y to factor t: $\mu_{Y\to t}(Y) = \prod_{l \in \mathrm{ne}_Y \setminus t} \mu_{l\to Y}(Y)$

Generally, messages from factors t to variables Y are given by

$$\mu_{t\to Y}(Y) = \sum_{Z_1} \dots \sum_{Z_M} t(Y, Z_1, \dots, Z_M) \prod_{Z \in \mathrm{ne}_t \setminus Y} \mu_{Z\to t}(Z)$$

where the $Z_i \in \mathrm{ne}_t \setminus Y$. If t is a factor leaf node, then it has (at most) one neighbour, say variable Y. Thus $\mathrm{ne}_t \setminus Y = \emptyset$ and hence

$$\mu_{t\to Y}(Y) = t(Y).$$

Likewise, if V is a variable leaf node, with V appearing only in factor r:

$$\mu_{V\to r}(V) = \prod_{l \in \mathrm{ne}_V \setminus r} \mu_{l\to V}(V) = 1$$

Summary: sum-product algorithm

- Message from factor t to variable Y:

$$\mu_{t\to Y}(Y) = \sum_{Z_1} \dots \sum_{Z_M} t(Y, Z_1, \dots, Z_M) \prod_{Z \in \mathrm{ne}_t \setminus Y} \mu_{Z\to t}(Z)$$

- Message from variable Y to factor t:

$$\mu_{Y\to t}(Y) = \prod_{l \in \mathrm{ne}_Y \setminus t} \mu_{l\to Y}(Y)$$

- Message from a factor leaf node f to variable Y: $\mu_{f\to Y}(Y) = f(Y)$
- Message from a variable leaf node Z to factor t: $\mu_{Z\to t}(Z) = 1$
- Marginal of Y:

$$f(Y) = \prod_{t \in \mathrm{ne}_Y} \mu_{t\to Y}(Y)$$

.