Artificial Intelligence

Belief Propagation and Junction Trees

Andres Mendez-Vazquez

March 28, 2016

Outline

Outline

Cinvestav

Introduction

We will be looking at the following algorithms

- Pearl's Belief Propagation Algorithm
- Junction Tree Algorithm

Introduction

We will be looking at the following algorithms

- Pearl's Belief Propagation Algorithm
- Junction Tree Algorithm

Belief Propagation Algorithm

- The algorithm was first proposed by Judea Pearl in 1982, who formulated this algorithm on trees, and was later extended to polytrees.

Cinvestav

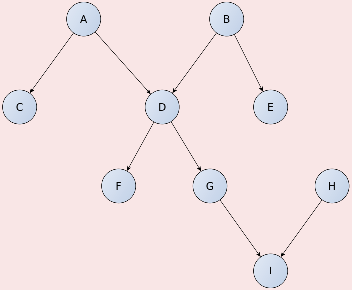
Introduction

We will be looking at the following algorithms

- Pearl's Belief Propagation Algorithm
- Junction Tree Algorithm

Belief Propagation Algorithm

- The algorithm was first proposed by Judea Pearl in 1982, who formulated this algorithm on trees, and was later extended to polytrees.



Introduction

Something Notable

- It has since been shown to be a useful approximate algorithm on general graphs.

Junction Tree Algorithm

- The junction tree algorithm (also known as 'Clique Tree') is a method used in machine learning to extract marginalization in general graphs.

Introduction

Something Notable

- It has since been shown to be a useful approximate algorithm on general graphs.

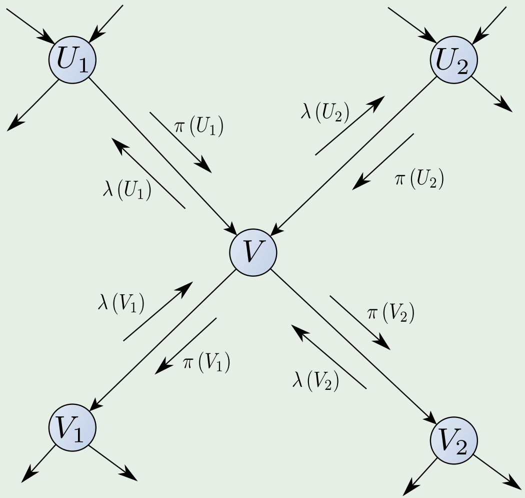
Junction Tree Algorithm

- The junction tree algorithm (also known as 'Clique Tree') is a method used in machine learning to extract marginalization in general graphs.
- it entails performing belief propagation on a modified graph called a junction tree by cycle elimination

Outline

Example

The Message Passing Stuff



Thus

We can do the following

To pass information from below and from above to a certain node V.

Thus

We can do the following

To pass information from below and from above to a certain node V.

Thus

We call those messages

- π from above.
- λ from below.

Thus

We can do the following

To pass information from below and from above to a certain node V.

Thus

We call those messages

- π from above.
- λ from below.

Thus

We can do the following

To pass information from below and from above to a certain node V.

Thus

We call those messages

- π from above.
- λ from below.

Outline

Cinvestav

Inference on Trees

Recall

A rooted tree is a DAG

Inference on Trees

Recall

A rooted tree is a DAG

Now

- Let (G, P) be a Bayesian network whose DAG is a tree.
- Let a be a set of values of a subset $A \subset V$.

- The general case can be inferred from it

Inference on Trees

Recall

A rooted tree is a DAG

Now

- Let (G, P) be a Bayesian network whose DAG is a tree.
- Let a be a set of values of a subset $A \subset V$.

For simplicity

- Imagine that each node has two children.
- The general case can be inferred from it.

Inference on Trees

Recall

A rooted tree is a DAG

Now

- Let (G, P) be a Bayesian network whose DAG is a tree.
- Let a be a set of values of a subset $A \subset V$.

For simplicity

- Imagine that each node has two children.
- The general case can be inferred from it.

Inference on Trees

Recall

A rooted tree is a DAG

Now

- Let (G, P) be a Bayesian network whose DAG is a tree.
- Let a be a set of values of a subset $A \subset V$.

For simplicity

- Imagine that each node has two children.
- The general case can be inferred from it.

Then

Let D_X be the subset of A

- Containing all members that are in the subtree rooted at X
- Including X if $X \in A$

Then

Let D_X be the subset of A

- Containing all members that are in the subtree rooted at X
- Including X if $X \in A$

Let N_X be the subset

- Containing all members of A that are non-descendant's of X.
- This set includes X if $X \in A$

Then

Let D_X be the subset of A

- Containing all members that are in the subtree rooted at X
- Including X if $X \in A$

Let N_X be the subset

- Containing all members of A that are non-descendant's of X.
- This set includes X if $X \in A$

Then

Let D_X be the subset of A

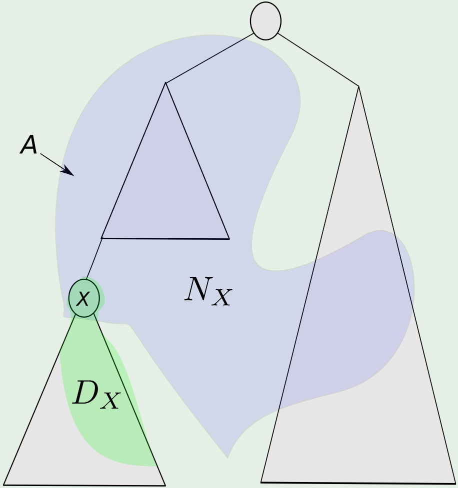
- Containing all members that are in the subtree rooted at X
- Including X if $X \in A$

Let N_X be the subset

- Containing all members of A that are non-descendant's of X.
- This set includes X if $X \in A$

Example

We have that $A = N_X \cup D_X$



Thus

We have for each value of x

$$
\begin{aligned}
P(x|A) &= P(x|d_X, n_X)\\
&= \frac{P(d_X, n_X|x)\,P(x)}{P(d_X, n_X)}\\
&= \frac{P(d_X|x, n_X)\,P(n_X|x)\,P(x)}{P(d_X, n_X)}\\
&= \frac{P(d_X|x, n_X)\,P(n_X, x)\,P(x)}{P(x)\,P(d_X, n_X)}\\
&= \frac{P(d_X|x)\,P(x|n_X)\,P(n_X)}{P(d_X, n_X)} \text{ Here because } d\text{-speration if } X \notin A\\
&= \frac{P(d_X|x)\,P(x|n_X)\,P(n_X)}{P(d_X|n_X)\,P(n_X)}
\end{aligned}
$$

Note: You need to prove when $X \in A$

Thus

We have for each value of x

$$
\begin{aligned}
P(x|A) &= P(x|d_X, n_X) \\
&= \frac{P(d_X, n_X|x)\, P(x)}{P(d_X, n_X)} \\
&= \frac{P(d_X|x, n_X)\, P(n_X|x)\, P(x)}{P(d_X, n_X)} \\
&= \frac{P(d_X|x, n_X)\, P(n_X, x)\, P(x)}{P(x)\, P(d_X, n_X)} \\
&= \frac{P(d_X|x)\, P(x|n_X)\, P(n_X)}{P(d_X, n_X)} \text{ Here because } d\text{-speration if } X \notin A \\
&= \frac{P(d_X|x)\, P(x|n_X)\, P(n_X)}{P(d_X|n_X)\, P(n_X)}
\end{aligned}
$$

Note: You need to prove when $X \in A$

Thus

We have for each value of x

$$
\begin{aligned}
P(x|A) &= P(x|d_X, n_X) \\
&= \frac{P(d_X, n_X|x)\,P(x)}{P(d_X, n_X)} \\
&= \frac{P(d_X|x, n_X)\,P(n_X|x)\,P(x)}{P(d_X, n_X)} \\
&= \frac{P(d_X|x, n_X)\,P(n_X, x)\,P(x)}{P(x)\,P(d_X, n_X)} \\
&= \frac{P(d_X|x)\,P(x|n_X)\,P(n_X)}{P(d_X, n_X)} \quad \text{Here because } d\text{-speration if } X \notin A \\
&= \frac{P(d_X|x)\,P(x|n_X)\,P(n_X)}{P(d_X|n_X)\,P(n_X)}
\end{aligned}
$$

Note: You need to prove when $X \in A$

Thus

We have for each value of x

$$
\begin{aligned}
P\left(x|A\right) &= P\left(x|d_X, n_X\right)\\
&= \frac{P\left(d_X, n_X|x\right)P\left(x\right)}{P\left(d_X, n_X\right)}\\
&= \frac{P\left(d_X|x, n_X\right)P\left(n_X|x\right)P\left(x\right)}{P\left(d_X, n_X\right)}\\
&= \frac{P\left(d_X|x, n_X\right)P\left(n_X, x\right)P\left(x\right)}{P\left(x\right)P\left(d_X, n_X\right)}\\
&= \frac{P\left(d_X|x\right)P\left(x|n_X\right)P\left(n_X\right)}{P\left(d_X, n_X\right)} \quad \text{Here because } d\text{-speration if } X \notin A\\
&= \frac{P\left(d_X|x\right)P\left(x|n_X\right)P\left(n_X\right)}{P\left(d_X|n_X\right)P\left(n_X\right)}
\end{aligned}
$$

Note: You need to prove when $X \in A$

Thus

We have for each value of x

$$
\begin{aligned}
P(x|A) &= P(x|d_X, n_X) \\
&= \frac{P(d_X, n_X|x)\,P(x)}{P(d_X, n_X)} \\
&= \frac{P(d_X|x, n_X)\,P(n_X|x)\,P(x)}{P(d_X, n_X)} \\
&= \frac{P(d_X|x, n_X)\,P(n_X, x)\,P(x)}{P(x)\,P(d_X, n_X)} \\
&= \frac{P(d_X|x)\,P(x|n_X)\,P(n_X)}{P(d_X, n_X)} \quad \text{Here because } d\text{-speration if } X \notin A \\
&= \frac{P(d_X|x)\,P(x|n_X)\,P(n_X)}{P(d_X|n_X)\,P(n_X)}
\end{aligned}
$$

Note: You need to prove when $X \in A$

Thus

We have for each value of x

$$
\begin{aligned}
P\left(x|A\right) &= P\left(x|d_X, n_X\right)\\
&= \frac{P\left(d_X, n_X|x\right)P\left(x\right)}{P\left(d_X, n_X\right)}\\
&= \frac{P\left(d_X|x, n_X\right)P\left(n_X|x\right)P\left(x\right)}{P\left(d_X, n_X\right)}\\
&= \frac{P\left(d_X|x, n_X\right)P\left(n_X, x\right)P\left(x\right)}{P\left(x\right)P\left(d_X, n_X\right)}\\
&= \frac{P\left(d_X|x\right)P\left(x|n_X\right)P\left(n_X\right)}{P\left(d_X, n_X\right)} \text{ Here because } d\text{-speration if } X \notin A\\
&= \frac{P\left(d_X|x\right)P\left(x|n_X\right)P\left(n_X\right)}{P\left(d_X|n_X\right)P\left(n_X\right)}
\end{aligned}
$$

Note: You need to prove when $X \in A$

Thus

We have for each value of x

$$\begin{aligned}
P\left(x|A\right) &= P\left(x|d_X, n_X\right)\\
&= \frac{P\left(d_X, n_X|x\right)P\left(x\right)}{P\left(d_X, n_X\right)}\\
&= \frac{P\left(d_X|x, n_X\right)P\left(n_X|x\right)P\left(x\right)}{P\left(d_X, n_X\right)}\\
&= \frac{P\left(d_X|x, n_X\right)P\left(n_X, x\right)P\left(x\right)}{P\left(x\right)P\left(d_X, n_X\right)}\\
&= \frac{P\left(d_X|x\right)P\left(x|n_X\right)P\left(n_X\right)}{P\left(d_X, n_X\right)} \text{ Here because } d\text{-speration if } X \notin A\\
&= \frac{P\left(d_X|x\right)P\left(x|n_X\right)P\left(n_X\right)}{P\left(d_X|n_X\right)P\left(n_X\right)}
\end{aligned}$$

Note: You need to prove when $X \in A$

Thus

We have for each value of x

$$P(x|A) = \frac{P(d_X|x)\,P(x|n_X)}{P(d_X|n_X)}$$
$$= \beta P(d_X|x)\,P(x|n_X)$$

where β, the normalizing factor, is a constant not depending on x.

Outline

Now, we develop the messages

We want

- $\lambda(x) \simeq P(d_X|x)$
- $\pi(x) \simeq P(x|n_X)$
  - Where $\simeq$ means "proportional to"

Now, we develop the messages

We want

- $\lambda(x) \simeq P(d_X|x)$
- $\pi(x) \simeq P(x|n_X)$
    - Where $\simeq$ means "proportional to"

Meaning

$\pi(x)$ may not be equal to $P(x|n_X)$, but $\pi(x) = k \times P(x|n_X)$.

where α, the normalizing factor, is a constant not depending on

Now, we develop the messages

We want

- $\lambda(x) \simeq P(d_X|x)$
- $\pi(x) \simeq P(x|n_X)$
  - Where $\simeq$ means "proportional to"

Meaning

$\pi(x)$ may not be equal to $P(x|n_X)$, but $\pi(x) = k \times P(x|n_X)$.

Once, we have that

$$P(x|a) = \alpha\lambda(x)\pi(x)$$

where α, the normalizing factor, is a constant not depending on x.

Now, we develop the messages

We want

- $\lambda(x) \simeq P(d_X|x)$
- $\pi(x) \simeq P(x|n_X)$
  - Where $\simeq$ means "proportional to"

Meaning

$\pi(x)$ may not be equal to $P(x|n_X)$, but $\pi(x) = k \times P(x|n_X)$.

Once, we have that

$$P(x|a) = \alpha\lambda(x)\pi(x)$$

where α, the normalizing factor, is a constant not depending on x.

Now, we develop the messages

We want

- $\lambda(x) \simeq P(d_X|x)$
- $\pi(x) \simeq P(x|n_X)$
  - Where $\simeq$ means "proportional to"

Meaning

$\pi(x)$ may not be equal to $P(x|n_X)$, but $\pi(x) = k \times P(x|n_X)$.

Once, we have that

$$P(x|a) = \alpha\lambda(x)\pi(x)$$

where α, the normalizing factor, is a constant not depending on x.

Now, we develop the messages

We want

- $\lambda(x) \simeq P(d_X|x)$
- $\pi(x) \simeq P(x|n_X)$
  - Where $\simeq$ means "proportional to"

Meaning

$\pi(x)$ may not be equal to $P(x|n_X)$, but $\pi(x) = k \times P(x|n_X)$.

Once, we have that

$$P(x|a) = \alpha\lambda(x)\pi(x)$$

where α, the normalizing factor, is a constant not depending on x.

Developing $\lambda\left(x\right)$

We need

- $\lambda\left(x\right) \simeq P\left(d_X|x\right)$

Developing $\lambda(x)$

We need

- $\lambda(x) \simeq P(d_X|x)$

Case 1: $X \in A$ and $X \in D_X$

Given any $X = \hat{x}$, we have that for $P(d_X|x) = 0$ for $x \neq \hat{x}$

- $\lambda(x) = 0$ for $x \neq \hat{x}$

Developing $\lambda(x)$

We need

- $\lambda(x) \simeq P(d_X|x)$

Case 1: $X \in A$ and $X \in D_X$

Given any $X = \hat{x}$, we have that for $P(d_X|x) = 0$ for $x \neq \hat{x}$

Thus, to achieve proportionality, we can set

- $\lambda(\hat{x}) \equiv 1$
- $\lambda(x) \equiv 0$ for $x \neq \hat{x}$

Developing $\lambda(x)$

We need

- $\lambda(x) \simeq P(d_X|x)$

Case 1: $X \in A$ and $X \in D_X$

Given any $X = \hat{x}$, we have that for $P(d_X|x) = 0$ for $x \neq \hat{x}$

Thus, to achieve proportionality, we can set

- $\lambda(\hat{x}) \equiv 1$
- $\lambda(x) \equiv 0$ for $x \neq \hat{x}$

Now

Case 2: $X \notin A$ and X is a leaf

Then, $d_X = \emptyset$ and

$$P\left(d_X|x\right) = P\left(\emptyset|x\right) = 1 \text{ for all values of } x$$

Thus, to achieve proportionality, we can set

$$\lambda\left(x\right) \equiv 1 \text{ for all values of } x$$

Now

Case 2: $X \notin A$ and X is a leaf

Then, $d_X = \emptyset$ and

$$P\left(d_X|x\right) = P\left(\emptyset|x\right) = 1 \text{ for all values of } x$$

Thus, to achieve proportionality, we can set

$$\lambda\left(x\right) \equiv 1 \text{ for all values of } x$$

Finally

Case 3: $X \notin A$ and X is a non-leaf

Let Y be X's left child, W be X's right child.

Since $X \notin A$

$$D_X = D_Y \cup D_W$$

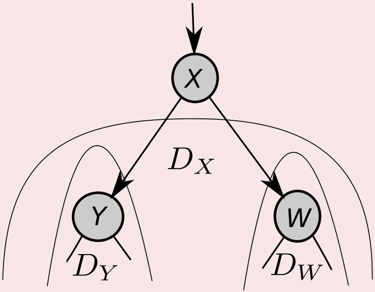
Finally

Case 3: $X \notin A$ and X is a non-leaf

Let Y be X's left child, W be X's right child.

Since $X \notin A$

$$D_X = D_Y \cup D_W$$



Thus

We have then

$$
\begin{aligned}
P\left(d_X|x\right) &= P\left(d_Y, d_W|x\right)\\
&= P\left(d_Y|x\right)P\left(d_W|x\right) \text{ Because the d-separation at } X\\
&= \sum_y P\left(d_Y, y|x\right)\sum_w P\left(d_W, w|x\right)\\
&= \sum_y P\left(y|x\right)P\left(d_Y|y\right)\sum_w P\left(w|x\right)P\left(d_W|w\right)\\
&\simeq \sum_y P\left(y|x\right)\lambda\left(y\right)\sum_w P\left(w|x\right)\lambda\left(w\right)
\end{aligned}
$$

Thus

We have then

$$
\begin{aligned}
P\left(d_X|x\right) &= P\left(d_Y, d_W|x\right)\\
&= P\left(d_Y|x\right)P\left(d_W|x\right) \text{ Because the d-separation at } X\\
&= \sum_y P\left(d_Y, y|x\right)\sum_w P\left(d_W, w|x\right)\\
&= \sum_y P\left(y|x\right)P\left(d_Y|y\right)\sum_w P\left(w|x\right)P\left(d_W|w\right)\\
&\simeq \sum_y P\left(y|x\right)\lambda\left(y\right)\sum_w P\left(w|x\right)\lambda\left(w\right)
\end{aligned}
$$

Thus, we can get proportionality by defining for all values of x

- $\lambda_Y\left(x\right) = \sum_y P\left(y|x\right)\lambda\left(y\right)$
- $\lambda_W\left(x\right) = \sum_w P\left(w|x\right)\lambda\left(w\right)$

Thus

We have then

$$
\begin{aligned}
P\left(d_X|x\right) &= P\left(d_Y, d_W|x\right)\\
&= P\left(d_Y|x\right)P\left(d_W|x\right) \text{ Because the d-separation at } X\\
&= \sum_y P\left(d_Y, y|x\right)\sum_w P\left(d_W, w|x\right)\\
&= \sum_y P\left(y|x\right)P\left(d_Y|y\right)\sum_w P\left(w|x\right)P\left(d_W|w\right)\\
&\simeq \sum_y P\left(y|x\right)\lambda\left(y\right)\sum_w P\left(w|x\right)\lambda\left(w\right)
\end{aligned}
$$

Thus, we can get proportionality by defining for all values of x:

- $\lambda_Y\left(x\right) = \sum_y P\left(y|x\right)\lambda\left(y\right)$
- $\lambda_W\left(x\right) = \sum_w P\left(w|x\right)\lambda\left(w\right)$

Thus

We have then

$$
\begin{aligned}
P\left(d_X|x\right) &= P\left(d_Y, d_W|x\right)\\
&= P\left(d_Y|x\right)P\left(d_W|x\right) \text{ Because the d-separation at } X\\
&= \sum_y P\left(d_Y, y|x\right)\sum_w P\left(d_W, w|x\right)\\
&= \sum_y P\left(y|x\right)P\left(d_Y|y\right)\sum_w P\left(w|x\right)P\left(d_W|w\right)\\
&\simeq \sum_y P\left(y|x\right)\lambda\left(y\right)\sum_w P\left(w|x\right)\lambda\left(w\right)
\end{aligned}
$$

Thus, we can get proportionality by defining for all values of x

- $\lambda_Y\left(x\right) = \sum_y P\left(y|x\right)\lambda\left(y\right)$
- $\lambda_W\left(x\right) = \sum_w P\left(w|x\right)\lambda\left(w\right)$

Thus

We have then

$$\begin{aligned}
P(d_X|x) &= P(d_Y, d_W|x) \\
&= P(d_Y|x)\,P(d_W|x) \text{ Because the d-separation at } X \\
&= \sum_y P(d_Y, y|x) \sum_w P(d_W, w|x) \\
&= \sum_y P(y|x)\,P(d_Y|y) \sum_w P(w|x)\,P(d_W|w) \\
&\simeq \sum_y P(y|x)\,\lambda(y) \sum_w P(w|x)\,\lambda(w)
\end{aligned}$$

Thus, we can get proportionality by defining for all values of x

- $\lambda_Y(x) = \sum_y P(y|x)\,\lambda(y)$
- $\lambda_W(x) = \sum_w P(w|x)\,\lambda(w)$

Thus

We have then

$$
\begin{aligned}
P\left(d_X|x\right) &= P\left(d_Y, d_W|x\right) \\
&= P\left(d_Y|x\right) P\left(d_W|x\right) \text{ Because the d-separation at } X \\
&= \sum_y P\left(d_Y, y|x\right) \sum_w P\left(d_W, w|x\right) \\
&= \sum_y P\left(y|x\right) P\left(d_Y|y\right) \sum_w P\left(w|x\right) P\left(d_W|w\right) \\
&\simeq \sum_y P\left(y|x\right) \lambda\left(y\right) \sum_w P\left(w|x\right) \lambda\left(w\right)
\end{aligned}
$$

Thus, we can get proportionality by defining for all values of x

- $\lambda_Y\left(x\right) = \sum_y P\left(y|x\right) \lambda\left(y\right)$
- $\lambda_W\left(x\right) = \sum_w P\left(w|x\right) \lambda\left(w\right)$

Thus

We have then

$$\begin{aligned}
P\left(d_X|x\right) &= P\left(d_Y, d_W|x\right) \\
&= P\left(d_Y|x\right)P\left(d_W|x\right) \text{ Because the d-separation at } X \\
&= \sum_y P\left(d_Y, y|x\right)\sum_w P\left(d_W, w|x\right) \\
&= \sum_y P\left(y|x\right)P\left(d_Y|y\right)\sum_w P\left(w|x\right)P\left(d_W|w\right) \\
&\simeq \sum_y P\left(y|x\right)\lambda\left(y\right)\sum_w P\left(w|x\right)\lambda\left(w\right)
\end{aligned}$$

Thus, we can get proportionality by defining for all values of x

- $\lambda_Y\left(x\right) = \sum_y P\left(y|x\right)\lambda\left(y\right)$
- $\lambda_W\left(x\right) = \sum_w P\left(w|x\right)\lambda\left(w\right)$

Thus

We have then

$$\lambda(x) = \lambda_Y(x)\,\lambda_W(x) \text{ for all values } x$$

Developing $\pi(x)$

We need

- $\pi(x) \simeq P(x|n_X)$

Developing $\pi(x)$

We need

- $\pi(x) \simeq P(x|n_X)$

Case 1: $X \in A$ and $X \in N_X$

Given any $X = \hat{x}$, we have:

- $P(\hat{x}|n_X) = P(\hat{x}|\hat{x}) = 1$
- $P(x|n_X) = P(x|\hat{x}) = 0$ for $x \neq \hat{x}$

- $\pi(x) = 0$ for $x \neq \hat{x}$

Developing $\pi(x)$

We need

- $\pi(x) \simeq P(x|n_X)$

Case 1: $X \in A$ and $X \in N_X$

Given any $X = \hat{x}$, we have:

- $P(\hat{x}|n_X) = P(\hat{x}|\hat{x}) = 1$
- $P(x|n_X) = P(x|\hat{x}) = 0$ for $x \neq \hat{x}$

Thus, to achieve proportionality, we can set

- $\pi(\hat{x}) = 1$
- $\pi(x) = 0$ for $x \neq \hat{x}$

Cinvestav

Developing $\pi(x)$

We need

- $\pi(x) \simeq P(x|n_X)$

Case 1: $X \in A$ and $X \in N_X$

Given any $X = \hat{x}$, we have:

- $P(\hat{x}|n_X) = P(\hat{x}|\hat{x}) = 1$
- $P(x|n_X) = P(x|\hat{x}) = 0$ for $x \neq \hat{x}$

Thus, to achieve proportionality, we can set

- $\pi(\hat{x}) \equiv 1$
- $\pi(x) \equiv 0$ for $x \neq \hat{x}$

Developing $\pi(x)$

We need

- $\pi(x) \simeq P(x|n_X)$

Case 1: $X \in A$ and $X \in N_X$

Given any $X = \hat{x}$, we have:

- $P(\hat{x}|n_X) = P(\hat{x}|\hat{x}) = 1$
- $P(x|n_X) = P(x|\hat{x}) = 0$ for $x \neq \hat{x}$

Thus, to achieve proportionality, we can set

- $\pi(\hat{x}) \equiv 1$
- $\pi(x) \equiv 0$ for $x \neq \hat{x}$

Cinvestav

Developing $\pi(x)$

We need

- $\pi(x) \simeq P(x|n_X)$

Case 1: $X \in A$ and $X \in N_X$

Given any $X = \hat{x}$, we have:

- $P(\hat{x}|n_X) = P(\hat{x}|\hat{x}) = 1$
- $P(x|n_X) = P(x|\hat{x}) = 0$ for $x \neq \hat{x}$

Thus, to achieve proportionality, we can set

- $\pi(\hat{x}) \equiv 1$
- $\pi(x) \equiv 0$ for $x \neq \hat{x}$

Cinvestav

Now

Case 2: $X \notin A$ and X is the root

In this specific case $n_X = \emptyset$ or the empty set of random variables.

Then

$$P(x|n_X) = P(x|\emptyset) = P(x) \text{ for all values of } x$$

Enforcing the proportionality, we get

$$\pi(x) \equiv P(x) \text{ for all values of } x$$

Now

Case 2: $X \notin A$ and X is the root

In this specific case $n_X = \emptyset$ or the empty set of random variables.

Then

$$P\left(x|n_X\right) = P\left(x|\emptyset\right) = P\left(x\right) \text{ for all values of } x$$

Enforcing the proportionality, we get

$$\pi\left(x\right) = P\left(x\right) \text{ for all values of } x$$

Now

Case 2: $X \notin A$ and X is the root

In this specific case $n_X = \emptyset$ or the empty set of random variables.

Then

$$P(x|n_X) = P(x|\emptyset) = P(x) \text{ for all values of } x$$

Enforcing the proportionality, we get

$$\pi(x) \equiv P(x) \text{ for all values of } x$$

Cinvestav

Then

Case 3: $X \notin A$ and X is not the root

Without loss of generality assume X is Z's right child and T is the Z's left child

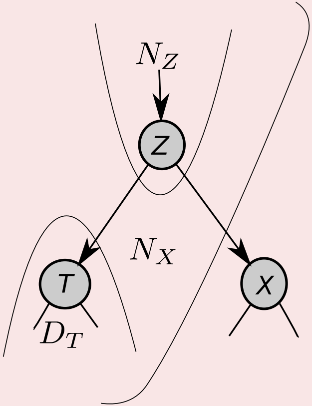
Then, $N_X = N_Z \cup D_T$

Then

Case 3: $X \notin A$ and X is not the root

Without loss of generality assume X is Z's right child and T is the Z's left child

Then, $N_X = N_Z \cup D_T$



Then

We have

$$
\begin{aligned}
P\left(x|n_X\right) &= \sum_z P\left(x|z\right) P\left(z|n_X\right) \\
&= \sum_z P\left(x|z\right) P\left(z|n_Z, d_T\right) \\
&= \sum_z P\left(x|z\right) \frac{P\left(z, n_Z, d_T\right)}{P\left(n_Z, d_T\right)} \\
&= \sum_z P\left(x|z\right) \frac{P\left(d_T, z|n_Z\right) P\left(n_Z\right)}{P\left(n_Z, d_T\right)} \\
&= \sum_z P\left(x|z\right) \frac{P\left(d_T|z, n_Z\right) P\left(z|n_Z\right) P(n_Z)}{P\left(n_Z, d_T\right)} \\
&= \sum_z P\left(x|z\right) \frac{P\left(d_T|z\right) P\left(z|n_Z\right) P(n_Z)}{P\left(n_Z, d_T\right)} \quad \text{Again the d-separation for}
\end{aligned}
$$

Then

We have

$$
\begin{aligned}
P\left(x|n_X\right) &= \sum_z P\left(x|z\right) P\left(z|n_X\right)\\
&= \sum_z P\left(x|z\right) P\left(z|n_Z, d_T\right)\\
&= \sum_z P\left(x|z\right) \frac{P\left(z, n_Z, d_T\right)}{P\left(n_Z, d_T\right)}\\
&= \sum_z P\left(x|z\right) \frac{P\left(d_T, z|n_Z\right) P\left(n_Z\right)}{P\left(n_Z, d_T\right)}\\
&= \sum_z P\left(x|z\right) \frac{P\left(d_T|z, n_Z\right) P\left(z|n_Z\right) P(n_Z)}{P\left(n_Z, d_T\right)}\\
&= \sum_z P\left(x|z\right) \frac{P\left(d_T|z\right) P\left(z|n_Z\right) P(n_Z)}{P\left(n_Z, d_T\right)} \quad \text{Again the d-separation for}
\end{aligned}
$$

Then

We have

$$
\begin{aligned}
P(x|n_X) &= \sum_z P(x|z)\,P(z|n_X) \\
&= \sum_z P(x|z)\,P(z|n_Z, d_T) \\
&= \sum_z P(x|z)\,\frac{P(z, n_Z, d_T)}{P(n_Z, d_T)} \\
&= \sum_z P(x|z)\,\frac{P(d_T, z|n_Z)\,P(n_Z)}{P(n_Z, d_T)} \\
&= \sum_z P(x|z)\,\frac{P(d_T|z, n_Z)\,P(z|n_Z)\,P(n_Z)}{P(n_Z, d_T)} \\
&= \sum_z P(x|z)\,\frac{P(d_T|z)\,P(z|n_Z)\,P(n_Z)}{P(n_Z, d_T)} \quad \text{Again the d-separation for } z
\end{aligned}
$$

Then

We have

$$
\begin{aligned}
P\left(x|n_X\right) &= \sum_z P\left(x|z\right) P\left(z|n_X\right) \\
&= \sum_z P\left(x|z\right) P\left(z|n_Z, d_T\right) \\
&= \sum_z P\left(x|z\right) \frac{P\left(z, n_Z, d_T\right)}{P\left(n_Z, d_T\right)} \\
&= \sum_z P\left(x|z\right) \frac{P\left(d_T, z|n_Z\right) P\left(n_Z\right)}{P\left(n_Z, d_T\right)} \\
&= \sum_z P\left(x|z\right) \frac{P\left(d_T|z, n_Z\right) P\left(z|n_Z\right) P(n_Z)}{P\left(n_Z, d_T\right)} \\
&= \sum_z P\left(x|z\right) \frac{P\left(d_T|z\right) P\left(z|n_Z\right) P(n_Z)}{P\left(n_Z, d_T\right)}
\end{aligned}
$$

Again the d-separation for

Then

We have

$$
\begin{aligned}
P\left(x|n_X\right) &= \sum_z P\left(x|z\right) P\left(z|n_X\right) \\
&= \sum_z P\left(x|z\right) P\left(z|n_Z, d_T\right) \\
&= \sum_z P\left(x|z\right) \frac{P\left(z, n_Z, d_T\right)}{P\left(n_Z, d_T\right)} \\
&= \sum_z P\left(x|z\right) \frac{P\left(d_T, z|n_Z\right) P\left(n_Z\right)}{P\left(n_Z, d_T\right)} \\
&= \sum_z P\left(x|z\right) \frac{P\left(d_T|z, n_Z\right) P\left(z|n_Z\right) P(n_Z)}{P\left(n_Z, d_T\right)} \\
&= \sum_z P\left(x|z\right) \frac{P\left(d_T|z\right) P\left(z|n_Z\right) P(n_Z)}{P\left(n_Z, d_T\right)} \quad \text{Again the d-separation for } z
\end{aligned}
$$

Then

We have

$$
\begin{aligned}
P(x|n_X) &= \sum_z P(x|z)\, P(z|n_X) \\
&= \sum_z P(x|z)\, P(z|n_Z, d_T) \\
&= \sum_z P(x|z)\, \frac{P(z, n_Z, d_T)}{P(n_Z, d_T)} \\
&= \sum_z P(x|z)\, \frac{P(d_T, z|n_Z)\, P(n_Z)}{P(n_Z, d_T)} \\
&= \sum_z P(x|z)\, \frac{P(d_T|z, n_Z)\, P(z|n_Z)\, P(n_Z)}{P(n_Z, d_T)} \\
&= \sum_z P(x|z)\, \frac{P(d_T|z)\, P(z|n_Z)\, P(n_Z)}{P(n_Z, d_T)} \text{ Again the d-separation for } z
\end{aligned}
$$

Last Step

We have

$$P(x|n_X) = \sum_z P(x|z)\frac{P(z|n_Z)\,P(n_Z)\,P(d_T|z)}{P(n_Z, d_T)}$$
$$= \gamma \sum_z P(x|z)\,\pi(z)\,\lambda_T(z)$$

where $\gamma = \frac{P(n_Z)}{P(n_Z, d_T)}$

Thus, we can achieve proportionality by

$$\pi_X(z) = \pi(z)\,\lambda_T(z)$$

Then, setting

$$\pi(x) = \sum_z P(x|z)\,\pi_X(z) \text{ for all values of } x$$

Last Step

We have

$$P(x|n_X) = \sum_z P(x|z) \frac{P(z|n_Z) P(n_Z) P(d_T|z)}{P(n_Z, d_T)}$$
$$= \gamma \sum_z P(x|z) \pi(z) \lambda_T(z)$$

where $\gamma = \frac{P(n_Z)}{P(n_Z, d_T)}$

Thus, we can achieve proportionality by

$$\pi_X(z) \equiv \pi(z) \lambda_T(z)$$

Then, setting

$$\pi(x) \equiv \sum_z P(x|z) \pi_X(z) \text{ for all values of } x$$

Last Step

We have

$$
\begin{aligned}
P\left(x|n_X\right) &= \sum_z P\left(x|z\right) \frac{P\left(z|n_Z\right) P\left(n_Z\right) P\left(d_T|z\right)}{P\left(n_Z, d_T\right)} \\
&= \gamma \sum_z P\left(x|z\right) \pi\left(z\right) \lambda_T\left(z\right)
\end{aligned}
$$

where $\gamma = \frac{P(n_Z)}{P(n_Z, d_T)}$

Thus, we can achieve proportionality by

$$
\pi_X\left(z\right) \equiv \pi\left(z\right) \lambda_T\left(z\right)
$$

Then, setting

$$
\pi\left(x\right) \equiv \sum_z P\left(x|z\right) \pi_X\left(z\right) \text{ for all values of } x
$$

Outline

Cinvestav

How do we implement this?

We require the following functions

- initial_tree
- update-tree

intial_tree has the following input and outputs

Input: $((G, P)$, A, a, $P(x|a))$

Output: After this call A and a are both empty making $P(x|a)$ the prior probability of x.

Then each time a variable V is instantiated for $\hat{v}$ the routine update-tree is called

Input: $((G, P)$, A, a, V, $\hat{v}$, $P(x|a))$

Output: After this call V has been added to A, $\hat{v}$ has been added to a and for every value of x, $P(x|a)$ has been updated to be the conditional probability of x given the new a.

How do we implement this?

We require the following functions

- initial_tree
- update-tree

intial_tree has the following input and outputs

Input: $((G, P)$, A, a, $P(x|a))$

Output: After this call A and a are both empty making $P(x|a)$ the prior probability of x.

Then each time a variable V is instantiated for $\hat{v}$ the routine update-tree is called

Input: $((G, P)$, A, a, V, $\hat{v}$, $P(x|a))$

Output: After this call V has been added to A, $\hat{v}$ has been added to a and for every value of x, $P(x|a)$ has been updated to be the conditional probability of x given the new a.

How do we implement this?

We require the following functions

- initial_tree
- update-tree

intial_tree has the following input and outputs

Input: $((G, P)$, A, a, $P(x|a))$

Output: After this call A and a are both empty making $P(x|a)$ the prior probability of x.

Then each time a variable V is instantiated for $\hat{v}$ the routine update-tree is called

Input: $((G, P)$, A, a, V, $\hat{v}$, $P(x|a))$

Output: After this call V has been added to A, $\hat{v}$ has been added to a and for every value of x, $P(x|a)$ has been updated to be the conditional probability of x given the new a.

Algorithm: Inference-in-trees

Problem

Given a Bayesian network whose DAG is a tree, determine the probabilities of the values of each node conditional on specified values of the nodes in some subset.

Input

Bayesian network (G, P) whose DAG is a tree, where $G = (V, E)$, and a set of values a of a subset A $\subseteq$ V.

Output

The Bayesian network (G, P) updated according to the values in a. The λ and π values and messages and $P(x|a)$ for each X$\in$V are considered part of the network.

Algorithm: Inference-in-trees

Problem

Given a Bayesian network whose DAG is a tree, determine the probabilities of the values of each node conditional on specified values of the nodes in some subset.

Input

Bayesian network (G, P) whose DAG is a tree, where $G = (V, E)$, and a set of values a of a subset A $\subseteq$ V.

Output

The Bayesian network (G, P) updated according to the values in a. The λ and π values and messages and $P(x|a)$ for each X$\in$V are considered part of the network.

Algorithm: Inference-in-trees

Problem

Given a Bayesian network whose DAG is a tree, determine the probabilities of the values of each node conditional on specified values of the nodes in some subset.

Input

Bayesian network (G, P) whose DAG is a tree, where $G = (V, E)$, and a set of values a of a subset A $\subseteq$ V.

Output

The Bayesian network (G, P) updated according to the values in a. The λ and π values and messages and $P(x|a)$ for each X$\in$V are considered part of the network.

Cinvestav

Initializing the tree

void initial_tree

input: **(Bayesian-network& $(\mathbb{G}, P)$ where $\mathbb{G} = (V, E)$, set-of-variables& A, set-of-variable-values& a)**

1. **A** $= \emptyset$
2. **a** $= \emptyset$
3. for (each X$\in$V)
4.   for (each value x of X)
5. $\lambda(x) = 1$ // Compute λ values.
6.   for (the parent Z of X) // Does nothing if X is the a root.
7.   for (each value z of Z)
8. $\lambda_X(z) = 1$ // Compute λ messages.
9. for (each value r of the root R)
10. $P(r|\mathbf{a}) = P(r)$ // Compute $P(r|\mathbf{a})$.
11. $\pi(r) = P(r)$ // Compute R's π values.
12. for (each child X of R)
13.   send_π_msg(R, X)

Initializing the tree

void initial_tree

input: **(Bayesian-network& $(\mathbb{G}, P)$ where $\mathbb{G} = (V, E)$, set-of-variables& A, set-of-variable-values& a)**

1. **A = $\emptyset$**
2. **a = $\emptyset$**
3. **for (each X$\in$V)**
4. **for (each value x of X)**
5. $\lambda(x) = 1$ // Compute λ values.
6. for (the parent Z of X) // Does nothing if X is the a root.
7. for (each value z of Z)
8. $\lambda_X(z) = 1$ // Compute λ messages.
9. for (each value r of the root R)
10. $P(r|\text{a}) = P(r)$ // Compute $P(r|\text{a})$.
11. $\pi(r) = P(r)$ // Compute R's π values.
12. for (each child X of R)
13. send_π_msg(R, X)

Initializing the tree

void initial_tree

input: **(Bayesian-network& $(\mathbb{G}, P)$ where $\mathbb{G} = (V, E)$, set-of-variables& A, set-of-variable-values& a)**

A $= \emptyset$

a $= \emptyset$

for (each X$\in$V)

  for (each value x of X)

    $\lambda(x) = 1$ **// Compute λ values.**

  for (the parent Z of X) **// Does nothing if X is the a root.**

    for (each value z of Z)

      $\lambda_X(z) = 1$ **// Compute λ messages.**

for (each value r of the root R)

  $P(r|\mathbf{a}) = P(r)$ **// Compute $P(r|\mathbf{a})$.**

  $\pi(r) = P(r)$ **// Compute R's π values.**

for (each child X of R)

  send_π_msg(R, X)

Initializing the tree

void initial_tree

input: **(Bayesian-network& $(\mathbb{G}, P)$ where $\mathbb{G} = (V, E)$, set-of-variables& A, set-of-variable-values& a)**

```
A = ∅
a = ∅
for (each X∈V)
    for (each value x of X)
        λ(x) = 1                    // Compute λ values.
    for (the parent Z of X)         // Does nothing if X is the a root.
        for (each value z of Z)
            λ_X(z) = 1              // Compute λ messages.
for (each value r of the root R)
    P(r|a) = P(r)                   // Compute P(r|a).
    π(r) = P(r)                     // Compute R's π values.
for (each child X of R)
    send_π_msg(R, X)
```

Initializing the tree

void initial_tree

input: **(Bayesian-network& $(\mathbb{G}, P)$ where $\mathbb{G} = (V, E)$, set-of-variables& A, set-of-variable-values& a)**

A $= \emptyset$

a $= \emptyset$

for (each X$\in$V)

  for (each value x of X)

    $\lambda(x) = 1$ **// Compute λ values.**

  for (the parent Z of X) **// Does nothing if X is the a root.**

    for (each value z of Z)

      $\lambda_X(z) = 1$ **// Compute λ messages.**

for (each value r of the root R)

  $P(r|\mathbf{a}) = P(r)$ **// Compute $P(r|\mathbf{a})$.**

  $\pi(r) = P(r)$ **// Compute R's π values.**

for (each child X of R)

  send_π_msg(R, X)

Initializing the tree

void initial_tree

input: **(Bayesian-network& $(\mathbb{G}, P)$ where $\mathbb{G} = (V, E)$, set-of-variables& A, set-of-variable-values& a)**

```
A = ∅
a = ∅
for (each X∈V)
    for (each value x of X)
        λ(x) = 1                 // Compute λ values.
    for (the parent Z of X)      // Does nothing if X is the a root.
        for (each value z of Z)
            λ_X(z) = 1           // Compute λ messages.
for (each value r of the root R)
    P(r|a) = P(r)                // Compute P(r|a).
    π(r) = P(r)                  // Compute R's π values.
for (each child X of R)
    send_π_msg(R, X)
```

Updating the tree

void update_tree

Input: (Bayesian-network& $(\mathbb{G}, P)$ where $\mathbb{G} = (V, E)$, set-of-variables& A, set-of-variable-values& a, variable V, variable-value $\hat{v}$)

1. A $=$ A$\cup\{V\}$, a$=$ a$\cup\{\hat{v}\}$, $\lambda(\hat{v}) = 1$, $\pi(\hat{v}) = 1$, $P(\hat{v}|\text{a}) = 1$ // Add V to A and instantiate V to $\hat{v}$
2. a $= \emptyset$
3. for (each value of $v \neq \hat{v}$)
4. $\quad\lambda(v) = 0$, $\pi(v) = 0$, $P(v|\text{a}) = 0$
5. if (V is not the root && V's parent $Z \notin$A)
6. $\quad$send_λ_msg(V, Z)
7. for (each child X of V such that $X \notin$A))
8. $\quad$send_π_msg(V, X)

Updating the tree

void update_tree

Input: (Bayesian-network& $(\mathbb{G}, P)$ where $\mathbb{G} = (V, E)$, set-of-variables& A, set-of-variable-values& a, variable V, variable-value $\hat{v}$)

1. A $=$ A$\cup\{V\}$, a$=$ a$\cup\{\hat{v}\}$, $\lambda(\hat{v}) = 1$, $\pi(\hat{v}) = 1$, $P(\hat{v}|$a$) = 1$ // Add V to A and instantiate V to $\hat{v}$
2. a $= \emptyset$
3. for (each value of $v \neq \hat{v}$)
4. $\lambda(v) = 0$, $\pi(v) = 0$, $P(v|$a$) = 0$
5. if (V is not the root && V 's parent $Z \notin$A)
6. send_λ_msg(V, Z)
7. for (each child X of V such that $X \notin$A))
8. send_π_msg(V, X)

Updating the tree

void update_tree

Input: (Bayesian-network& $(\mathbb{G}, P)$ where $\mathbb{G} = (V, E)$, set-of-variables& A, set-of-variable-values& a, variable V, variable-value $\hat{v}$)

1. A $=$ A$\cup \{V\}$, a$=$ a$\cup \{\hat{v}\}$, $\lambda(\hat{v}) = 1$, $\pi(\hat{v}) = 1$, $P(\hat{v}|$a$) = 1$ // Add V to A and instantiate V to $\hat{v}$
2. a $= \emptyset$
3. for (each value of $v \neq \hat{v}$)
4. $\quad\quad \lambda(v) = 0$, $\pi(v) = 0$, $P(v|$a$) = 0$
5. if (V is not the root && V 's parent $Z \notin$A)
6. $\quad\quad$ send_λ_msg(V, Z)
7. for (each child X of V such that $X \notin$A))
8. $\quad\quad$ send_π_msg(V, X)

Updating the tree

void update_tree

Input: (Bayesian-network& $(\mathbb{G}, P)$ where $\mathbb{G} = (V, E)$, set-of-variables& A, set-of-variable-values& a, variable V, variable-value $\hat{v}$)

1. A $=$ A$\cup\{V\}$, a$=$ a$\cup\{\hat{v}\}$, $\lambda(\hat{v}) = 1$, $\pi(\hat{v}) = 1$, $P(\hat{v}|$a$) = 1$ // Add V to A and instantiate V to $\hat{v}$
2. a $= \emptyset$
3. for (each value of $v \neq \hat{v}$)
4. $\quad\lambda(v) = 0$, $\pi(v) = 0$, $P(v|$a$) = 0$
5. if (V is not the root && V 's parent $Z \notin$A)
6. $\quad$send_λ_msg(V, Z)
7. for (each child X of V such that $X \notin$A))
8. $\quad$send_π_msg(V, X)

Updating the tree

void update_tree

Input: (Bayesian-network& $(\mathbb{G}, P)$ where $\mathbb{G} = (V, E)$, set-of-variables& A, set-of-variable-values& a, variable V , variable-value $\hat{v}$)

1. A $=$ A$\cup\{V\}$, a$=$ a$\cup\{\hat{v}\}$, $\lambda(\hat{v}) = 1$, $\pi(\hat{v}) = 1$, $P(\hat{v}|$a$) = 1$ // Add V to A and instantiate V to $\hat{v}$
2. a $= \emptyset$
3. for (each value of $v \neq \hat{v}$)
4. $\quad\lambda(v) = 0$, $\pi(v) = 0$, $P(v|$a$) = 0$
5. if (V is not the root && V 's parent $Z \notin$A)
6. $\quad$send_λ_msg(V, Z)
7. for (each child X of V such that $X \notin$A))
8. $\quad$send_π_msg(V, X)

Updating the tree

void update_tree

Input: (Bayesian-network& $(\mathbb{G}, P)$ where $\mathbb{G} = (V, E)$, set-of-variables& A, set-of-variable-values& a, variable V , variable-value $\hat{v}$)

1. A $=$ A$\cup\{V\}$, a$=$ a$\cup\{\hat{v}\}$, $\lambda(\hat{v}) = 1$, $\pi(\hat{v}) = 1$, $P(\hat{v}|\text{a}) = 1$ // Add V to A and instantiate V to $\hat{v}$
2. a $= \emptyset$
3. for (each value of $v \neq \hat{v}$)
4. $\lambda(v) = 0$, $\pi(v) = 0$, $P(v|\text{a}) = 0$
5. if (V is not the root && V 's parent $Z \notin$A)
6.   send_λ_msg(V, Z)
7. for (each child X of V such that $X \notin$A))
8.   send_π_msg(V, X)

Sending the λ message

void send_λ_msg(node Y , node X)

Note: For simplicity $(\mathbb{G}, P)$ is not shown as input.

1. for (each value of x)
2. $\quad\lambda_Y(x) = \sum_y P(y|x)\,\lambda(y)$ // Y sends X a λ message
3. $\quad\lambda(x) = \prod_{U \in CH_X} \lambda_U(x)$ // Compute $X's$ λ values
4. $\quad P(x|\text{a}) = \alpha\lambda(x)\,\pi(x)$ // Compute $P(x|\text{a})$
5. normalize $P(x|\text{a})$
6. if (X is not the root and $X's$ parent $Z \notin$A)
7. $\quad$send_λ_msg(X, Z)
8. for (each child W of X such that $W \neq Y$ and $W \in$A))
9. $\quad$send_π_msg(X, W)

Sending the λ message

void send_λ_msg(node Y , node X)

Note: For simplicity $(\mathbb{G}, P)$ is not shown as input.

1. for (each value of x)
2. $\qquad \lambda_Y(x) = \sum_y P(y|x)\lambda(y)$ $\qquad$ // Y sends X a λ message
3. $\qquad \lambda(x) = \prod_{U \in CH_X} \lambda_U(x)$ $\qquad$ // Compute $X's$ λ values
4. $\qquad P(x|\text{a}) = \alpha\lambda(x)\pi(x)$ $\qquad$ // Compute $P(x|\text{a})$
5. normalize $P(x|\text{a})$
6. if (X is not the root and $X's$ parent $Z \notin$A)
7. $\qquad$ send_λ_msg(X, Z)
8. for (each child W of X such that $W \neq Y$ and $W \in$A))
9. $\qquad$ send_π_msg(X, W)

Sending the λ message

void send_λ_msg(node Y , node X)

Note: For simplicity $(\mathbb{G}, P)$ is not shown as input.

1. for (each value of x)
2. $\quad \lambda_Y(x) = \sum_y P(y|x)\lambda(y)$ // Y sends X a λ message
3. $\quad \lambda(x) = \prod_{U \in CH_X} \lambda_U(x)$ // Compute $X's$ λ values
4. $\quad P(x|\text{a}) = \alpha\lambda(x)\pi(x)$ // Compute $P(x|\text{a})$
5. normalize $P(x|\text{a})$
6. if (X is not the root and $X's$ parent $Z \notin$A)
7. $\quad$ send_λ_msg(X, Z)
8. for (each child W of X such that $W \neq Y$ and $W \in$A))
9. $\quad$ send_π_msg(X, W)

Sending the λ message

void send_λ_msg(node Y , node X)

Note: For simplicity $(\mathbb{G}, P)$ is not shown as input.

1. for (each value of x)
2. $\quad \lambda_Y(x) = \sum_y P(y|x)\lambda(y)$ // Y sends X a λ message
3. $\quad \lambda(x) = \prod_{U \in CH_X} \lambda_U(x)$ // Compute $X's$ λ values
4. $\quad P(x|\text{a}) = \alpha\lambda(x)\pi(x)$ // Compute $P(x|\text{a})$
5. normalize $P(x|\text{a})$
6. if (X is not the root and $X's$ parent $Z \notin$A)
7. $\quad$ send_λ_msg(X, Z)
8. for (each child W of X such that $W \neq Y$ and $W \in$A))
9. $\quad$ send_π_msg(X, W)

Sending the π message

void send_π_msg(node Z , node X)

Note: For simplicity $(\mathbb{G}, P)$ is not shown as input.

1. for (each value of z)
2. $\pi_X(z) = \pi(z) \prod_{Y \in CH_Z - \{X\}} \lambda_Y(z)$ // Z sends X a π message
3. for (each value of x)
4. $\pi(x) = \sum_z P(x|z) \pi_X(z)$ // Compute$X's$ π values
5. $P(x|a) = \alpha \lambda(x) \pi(x)$ // Compute $P(x|$a)
6. normalize $P(x|a)$
7. for (each child Y of X such that $Y \notin$A))
8. send_π_msg(X, Y)

Sending the π message

void send_π_msg(node Z , node X)

Note: For simplicity $(\mathbb{G}, P)$ is not shown as input.

for (each value of z)

  $\pi_X(z) = \pi(z) \prod_{Y \in CH_Z - \{X\}} \lambda_Y(z)$ // Z sends X a π message

for (each value of x)

  $\pi(x) = \sum_z P(x|z) \pi_X(z)$ // Compute$X's$ π values

  $P(x|a) = \alpha \lambda(x) \pi(x)$ // Compute $P(x|$a)

normalize $P(x|a)$

for (each child Y of X such that $Y \notin$A))

  send_π_msg(X, Y)

Sending the π message

void send_π_msg(node Z , node X)

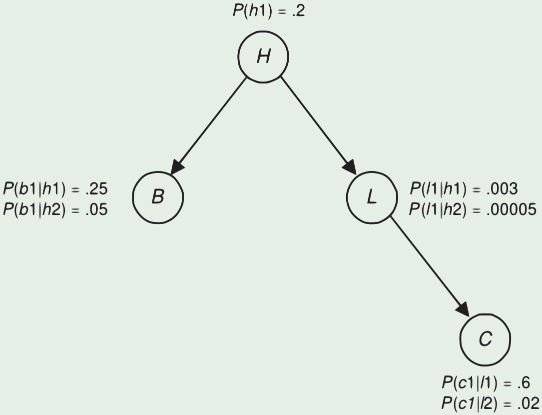
Note: For simplicity $(\mathbb{G}, P)$ is not shown as input.

1. for (each value of z)
2. $\pi_X(z) = \pi(z) \prod_{Y \in CH_Z - \{X\}} \lambda_Y(z)$ // Z sends X a π message
3. for (each value of x)
4. $\pi(x) = \sum_z P(x|z) \pi_X(z)$ // Compute$X's$ π values
5. $P(x|a) = \alpha\lambda(x)\pi(x)$ // Compute $P(x|$a)
6. normalize $P(x|a)$
7. for (each child Y of X such that $Y \notin$A))
8. send_π_msg(X, Y)

Cinvestav

Example of Tree Initialization

We have then



Calling initial_tree(($\mathbb{G}, P$), A, a)

We have then

- A$=\emptyset$, a$=\emptyset$

Calling initial_tree(($\mathbb{G}, P$), A, a)

We have then

- A=$\emptyset$, a=$\emptyset$

Compute λ values

- $\lambda(h1) = 1$;$\lambda(h2) = 1$;
- $\lambda(b1) = 1$;$\lambda(b2) = 1$;
- $\lambda(l1) = 1$;$\lambda(l2) = 1$;
- $\lambda(c1) = 1$;$\lambda(c2) = 1$;

Calling initial_tree(($\mathbb{G}, P$), A, a)

We have then

- A=$\emptyset$, a=$\emptyset$

Compute λ values

- $\lambda(h1) = 1;\lambda(h2) = 1;$
- $\lambda(b1) = 1; \lambda(b2) = 1;$
- $\lambda(l1) = 1; \lambda(l2) = 1;$
- $\lambda(c1) = 1; \lambda(c2) = 1;$

Compute λ messages

- $\lambda_B(h1) = 1; \lambda_B(h2) = 1;$
- $\lambda_L(h1) = 1; \lambda_L(h2) = 1;$
- $\lambda_C(l1) = 1; \lambda_C(l2) = 1;$

Calling initial_tree(($\mathbb{G}, P$), A, a)

We have then

- A=$\emptyset$, a=$\emptyset$

Compute λ values

- $\lambda(h1) = 1; \lambda(h2) = 1;$
- $\lambda(b1) = 1; \lambda(b2) = 1;$
- $\lambda(l1) = 1; \lambda(l2) = 1;$
- $\lambda(c1) = 1; \lambda(c2) = 1;$

Compute λ messages

- $\lambda_B(h1) = 1; \lambda_B(h2) = 1;$
- $\lambda_L(h1) = 1; \lambda_L(h2) = 1;$
- $\lambda_C(l1) = 1; \lambda_C(l2) = 1;$

Calling initial_tree(($\mathbb{G}, P$), A, a)

We have then

- A=$\emptyset$, a=$\emptyset$

Compute λ values

- $\lambda(h1) = 1; \lambda(h2) = 1;$
- $\lambda(b1) = 1; \lambda(b2) = 1;$
- $\lambda(l1) = 1; \lambda(l2) = 1;$
- $\lambda(c1) = 1; \lambda(c2) = 1;$

Compute λ messages

- $\lambda_B(h1) = 1; \lambda_B(h2) = 1;$
- $\lambda_L(h1) = 1; \lambda_L(h2) = 1;$
- $\lambda_C(l1) = 1; \lambda_C(l2) = 1;$

Calling initial_tree(($\mathbb{G}, P$), A, a)

We have then

- A=$\emptyset$, a=$\emptyset$

Compute λ values

- $\lambda(h1) = 1;\lambda(h2) = 1;$
- $\lambda(b1) = 1; \lambda(b2) = 1;$
- $\lambda(l1) = 1; \lambda(l2) = 1;$
- $\lambda(c1) = 1; \lambda(c2) = 1;$

Compute λ messages

- $\lambda_B(h1) = 1; \lambda_B(h2) = 1;$
- $\lambda_L(h1) = 1; \lambda_L(h2) = 1;$
- $\lambda_C(l1) = 1; \lambda_C(l2) = 1;$

Calling initial_tree(($\mathbb{G}, P$), A, a)

We have then

- A=$\emptyset$, a=$\emptyset$

Compute λ values

- $\lambda(h1) = 1;\lambda(h2) = 1;$
- $\lambda(b1) = 1; \lambda(b2) = 1;$
- $\lambda(l1) = 1; \lambda(l2) = 1;$
- $\lambda(c1) = 1; \lambda(c2) = 1;$

Compute λ messages

- $\lambda_B(h1) = 1; \lambda_B(h2) = 1;$
- $\lambda_L(h1) = 1; \lambda_L(h2) = 1;$
- $\lambda_C(l1) = 1; \lambda_C(l2) = 1;$

Calling initial_tree(($\mathbb{G}, P$), A, a)

We have then

- A=$\emptyset$, a=$\emptyset$

Compute λ values

- $\lambda(h1) = 1; \lambda(h2) = 1;$
- $\lambda(b1) = 1; \lambda(b2) = 1;$
- $\lambda(l1) = 1; \lambda(l2) = 1;$
- $\lambda(c1) = 1; \lambda(c2) = 1;$

Compute λ messages

- $\lambda_B(h1) = 1; \lambda_B(h2) = 1;$
- $\lambda_L(h1) = 1; \lambda_L(h2) = 1;$
- $\lambda_C(l1) = 1; \lambda_C(l2) = 1;$

Calling initial_tree$((\mathbb{G}, P)$, A, a)

Compute $P\left(h|\emptyset\right)$

- $P(h1|\emptyset) = P(h1) = 0.2$
- $P(h2|\emptyset) = P(h2) = 0.8$

Cinvestav

Calling initial_tree(($\mathbb{G}$, P), A, a)

Compute $P(h|\emptyset)$

- $P(h1|\emptyset) = P(h1) = 0.2$
- $P(h2|\emptyset) = P(h2) = 0.8$

Compute H's π values

- $\pi(h1) = P(h1) = 0.2$
- $\pi(h2) = P(h2) = 0.8$

- send_π_msg(H, B)

Calling initial_tree(($\mathbb{G}$, P), A, a)

Compute $P\left(h|\emptyset\right)$

- $P(h1|\emptyset) = P(h1) = 0.2$
- $P(h2|\emptyset) = P(h2) = 0.8$

Compute H's π values

- $\pi(h1) = P(h1) = 0.2$
- $\pi(h2) = P(h2) = 0.8$

Send messages

- send_π_msg(H, B)
- send_π_msg(H, L)

Calling initial_tree($(\mathbb{G}, P)$, A, a)

Compute $P(h|\emptyset)$

- $P(h1|\emptyset) = P(h1) = 0.2$
- $P(h2|\emptyset) = P(h2) = 0.8$

Compute H's π values

- $\pi(h1) = P(h1) = 0.2$
- $\pi(h2) = P(h2) = 0.8$

Send messages

- send_π_msg(H, B)
- send_π_msg(H, L)

Cinvestav

Calling initial_tree(($\mathbb{G}$, P), A, a)

Compute $P\left(h|\emptyset\right)$

- $P(h1|\emptyset) = P(h1) = 0.2$
- $P(h2|\emptyset) = P(h2) = 0.8$

Compute H's π values

- $\pi(h1) = P(h1) = 0.2$
- $\pi(h2) = P(h2) = 0.8$

Send messages

- send_π_msg(H, B)
- send_π_msg(H, L)

Calling initial_tree(($\mathbb{G}$, P), A, a)

Compute $P\left(h|\emptyset\right)$

- $P(h1|\emptyset) = P(h1) = 0.2$
- $P(h2|\emptyset) = P(h2) = 0.8$

Compute H's π values

- $\pi(h1) = P(h1) = 0.2$
- $\pi(h2) = P(h2) = 0.8$

Send messages

- send_π_msg(H, B)
- send_π_msg(H, L)

Cinvestav

The call send_π_msg(H, B)

H sends B a π message

- $\pi_B(h1) = \pi(h1)\lambda_L(h1) = 0.2 \times 1 = 0.2$
- $\pi_B(h2) = \pi(h2)\lambda_L(h2) = 0.8 \times 1 = 0.8$

The call send_π_msg(H, B)

H sends B a π message

- $\pi_B(h1) = \pi(h1)\lambda_L(h1) = 0.2 \times 1 = 0.2$
- $\pi_B(h2) = \pi(h2)\lambda_L(h2) = 0.8 \times 1 = 0.8$

Compute B's π values

$$\begin{aligned}
\pi(b1) &= P(b1|h1)\,\pi_B(h1) + P(b1|h2)\,\pi_B(h2) \\
&= (0.25)(0.2) + (0.05)(0.8) = 0.09 \\
\pi(b2) &= P(b2|h1)\,\pi_B(h1) + P(b2|h2)\,\pi_B(h2) \\
&= (0.75)(0.2) + (0.95)(0.8) = 0.91
\end{aligned}$$

Cinvestav

The call send_π_msg(H, B)

H sends B a π message

- $\pi_B(h1) = \pi(h1)\lambda_L(h1) = 0.2 \times 1 = 0.2$
- $\pi_B(h2) = \pi(h2)\lambda_L(h2) = 0.8 \times 1 = 0.8$

Compute B's π values

$$\pi\left(b1\right) = P\left(b1|h1\right)\pi_B\left(h1\right) + P\left(b1|h2\right)\pi_B\left(h2\right)$$

$$= \left(0.25\right)\left(0.2\right) + \left(0.05\right)\left(0.8\right) = 0.09$$

$$\pi\left(b2\right) = P\left(b2|h1\right)\pi_B\left(h1\right) + P\left(b2|h2\right)\pi_B\left(h2\right)$$

$$= \left(0.75\right)\left(0.2\right) + \left(0.95\right)\left(0.8\right) = 0.91$$

The call send_π_msg(H, B)

H sends B a π message

- $\pi_B(h1) = \pi(h1)\lambda_L(h1) = 0.2 \times 1 = 0.2$
- $\pi_B(h2) = \pi(h2)\lambda_L(h2) = 0.8 \times 1 = 0.8$

Compute B's π values

$$
\begin{aligned}
\pi(b1) &= P(b1|h1)\,\pi_B(h1) + P(b1|h2)\,\pi_B(h2) \\
&= (0.25)(0.2) + (0.05)(0.8) = 0.09
\end{aligned}
$$

$$
\begin{aligned}
\pi(b2) &= P(b2|h1)\,\pi_B(h1) + P(b2|h2)\,\pi_B(h2) \\
&= (0.75)(0.2) + (0.95)(0.8) = 0.91
\end{aligned}
$$

The call send_π_msg(H, B)

H sends B a π message

- $\pi_B(h1) = \pi(h1)\lambda_L(h1) = 0.2 \times 1 = 0.2$
- $\pi_B(h2) = \pi(h2)\lambda_L(h2) = 0.8 \times 1 = 0.8$

Compute B's π values

$$\begin{aligned}
\pi\left(b1\right) &= P\left(b1|h1\right)\pi_B\left(h1\right) + P\left(b1|h2\right)\pi_B\left(h2\right)\\
&= \left(0.25\right)\left(0.2\right) + \left(0.05\right)\left(0.8\right) = 0.09\\
\pi\left(b2\right) &= P\left(b2|h1\right)\pi_B\left(h1\right) + P\left(b2|h2\right)\pi_B\left(h2\right)\\
&= \left(0.75\right)\left(0.2\right) + \left(0.95\right)\left(0.8\right) = 0.91
\end{aligned}$$

The call send_π_msg(H, B)

H sends B a π message

- $\pi_B(h1) = \pi(h1)\lambda_L(h1) = 0.2 \times 1 = 0.2$
- $\pi_B(h2) = \pi(h2)\lambda_L(h2) = 0.8 \times 1 = 0.8$

Compute B's π values

$$\begin{aligned}
\pi\left(b1\right) &= P\left(b1|h1\right)\pi_B\left(h1\right) + P\left(b1|h2\right)\pi_B\left(h2\right)\\
&= \left(0.25\right)\left(0.2\right) + \left(0.05\right)\left(0.8\right) = 0.09\\
\pi\left(b2\right) &= P\left(b2|h1\right)\pi_B\left(h1\right) + P\left(b2|h2\right)\pi_B\left(h2\right)\\
&= \left(0.75\right)\left(0.2\right) + \left(0.95\right)\left(0.8\right) = 0.91
\end{aligned}$$

The call send_π_msg(H, B)

Compute $P(b|\emptyset)$

- $P(b1|\emptyset) = \alpha\lambda(b1)\pi(b1) = \alpha(1)(0.09) = 0.09\alpha$
- $P(b2|\emptyset) = \alpha\lambda(b2)\pi(b2) = \alpha(1)(0.91) = 0.91\alpha$

The call send_π_msg(H, B)

Compute $P(b|\emptyset)$

- $P(b1|\emptyset) = \alpha\lambda(b1)\pi(b1) = \alpha(1)(0.09) = 0.09\alpha$
- $P(b2|\emptyset) = \alpha\lambda(b2)\pi(b2) = \alpha(1)(0.91) = 0.91\alpha$

Then, normalize

$$P(b1|\emptyset) = \frac{0.09\alpha}{0.09\alpha + 0.91\alpha} = 0.09$$

$$P(b2|\emptyset) = \frac{0.91\alpha}{0.09\alpha + 0.91\alpha} = 0.91$$

Cinvestav

The call send_π_msg(H, B)

Compute $P\left(b|\emptyset\right)$

- $P\left(b1|\emptyset\right) = \alpha\lambda\left(b1\right)\pi\left(b1\right) = \alpha\left(1\right)\left(0.09\right) = 0.09\alpha$
- $P\left(b2|\emptyset\right) = \alpha\lambda\left(b2\right)\pi\left(b2\right) = \alpha\left(1\right)\left(0.91\right) = 0.91\alpha$

Then, normalize

$$P\left(b1|\emptyset\right) = \frac{0.09\alpha}{0.09\alpha + 0.91\alpha} = 0.09$$

$$P\left(b2|\emptyset\right) = \frac{0.91\alpha}{0.09\alpha + 0.91\alpha} = 0.91$$

The call send_π_msg(H, B)

Compute $P\left(b|\emptyset\right)$

- $P\left(b1|\emptyset\right) = \alpha\lambda\left(b1\right)\pi\left(b1\right) = \alpha\left(1\right)\left(0.09\right) = 0.09\alpha$
- $P\left(b2|\emptyset\right) = \alpha\lambda\left(b2\right)\pi\left(b2\right) = \alpha\left(1\right)\left(0.91\right) = 0.91\alpha$

Then, normalize

$$P\left(b1|\emptyset\right) = \frac{0.09\alpha}{0.09\alpha + 0.91\alpha} = 0.09$$
$$P\left(b2|\emptyset\right) = \frac{0.91\alpha}{0.09\alpha + 0.91\alpha} = 0.91$$

Send the call send_π_msg(H, L)

H sends L a π message

- $\pi_L(h1) = \pi(h1)\,\lambda_B(h1) = (0.2)(1) = 0.2$
- $\pi_L(h2) = \pi(h2)\,\lambda_B(h2) = (0.8)(1) = 0.8$

Send the call send_π_msg(H, L)

H sends L a π message

- $\pi_L(h1) = \pi(h1)\,\lambda_B(h1) = (0.2)(1) = 0.2$
- $\pi_L(h2) = \pi(h2)\,\lambda_B(h2) = (0.8)(1) = 0.8$

Compute $L's$ π values

$$\begin{aligned}\pi(l1) &= P(l1|h1)\,\pi_L(h1) + P(l1|h2)\,\pi_L(h2)\\ &= (0.003)(0.2) + (0.00005)(0.8) = 0.00064\\ \pi(l2) &= P(l2|h1)\,\pi_B(h1) + P(l2|h2)\,\pi_B(h2)\\ &= (0.997)(0.2) + (0.99995)(0.8) = 0.99936\end{aligned}$$

Send the call send_π_msg(H, L)

H sends L a π message

- $\pi_L(h1) = \pi(h1)\,\lambda_B(h1) = (0.2)(1) = 0.2$
- $\pi_L(h2) = \pi(h2)\,\lambda_B(h2) = (0.8)(1) = 0.8$

Compute $L's$ π values

$$\begin{aligned}\pi(l1) &= P(l1|h1)\,\pi_L(h1) + P(l1|h2)\,\pi_L(h2)\\ &= (0.003)(0.2) + (0.00005)(0.8) = 0.00064\end{aligned}$$

$$\begin{aligned}\pi(l2) &= P(l2|h1)\,\pi_B(h1) + P(l2|h2)\,\pi_B(h2)\\ &= (0.997)(0.2) + (0.99995)(0.8) = 0.99936\end{aligned}$$

Compute $P(l|\emptyset)$

- $P(l1|\emptyset) = \alpha\lambda(l1)\,\pi(l1) = \alpha(1)(0.00064) = 0.00064\alpha$
- $P(l2|\emptyset) = \alpha\lambda(l2)\,\pi(l2) = \alpha(1)(0.99936) = 0.99936\alpha$

Send the call send_π_msg(H, L)

H sends L a π message

- $\pi_L(h1) = \pi(h1)\,\lambda_B(h1) = (0.2)(1) = 0.2$
- $\pi_L(h2) = \pi(h2)\,\lambda_B(h2) = (0.8)(1) = 0.8$

Compute $L's$ π values

$$
\begin{aligned}
\pi(l1) &= P(l1|h1)\,\pi_L(h1) + P(l1|h2)\,\pi_L(h2) \\
&= (0.003)(0.2) + (0.00005)(0.8) = 0.00064 \\
\pi(l2) &= P(l2|h1)\,\pi_B(h1) + P(l2|h2)\,\pi_B(h2) \\
&= (0.997)(0.2) + (0.99995)(0.8) = 0.99936
\end{aligned}
$$

Compute $P(l|\emptyset)$

- $P(l1|\emptyset) = \alpha\lambda(l1)\,\pi(l1) = \alpha(1)(0.00064) = 0.00064\alpha$
- $P(l2|\emptyset) = \alpha\lambda(l2)\,\pi(l2) = \alpha(1)(0.99936) = 0.99936\alpha$

Send the call send_π_msg(H, L)

H sends L a π message

- $\pi_L(h1) = \pi(h1)\lambda_B(h1) = (0.2)(1) = 0.2$
- $\pi_L(h2) = \pi(h2)\lambda_B(h2) = (0.8)(1) = 0.8$

Compute $L's$ π values

$$\begin{aligned}
\pi(l1) &= P(l1|h1)\pi_L(h1) + P(l1|h2)\pi_L(h2) \\
&= (0.003)(0.2) + (0.00005)(0.8) = 0.00064 \\
\pi(l2) &= P(l2|h1)\pi_B(h1) + P(l2|h2)\pi_B(h2) \\
&= (0.997)(0.2) + (0.99995)(0.8) = 0.99936
\end{aligned}$$

Compute $P(l|\emptyset)$

- $P(l1|\emptyset) = \alpha\lambda(l1)\pi(l1) = \alpha(1)(0.00064) = 0.00064\alpha$
- $P(l2|\emptyset) = \alpha\lambda(l2)\pi(l2) = \alpha(1)(0.99936) = 0.99936\alpha$

Send the call send_π_msg(H, L)

H sends L a π message

- $\pi_L(h1) = \pi(h1)\lambda_B(h1) = (0.2)(1) = 0.2$
- $\pi_L(h2) = \pi(h2)\lambda_B(h2) = (0.8)(1) = 0.8$

Compute $L's$ π values

$$\begin{aligned}
\pi(l1) &= P(l1|h1)\pi_L(h1) + P(l1|h2)\pi_L(h2) \\
&= (0.003)(0.2) + (0.00005)(0.8) = 0.00064 \\
\pi(l2) &= P(l2|h1)\pi_B(h1) + P(l2|h2)\pi_B(h2) \\
&= (0.997)(0.2) + (0.99995)(0.8) = 0.99936
\end{aligned}$$

Compute $P(l|\emptyset)$

- $P(l1|\emptyset) = \alpha\lambda(l1)\pi(l1) = \alpha(1)(0.00064) = 0.00064\alpha$
- $P(l2|\emptyset) = \alpha\lambda(l2)\pi(l2) = \alpha(1)(0.99936) = 0.99936\alpha$

Send the call send_π_msg(H, L)

Then, normalize

$$P(l1|\emptyset) = \frac{0.00064\alpha}{0.00064\alpha + 0.99936\alpha} = 0.00064$$

$$P(l2|\emptyset) = \frac{0.99936\alpha}{0.00064\alpha + 0.99936\alpha} = 0.99936$$

Send the call send_π_msg(H, L)

Then, normalize

$$P\left(l1|\emptyset\right) = \frac{0.00064\alpha}{0.00064\alpha + 0.99936\alpha} = 0.00064$$
$$P\left(l2|\emptyset\right) = \frac{0.99936\alpha}{0.00064\alpha + 0.99936\alpha} = 0.99936$$

Cinvestav

Send the call send_π_msg(L, C)

L sends C a π message

- $\pi_C(l1) = \pi(l1) = 0.00064$
- $\pi_C(l2) = \pi(l2) = 0.99936$

Send the call send_π_msg(L, C)

L sends C a π message

- $\pi_C(l1) = \pi(l1) = 0.00064$
- $\pi_C(l2) = \pi(l2) = 0.99936$

Compute $C's$ π values

$$
\begin{aligned}
\pi(c1) &= P(c1|l1)\,\pi_C(l1) + P(c1|l2)\,\pi_C(l2)\\
&= (0.6)(0.00064) + (0.02)(0.99936) = 0.02037\\
\pi(c2) &= P(c2|l1)\,\pi_C(h1) + P(c2|l2)\,\pi_C(l2)\\
&= (0.4)(0.00064) + (0.98)(0.99936) = 0.97963
\end{aligned}
$$

Send the call send_π_msg(L, C)

L sends C a π message

- $\pi_C(l1) = \pi(l1) = 0.00064$
- $\pi_C(l2) = \pi(l2) = 0.99936$

Compute $C's$ π values

$$\begin{aligned}
\pi(c1) &= P(c1|l1)\,\pi_C(l1) + P(c1|l2)\,\pi_C(l2) \\
&= (0.6)(0.00064) + (0.02)(0.99936) = 0.02037 \\
\pi(c2) &= P(c2|l1)\,\pi_C(h1) + P(c2|l2)\,\pi_C(l2) \\
&= (0.4)(0.00064) + (0.98)(0.99936) = 0.97963
\end{aligned}$$

Cinvestav

Send the call send_π_msg(L, C)

Compute $P(c|\emptyset)$

- $P(c1|\emptyset) = \alpha\lambda(c1)\,\pi(c1) = \alpha(1)(0.02037) = 0.02037\alpha$
- $P(c2|\emptyset) = \alpha\lambda(c2)\,\pi(c2) = \alpha(1)(0.97963) = 0.97963\alpha$

Send the call send_π_msg(L, C)

Compute $P\left(c|\emptyset\right)$

- $P\left(c1|\emptyset\right)=\alpha\lambda\left(c1\right)\pi\left(c1\right)=\alpha\left(1\right)\left(0.02037\right)=0.02037\alpha$
- $P\left(c2|\emptyset\right)=\alpha\lambda\left(c2\right)\pi\left(c2\right)=\alpha\left(1\right)\left(0.97963\right)=0.97963\alpha$

Normalize

$$P\left(c1|\emptyset\right)=\frac{0.02037\alpha}{0.02037\alpha+0.97963\alpha}=0.02037$$

$$P\left(c2|\emptyset\right)=\frac{0.99936\alpha}{0.02037\alpha+0.97963\alpha}=0.97963$$

Send the call send_π_msg(L, C)

Compute $P(c|\emptyset)$

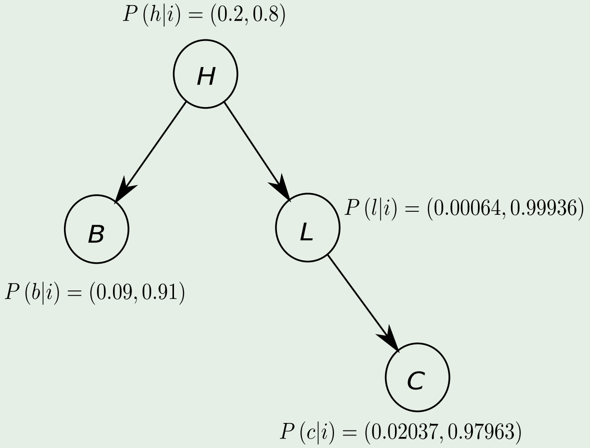
- $P(c1|\emptyset) = \alpha\lambda(c1)\,\pi(c1) = \alpha(1)(0.02037) = 0.02037\alpha$
- $P(c2|\emptyset) = \alpha\lambda(c2)\,\pi(c2) = \alpha(1)(0.97963) = 0.97963\alpha$

Normalize

$$P(c1|\emptyset) = \frac{0.02037\alpha}{0.02037\alpha + 0.97963\alpha} = 0.02037$$
$$P(c2|\emptyset) = \frac{0.99936\alpha}{0.02037\alpha + 0.97963\alpha} = 0.97963$$

Final Graph

We have then



For the Generalization Please look at...

Look at pages 123 - 156 at

Richard E. Neapolitan. 2003. Learning Bayesian Networks. Prentice-Hall, Inc

Cinvestav

History

Invented in 1988

Invented by Lauritzen and Spiegelhalter, 1988

Something Notable

The general idea is that the propagation of evidence through the network can be carried out more efficiently by representing the joint probability distribution on an undirected graph called the Junction tree (or Join tree).

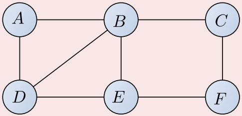
History

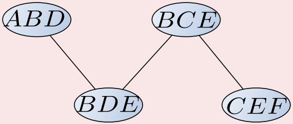
Invented in 1988

Invented by Lauritzen and Spiegelhalter, 1988

Something Notable

The general idea is that the propagation of evidence through the network can be carried out more efficiently by representing the joint probability distribution on an undirected graph called the Junction tree (or Join tree).





More in the Intuition

High-level Intuition

Computing marginals is straightforward in a tree structure.

Junction Tree Characteristics

The junction tree has the following characteristics

- It is an undirected tree
- Its nodes are clusters of variables (i.e. from the original BN)
- Given two clusters, C_1 and C_2, every node on the path between them contains their intersection $C_1 \cap C_2$

Cinvestav

Junction Tree Characteristics

The junction tree has the following characteristics

- It is an undirected tree
- Its nodes are clusters of variables (i.e. from the original BN)
- Given two clusters, C_1 and C_2, every node on the path between them contains their intersection $C_1 \cap C_2$

In addition

A Separator, S, is associated with each edge and contains the variables in the intersection between neighboring nodes

Cinvestav

Junction Tree Characteristics

The junction tree has the following characteristics

- It is an undirected tree
- Its nodes are clusters of variables (i.e. from the original BN)
- Given two clusters, C_1 and C_2, every node on the path between them contains their intersection $C_1 \cap C_2$

In addition

A Separator, S, is associated with each edge and contains the variables in the intersection between neighboring nodes

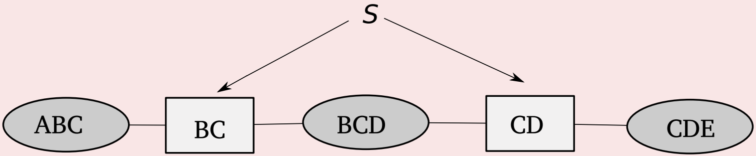
Junction Tree Characteristics

The junction tree has the following characteristics

- It is an undirected tree
- Its nodes are clusters of variables (i.e. from the original BN)
- Given two clusters, C_1 and C_2, every node on the path between them contains their intersection $C_1 \cap C_2$

In addition

A Separator, S, is associated with each edge and contains the variables in the intersection between neighboring nodes



Outline

Cinvestav

Outline

Cinvestav

Simplicial Node

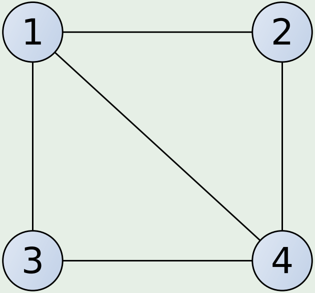
Simplicial Node

In a graph G, a vertex v is called **simplicial** if and only if the subgraph of G induced by the vertex set $\{v\} \cup N(v)$ is a clique.

- $N(v)$ is the neighbor of v in the Graph.

Example

Vertex 3 is simplicial, while 4 is not



Perfect Elimination Ordering

Definition

A graph G on n vertices is said to have a **perfect elimination ordering** if and only if there is an ordering $\{v_1, ..., v_n\}$ of G's vertices, such that each v_i is simplicial in the subgraph induced by the vertices $\{v_1, ..., v_i\}$.

Chordal Graph

Definition

A Chordal Graph is one in which all cycles of four or more vertices have a chord, which is an edge that is not part of the cycle but connects two vertices of the cycle.

Definition

For any two vertices $x, y \in G$ such that $(x, y) \in E$, a $x - y$ separator is a set $S \subset V$ such that the graph $G - S$ has at least two disjoint connected components, one of which contains x and another of which contains y.

Chordal Graph

Definition

A Chordal Graph is one in which all cycles of four or more vertices have a chord, which is an edge that is not part of the cycle but connects two vertices of the cycle.

Definition

For any two vertices $x, y \in G$ such that $(x, y) \in E$, a $x - y$ separator is a set $S \subset V$ such that the graph $G - S$ has at least two disjoint connected components, one of which contains x and another of which contains y.

Chordal Graph

Theorem

For a graph G on n vertices, the following conditions are equivalent:

1. G has a perfect elimination ordering.
2. G is chordal.
3. If H is any induced subgraph of G and S is a vertex separator of H of minimal size, S's vertices induce a clique.

Chordal Graph

Theorem

For a graph G on n vertices, the following conditions are equivalent:

1. G has a perfect elimination ordering.
2. G is chordal.
3. If H is any induced subgraph of G and S is a vertex separator of H of minimal size, S's vertices induce a clique.

Chordal Graph

Theorem

For a graph G on n vertices, the following conditions are equivalent:

1. G has a perfect elimination ordering.
2. G is chordal.
3. If H is any induced subgraph of G and S is a vertex separator of H of minimal size, S's vertices induce a clique.

Chordal Graph

Theorem

For a graph G on n vertices, the following conditions are equivalent:

1. G has a perfect elimination ordering.
2. G is chordal.
3. If H is any induced subgraph of G and S is a vertex separator of H of minimal size, S's vertices induce a clique.

Cinvestav

Maximal Clique

Definition

A maximal clique is a clique that cannot be extended by including one more adjacent vertex, meaning it is not a subset of a larger clique.

Maximal Clique

Definition

A maximal clique is a clique that cannot be extended by including one more adjacent vertex, meaning it is not a subset of a larger clique.

We have the the following Claims

1. A chordal graph with N vertices can have no more than N maximal cliques.
2. Given a chordal graph with $G = (V, E)$, where $|V| = N$, there exists an algorithm to find all the maximal cliques of G which takes no more than $O(N^4)$ time.

Maximal Clique

Definition

A maximal clique is a clique that cannot be extended by including one more adjacent vertex, meaning it is not a subset of a larger clique.

We have the the following Claims

1. A chordal graph with N vertices can have no more than N maximal cliques.
2. Given a chordal graph with $G = (V, E)$, where $|V| = N$, there exists an algorithm to find all the maximal cliques of G which takes no more than $O\left(N^4\right)$ time.

Cinvestav

Elimination Clique

Definition (Elimination Clique)

Given a chordal graph G, and an elimination ordering for G which does not add any edges.

- Suppose node i (Assuming a Labeling) is eliminated in some step of the elimination algorithm, then the clique consisting of the node i along with its neighbors during the elimination step (which must be fully connected since elimination does not add edges) is called an elimination clique.

Formally

Suppose node i is eliminated in the k^{th} step of the algorithm, and let $G^{(k)}$ be the graph just before the k^{th} elimination step. Then, the clique $C_i = \{i\} \cup N^{(k)}(i)$ where $N^{(k)}(i)$ is the neighbor of i in the Graph $G^{(k)}$.

Elimination Clique

Definition (Elimination Clique)

Given a chordal graph G, and an elimination ordering for G which does not add any edges.

- Suppose node i (Assuming a Labeling) is eliminated in some step of the elimination algorithm, then the clique consisting of the node i along with its neighbors during the elimination step (which must be fully connected since elimination does not add edges) is called an elimination clique.

Formally

Suppose node i is eliminated in the k^{th} step of the algorithm, and let $G^{(k)}$ be the graph just before the k^{th} elimination step. Then, the clique $C_i = \{i\} \cup N^{(k)}(i)$ where $N^{(k)}(i)$ is the neighbor of i in the Graph $G^{(k)}$.

From This

Theorem

Given a chordal graph and an elimination ordering which does not add any edges. Let $\mathcal{C}$ be the set of maximal cliques in the chordal graph, and let $\mathcal{C}_e = (\cup_{i \in V} C_i)$ be the set of elimination cliques obtained from this elimination ordering. Then, $\mathcal{C} \subseteq \mathcal{C}_e$. In other words, every maximal clique is also an elimination clique for this particular ordering.

Something Notable

The theorem proves the 2^{nd} claims given earlier. Firstly, it shows that a chordal graph cannot have more than N maximal cliques, since we have only N elimination cliques.

It is more

It gives us an efficient algorithm for finding these N maximal cliques.

- Simply go over each elimination clique and check whether it is maximal.

From This

Theorem

Given a chordal graph and an elimination ordering which does not add any edges. Let $\mathcal{C}$ be the set of maximal cliques in the chordal graph, and let $\mathcal{C}_e = (\cup_{i \in V} C_i)$ be the set of elimination cliques obtained from this elimination ordering. Then, $\mathcal{C} \subseteq \mathcal{C}_e$. In other words, every maximal clique is also an elimination clique for this particular ordering.

Something Notable

The theorem proves the 2^{nd} claims given earlier. Firstly, it shows that a chordal graph cannot have more than N maximal cliques, since we have only N elimination cliques.

It is more

It gives us an efficient algorithm for finding these N maximal cliques.

- Simply go over each elimination clique and check whether it is maximal.

From This

Theorem

Given a chordal graph and an elimination ordering which does not add any edges. Let $\mathcal{C}$ be the set of maximal cliques in the chordal graph, and let $\mathcal{C}_e = (\cup_{i \in V} C_i)$ be the set of elimination cliques obtained from this elimination ordering. Then, $\mathcal{C} \subseteq \mathcal{C}_e$. In other words, every maximal clique is also an elimination clique for this particular ordering.

Something Notable

The theorem proves the 2^{nd} claims given earlier. Firstly, it shows that a chordal graph cannot have more than N maximal cliques, since we have only N elimination cliques.

It is more

It gives us an efficient algorithm for finding these N maximal cliques.

- Simply go over each elimination clique and check whether it is maximal.

Therefore

Even with a brute force approach

It will not take more than $|\mathcal{C}_e|^2 \times D = O\left(N^3\right)$ with $D = \max_{C \in \mathcal{C}} |C|$.

Because

Since both clique size and number of elimination cliques is bounded by N

Observation

The maximum clique problem, which is NP-hard on general graphs, is easy on chordal graphs.

Therefore

Even with a brute force approach

It will not take more than $|\mathcal{C}_e|^2 \times D = O\left(N^3\right)$ with $D = \max_{C \in \mathcal{C}} |C|$.

Because

Since both clique size and number of elimination cliques is bounded by N

Observation

The maximum clique problem, which is NP-hard on general graphs, is easy on chordal graphs.

Therefore

Even with a brute force approach

It will not take more than $|\mathcal{C}_e|^2 \times D = O\left(N^3\right)$ with $D = \max_{C \in \mathcal{C}} |C|$.

Because

Since both clique size and number of elimination cliques is bounded by N

Observation

The maximum clique problem, which is NP-hard on general graphs, is easy on chordal graphs.

Outline

Cinvestav

We have the following definitions

Definition

The following are equivalent to the statement “G is a tree”

1. G is a connected, acyclic graph over N nodes.
2. G is a connected graph over N nodes with $N-1$ edges.
3. G is a minimal connected graph over N nodes.
4. (**Important**) G is a graph over N nodes, such that for any 2 nodes i and j in G , with $i \neq j$, there is a unique path from i to j in G.

Cinvestav

We have the following definitions

Definition

The following are equivalent to the statement "G is a tree"

1. G is a connected, acyclic graph over N nodes.
2. G is a connected graph over N nodes with $N-1$ edges.
3. G is a minimal connected graph over N nodes.
4. **(Important)** G is a graph over N nodes, such that for any 2 nodes i and j in G, with $i \neq j$, there is a unique path from i to j in G.

Theorem

For any graph $G=(V,E)$, the following statements are equivalent:

1. G has a junction tree.
2. G is chordal.

We have the following definitions

Definition

The following are equivalent to the statement “G is a tree”

1. G is a connected, acyclic graph over N nodes.
2. G is a connected graph over N nodes with $N-1$ edges.
3. G is a minimal connected graph over N nodes.
4. (**Important**) G is a graph over N nodes, such that for any 2 nodes i and j in G , with $i \neq j$, there is a unique path from i to j in G.

Theorem

For any graph $G = (V, E)$, the following statements are equivalent:

1. G has a junction tree.
2. G is chordal.

Cinvestav

We have the following definitions

Definition

The following are equivalent to the statement "G is a tree"

1. G is a connected, acyclic graph over N nodes.
2. G is a connected graph over N nodes with $N-1$ edges.
3. G is a minimal connected graph over N nodes.
4. **(Important)** G is a graph over N nodes, such that for any 2 nodes i and j in G, with $i \neq j$, there is a unique path from i to j in G.

Theorem

For any graph $G=(V,E)$, the following statements are equivalent:

1. G has a junction tree.
2. G is chordal.

Cinvestav

We have the following definitions

Definition

The following are equivalent to the statement "G is a tree"

1. G is a connected, acyclic graph over N nodes.
2. G is a connected graph over N nodes with $N-1$ edges.
3. G is a minimal connected graph over N nodes.
4. (**Important**) G is a graph over N nodes, such that for any 2 nodes i and j in G , with $i \neq j$, there is a unique path from i to j in G.

Theorem

For any graph $G=(V,E)$, the following statements are equivalent:

1. G has a junction tree.
2. G is chordal.

We have the following definitions

Definition

The following are equivalent to the statement "G is a tree"

1. G is a connected, acyclic graph over N nodes.
2. G is a connected graph over N nodes with $N-1$ edges.
3. G is a minimal connected graph over N nodes.
4. (**Important**) G is a graph over N nodes, such that for any 2 nodes i and j in G , with $i \neq j$, there is a unique path from i to j in G.

Theorem

For any graph $G = (V, E)$, the following statements are equivalent:

1. G has a junction tree.
2. G is chordal.

Cinvestav

We have the following definitions

Definition

The following are equivalent to the statement "G is a tree"

1. G is a connected, acyclic graph over N nodes.
2. G is a connected graph over N nodes with $N-1$ edges.
3. G is a minimal connected graph over N nodes.
4. (**Important**) G is a graph over N nodes, such that for any 2 nodes i and j in G , with $i \neq j$, there is a unique path from i to j in G.

Theorem

For any graph $G = (V, E)$, the following statements are equivalent:

1. G has a junction tree.
2. G is chordal.

Outline

Cinvestav

Definition

Junction Tree

Given a graph $G = (V, E)$, a graph $G' = (V', E')$ is said to be a Junction Tree for G, iff:

1. The nodes of G' are the maximal cliques of G (i.e. G' is a clique graph of G.)
2. G' is a tree.
3. Running Intersection Property / Junction Tree Property:
    1. For each $v \in V$, define G'_v to be the induced subgraph of G' consisting of exactly those nodes which correspond to maximal cliques of G that contain v. Then G'_v must be a connected graph.

Cinvestav

Outline

Cinvestav

Step 1

Given a DAG $G = (V, E)$ and $|V| = N$

Chordalize the graph using the elimination algorithm with an arbitrary elimination ordering, if required.

Step 1

Given a DAG $G = (V, E)$ and $|V| = N$

Chordalize the graph using the elimination algorithm with an arbitrary elimination ordering, if required.

For this, you can use the following greedy algorithm

Given a list of nodes:

1. Is the vertex simplicial? If it is not, make it simplicial.
2. If not remove it from the list.

Step 1

Given a DAG $G=(V,E)$ and $|V|=N$

Chordalize the graph using the elimination algorithm with an arbitrary elimination ordering, if required.

For this, you can use the following greedy algorithm

Given a list of nodes:

1. Is the vertex simplicial? If it is not, make it simplicial.
2. If not remove it from the list.

Step 1

Given a DAG $G = (V, E)$ and $|V| = N$

Chordalize the graph using the elimination algorithm with an arbitrary elimination ordering, if required.

For this, you can use the following greedy algorithm

Given a list of nodes:

1. Is the vertex simplicial? If it is not, make it simplicial.
2. If not remove it from the list.

Step 1

Another way

1. By the Moralization Procedure.
2. Triangulate the moral graph.

Moralization Procedure

1. Add edges between all pairs of nodes that have a common child.

Step 1

Another way

1. By the Moralization Procedure.
2. Triangulate the moral graph.

Moralization Procedure

1. Add edges between all pairs of nodes that have a common child.
2. Make all edges in the graph undirected.

Step 1

Another way

1. By the Moralization Procedure.
2. Triangulate the moral graph.

Moralization Procedure

1. Add edges between all pairs of nodes that have a common child.
2. Make all edges in the graph undirected.

Triangulate the moral graph

An undirected graph is triangulated if every cycle of length greater than 3 possesses a chord.

Step 2

Find the maximal cliques in the chordal graph

List the N Cliques

- $\left(\{v_N\} \cup N\left(v_N\right)\right) \cap \{v_1, ..., v_N\}$
- $\left(\{v_{N-1}\} \cup N\left(v_{N-1}\right)\right) \cap \{v_1, ..., v_{N-1}\}$
- $\cdots$
- $\{v_1\}$

Note: If the graph is Chordal this is not necessary because all the cliques are maximal.

Step 3

Compute the separator sets for each pair of maximal cliques and construct a weighted clique graph

For each pair of maximal cliques (C_i, C_j) in the graph

- We check whether they posses any common variables.

Cinvestav

Step 3

Compute the separator sets for each pair of maximal cliques and construct a weighted clique graph

For each pair of maximal cliques (C_i, C_j) in the graph

- We check whether they posses any common variables.

If yes, we designate a separator set

Between these 2 cliques as $S_{ij} = C_i \cap C_j$.

- Nodes are the Cliques
- Edges (C_i, C_j) are added with weight $|C_i \cap C_j|$ if $|C_i \cap C_j| > 0$.

Step 3

Compute the separator sets for each pair of maximal cliques and construct a weighted clique graph

For each pair of maximal cliques (C_i, C_j) in the graph

- We check whether they posses any common variables.

If yes, we designate a separator set

Between these 2 cliques as $S_{ij} = C_i \cap C_j$.

Then, we compute these separators trees

We build a clique graph:

- Nodes are the Cliques.
- Edges (C_i, C_j) are added with weight $|C_i \cap C_j|$ if $|C_i \cap C_j| > 0$.

Step 3

Compute the separator sets for each pair of maximal cliques and construct a weighted clique graph

For each pair of maximal cliques (C_i, C_j) in the graph

- We check whether they posses any common variables.

If yes, we designate a separator set

Between these 2 cliques as $S_{ij} = C_i \cap C_j$.

Then, we compute these separators trees

We build a clique graph:

- Nodes are the Cliques.
- Edges (C_i, C_j) are added with weight $|C_i \cap C_j|$ if $|C_i \cap C_j| > 0$.

Step 3

Compute the separator sets for each pair of maximal cliques and construct a weighted clique graph

For each pair of maximal cliques (C_i, C_j) in the graph

- We check whether they posses any common variables.

If yes, we designate a separator set

Between these 2 cliques as $S_{ij} = C_i \cap C_j$.

Then, we compute these separators trees

We build a clique graph:

- Nodes are the Cliques.
- Edges (C_i, C_j) are added with weight $|C_i \cap C_j|$ if $|C_i \cap C_j| > 0$.

Step 3

Compute the separator sets for each pair of maximal cliques and construct a weighted clique graph

For each pair of maximal cliques (C_i, C_j) in the graph

- We check whether they posses any common variables.

If yes, we designate a separator set

Between these 2 cliques as $S_{ij} = C_i \cap C_j$.

Then, we compute these separators trees

We build a clique graph:

- Nodes are the Cliques.
- Edges (C_i, C_j) are added with weight $|C_i \cap C_j|$ if $|C_i \cap C_j| > 0$.

Step 3

This step can be implemented quickly in practice using a hash table

Running Time: $O\left(|\mathcal{C}|^2 D\right) = O\left(N^2 D\right)$

Step 4

Compute a maximum-weight spanning tree on the weighted clique graph to obtain a junction tree

You can us for this the Kruskal and Prim for Maximum Weight Graph

We will give Kruskal's algorithm

For finding the maximum-weight spanning tree

Step 4

Compute a maximum-weight spanning tree on the weighted clique graph to obtain a junction tree

You can us for this the Kruskal and Prim for Maximum Weight Graph

We will give Kruskal's algorithm

For finding the maximum-weight spanning tree

Step 4

Maximal Kruskal's algorithm

Initialize an edgeless graph $\mathcal{T}$ with nodes that are all the maximal cliques in our chordal graph.

Then

We will add edges to $\mathcal{T}$ until it becomes a junction tree.

Sort the m edges e_i in our clique graph from step 3 by weight w_i

We have for $e_1, e_2, ..., e_m$ with $w_1 \geq w_2 \geq \cdots \geq w_1$

Step 4

Maximal Kruskal's algorithm

Initialize an edgeless graph $\mathcal{T}$ with nodes that are all the maximal cliques in our chordal graph.

Then

We will add edges to $\mathcal{T}$ until it becomes a junction tree.

Sort the m edges e_i in our clique graph from step 3 by weight w_i

We have for $e_1, e_2, ..., e_m$ with $w_1 \geq w_2 \geq \cdots \geq w_1$

Step 4

Maximal Kruskal's algorithm

Initialize an edgeless graph $\mathcal{T}$ with nodes that are all the maximal cliques in our chordal graph.

Then

We will add edges to $\mathcal{T}$ until it becomes a junction tree.

Sort the m edges e_i in our clique graph from step 3 by weight w_i

We have for $e_1, e_2, ..., e_m$ with $w_1 \geq w_2 \geq \cdots \geq w_1$

Cinvestav

Step 4

For $i = 1, 2, ..., m$

1. Add edge e_i to $\mathcal{T}$ if it does not introduce a cycle.
2. If $|\mathcal{C}| - 1$ edges have been added, quit.

Running Time given that $|E| = O\left(|\mathcal{C}|^2\right)$

$$O\left(|\mathcal{C}|^2 \log |\mathcal{C}|^2\right) = O\left(|\mathcal{C}|^2 \log |\mathcal{C}|\right) = O\left(N^2 \log N\right)$$

Cinvestav

Step 4

For $i = 1, 2, ..., m$

1. Add edge e_i to $\mathcal{T}$ if it does not introduce a cycle.
2. If $|\mathcal{C}| - 1$ edges have been added, quit.

Running Time given that $|E| = O\left(|\mathcal{C}|^2\right)$

$$O\left(|\mathcal{C}|^2 \log |\mathcal{C}|^2\right) = O\left(|\mathcal{C}|^2 \log |\mathcal{C}|\right) = O\left(N^2 \log N\right)$$

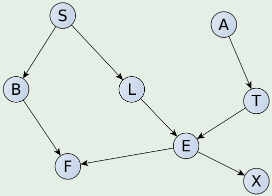
Outline

Cinvestav

Outline

How do you build a Junction Tree?

Given a General DAG

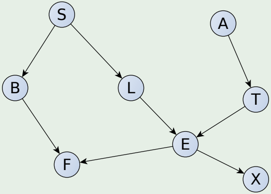


Build a Chordal Graph

- Moral Graph – marry common parents and remove arrows.

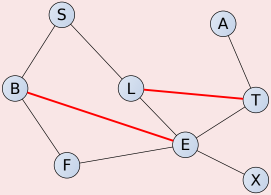
How do you build a Junction Tree?

Given a General DAG



Build a Chordal Graph

- Moral Graph – marry common parents and remove arrows.



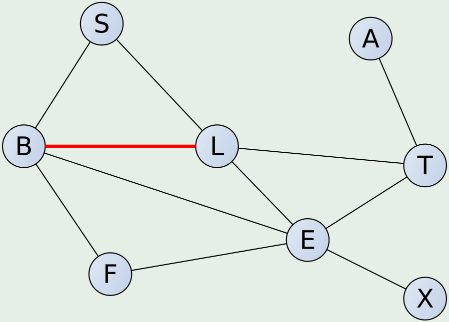
Outline

Cinvestav

How do you build a Junction Tree?

Triangulate the moral graph

- An undirected graph is triangulated if every cycle of length greater than 3 possesses a chord.



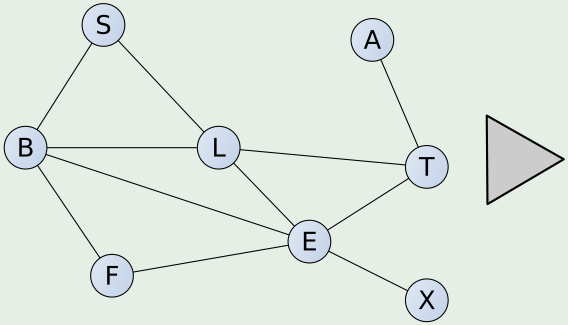
Outline

Cinvestav

Listing of Cliques

Identify the Cliques

- A clique is a subset of nodes which is **complete** (i.e. there is an edge between every pair of nodes) and **maximal**.

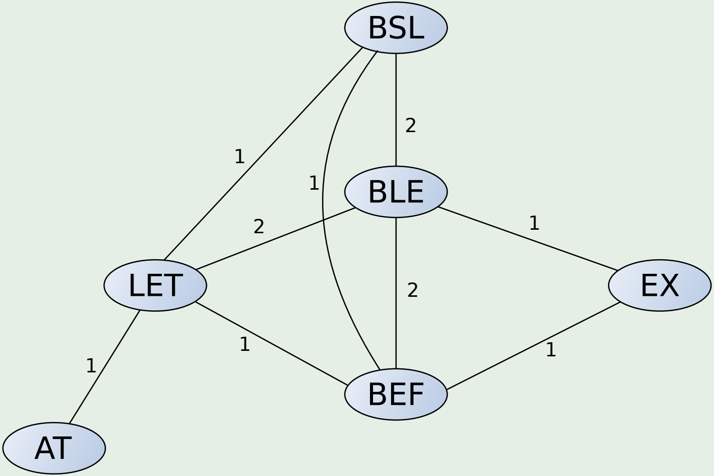


{B,S,L}
{B,L,E}
{B,E,F}
{L,E,T}
{A,T}
{E,X}

Build the Clique Graph

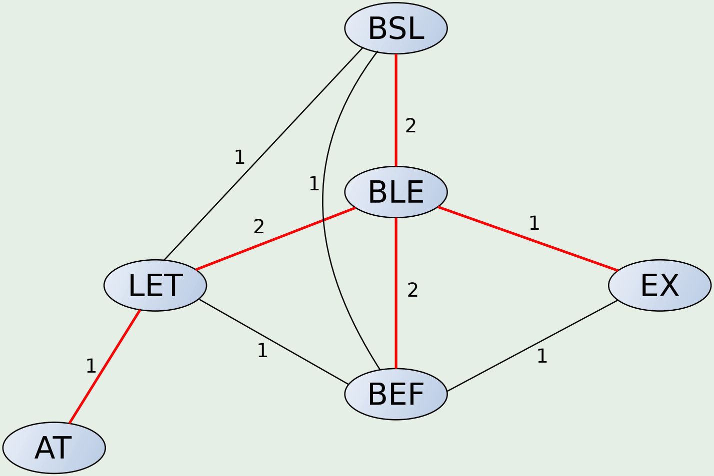
Clique Graph

- Add an edge between C_j and C_i with weight $|C_i \cap C_j| > 0$



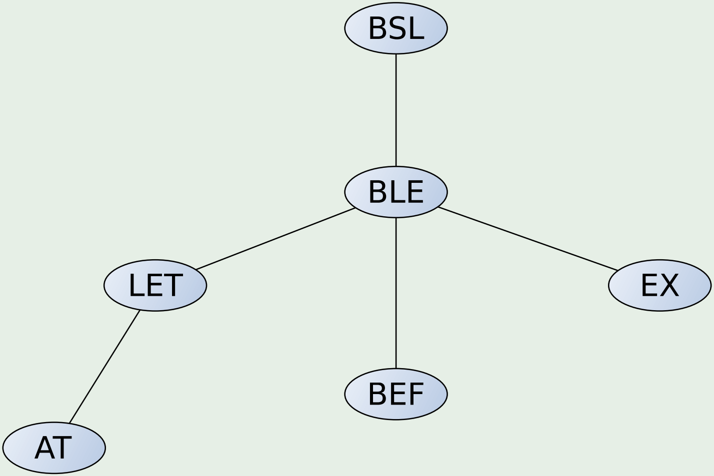
Getting The Junction Tree

Run the Maximum Kruskal's Algorithm



Getting The Junction Tree

Finally



Outline

Cinvestav

Potential Representation for the Junction Tree

Something Notable

- The joint probability distribution can now be represented in terms of potential functions, ϕ.
  - This is defined in each clique and each separator

Thus

$$P(x) = \frac{\prod_{i=1}^{n} \phi_C(x_{c_i})}{\prod_{j=1}^{m} \phi_S(x_{s_j})}$$

where $x = (x_{c_1}, \ldots, x_{c_n})$ and each variable x_{c_i} correspond to a clique and x_{s_j} correspond to a separator.

Basic idea is to represent probability distribution corresponding to any graph as a product of clique potentials

$$P(x) = \frac{1}{Z} \prod_{i=1}^{n} \phi_C(x_{c_i})$$

Potential Representation for the Junction Tree

Something Notable

- The joint probability distribution can now be represented in terms of potential functions, ϕ.
  - This is defined in each clique and each separator

Thus

$$P(\boldsymbol{x}) = \frac{\prod_{i=1}^{n} \phi_C(x_{c_i})}{\prod_{j=1}^{m} \phi_S(x_{s_j})}$$

where $\boldsymbol{x} = (x_{c_1}, ..., x_{c_n})$ and each variable x_{c_i} correspond to a clique and x_{s_j} correspond to a separator.

Basic idea is to represent probability distribution corresponding to any graph as a product of clique potentials

$$P(\boldsymbol{x}) = \frac{1}{Z} \prod_{i=1}^{n} \phi_C(x_{c_i})$$

Potential Representation for the Junction Tree

Something Notable

- The joint probability distribution can now be represented in terms of potential functions, ϕ.
  - This is defined in each clique and each separator

Thus

$$P(\boldsymbol{x}) = \frac{\prod_{i=1}^{n} \phi_C(x_{c_i})}{\prod_{j=1}^{m} \phi_S(x_{s_j})}$$

where $\boldsymbol{x} = (x_{c_1}, ..., x_{c_n})$ and each variable x_{c_i} correspond to a clique and x_{s_j} correspond to a separator.

Basic idea is to represent probability distribution corresponding to any graph as a product of clique potentials

$$P(\boldsymbol{x}) = \frac{1}{Z} \prod_{i=1}^{n} \phi_C(x_{c_i})$$

Then

Main idea

The idea is to transform one representation of the joint distribution to another in which for each clique, c, the potential function gives the marginal distribution for the variables in c, i.e.

$$\phi_C\left(x_c\right) = P\left(x_c\right)$$

This will also apply for each separator, s.

Now, Initialization

To initialize the potential functions

1. Set all potentials to unity
2. For each variable, x_i, select one node in the junction tree (i.e. one clique) containing both that variable and its parents, $pa(x_i)$, in the original DAG.
3. Multiply the potential by $P(x_i|pa(x_i))$

Now, Initialization

To initialize the potential functions

1. Set all potentials to unity
2. For each variable, x_i, select one node in the junction tree (i.e. one clique) containing both that variable and its parents, $pa(x_i)$, in the original DAG.
3. Multiply the potential by $P(x_i|pa(x_i))$

Now, Initialization

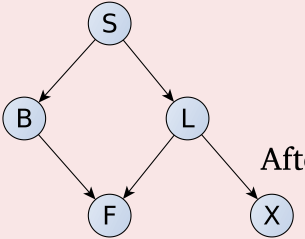
To initialize the potential functions

1. Set all potentials to unity
2. For each variable, x_i, select one node in the junction tree (i.e. one clique) containing both that variable and its parents, $pa(x_i)$, in the original DAG.
3. Multiply the potential by $P\left(x_i | pa\left(x_i\right)\right)$

Cinvestav

Then

For example, we have at the beginning $\phi_{BSL} = \phi_{BFL} = \phi_{LX} = 1$,then



After Initialization

$$\phi_{BSL} = P(b|s)\,P(l|s)\,P(s)$$

$$\phi_{BFL} = P(f|b,l)$$

$$\phi_{LX} = P(x|l)$$

Cinvestav

Outline

Cinvestav

Propagating Information in a Junction Tree

Passing Information using the separators

Passing information from one clique C_1 to another C_2 via the separator in between them, S_0, requires two steps

First Step

Obtain a new potential for S_0 by marginalizing out the variables in C_1 that are not in S_0:

$$\phi^*_{S_0} = \sum_{C_1 - S_0} \phi_{C_1}$$

Propagating Information in a Junction Tree

Passing Information using the separators

Passing information from one clique C_1 to another C_2 via the separator in between them, S_0, requires two steps

First Step

Obtain a new potential for S_0 by marginalizing out the variables in C_1 that are not in S_0:

$$\phi_{S_0}^{*} = \sum_{C_1 - S_0} \phi_{C_1}$$

Propagating Information in a Junction Tree

Second Step

Obtain a new potential for C_2:

$$\phi^*_{C_2} = \phi_{C_2}\lambda_{S_0}$$

Where

$$\lambda_{S_0} = \frac{\phi^*_{S_0}}{\phi_{S_0}}$$

Propagating Information in a Junction Tree

Second Step

Obtain a new potential for C_2:

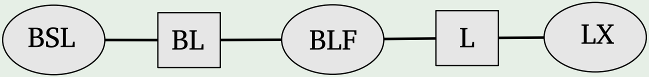
$$\phi^*_{C_2} = \phi_{C_2} \lambda_{S_0}$$

Where

$$\lambda_{S_0} = \frac{\phi^*_{S_0}}{\phi_{S_0}}$$

An Example

Consider a flow from the clique {B,S,L} to {B,L,F}



Outline

Cinvestav

An Example

Initial representation

$\phi_{BSL} = P(B\|S)\,P(L\|S)\,P(S)$		
	l_1	l_2
s_1, b_1	0.00015	0.04985
s_1, b_2	0.00045	0.14955
s_2, b_1	0.000002	0.039998
s_2, b_2	0.000038	0.759962

$\phi_{BL} = 1$		
	l_1	l_2
b_1	1	1
b_2	1	1

$\phi_{BLF} = P(F\|B,L)\,P(B)\,P(L) = P(F\|B,L)$		
	l_1	l_2
f_1, b_1	0.75	0.1
f_1, b_2	0.5	0.05
f_2, b_1	0.25	0.9
f_2, b_2	0.5	0.95

An Example

After Flow

$\phi_{BSL} = P(B\|S)\,P(L\|S)\,P(S)$		
	l_1	l_2
s_1, b_1	0.00015	0.04985
s_1, b_2	0.00045	0.14955
s_2, b_1	0.000002	0.039998
s_2, b_2	0.000038	0.759962

$\phi_{BL} = 1$		
	l_1	l_2
b_1	0.000152	0.089848
b_2	0.000488	0.909512

$\phi_{BLF} = P(F\|B, L)$		
	l_1	l_2
f_1, b_1	0.000114	0.0089848
f_1, b_2	0.000244	0.0454756
f_2, b_1	0.000038	0.0808632
f_2, b_2	0.000244	0.8640364

Now Introduce Evidence

We have

A flow from the clique {B,S,L} to {B,L,F}, but this time we he information that Joe is a smoker, $S = s_1$.

Incorporation of Evidence

$\phi_{BSL} = P(B|S)\,P(L|S)\,P(S)$

	l_1	l_2
s_1, b_1	0.00015	0.04985
s_1, b_2	0.00045	0.14955
s_2, b_1	0	0
s_2, b_2	0	0

$\phi_{BL} = 1$

	l_1	l_2
b_1	1	1
b_2	1	1

$\phi_{BLF} = P(F|B, L)$

	l_1	l_2
f_1, b_1	0.75	0.1
f_1, b_2	0.5	0.05
f_2, b_1	0.25	0.9
f_2, b_2	0.5	0.95

Now Introduce Evidence

We have

A flow from the clique {B,S,L} to {B,L,F}, but this time we he information that Joe is a smoker, $S = s_1$.

Incorporation of Evidence

$\phi_{BSL} = P(B|S)\,P(L|S)\,P(S)$

	l_1	l_2
s_1, b_1	0.00015	0.04985
s_1, b_2	0.00045	0.14955
s_2, b_1	0	0
s_2, b_2	0	0

$\phi_{BL} = 1$

	l_1	l_2
b_1	1	1
b_2	1	1

$\phi_{BLF} = P(F|B, L)$

	l_1	l_2
f_1, b_1	0.75	0.1
f_1, b_2	0.5	0.05
f_2, b_1	0.25	0.9
f_2, b_2	0.5	0.95

An Example

After Flow

$\phi_{BSL} = P(B\|S)P(L\|S)P(S)$		
	l_1	l_2
s_1, b_1	0.00015	0.04985
s_1, b_2	0.00045	0.14955
s_2, b_1	0	0
s_2, b_2	0	0

$\phi_{BL} = 1$		
	l_1	l_2
b_1	0.00015	0.04985
b_2	0.00045	0.14955

$\phi_{BLF} = P(F\|B, L)$		
	l_1	l_2
f_1, b_1	0.0001125	0.004985
f_1, b_2	0.000245	0.0074775
f_2, b_1	0.0000375	0.044865
f_2, b_2	0.000255	0.1420725

Outline

The Full Propagation

Two phase propagation (Jensen et al, 1990)

1. Select an arbitrary clique, C_0
2. Collection Phase – flows passed from periphery to C_0
3. Distribution Phase – flows passed from C_0 to periphery

The Full Propagation

Two phase propagation (Jensen et al, 1990)

1. Select an arbitrary clique, C_0
2. Collection Phase – flows passed from periphery to C_0
3. Distribution Phase – flows passed from C_0 to periphery

Cinvestav

The Full Propagation

Two phase propagation (Jensen et al, 1990)

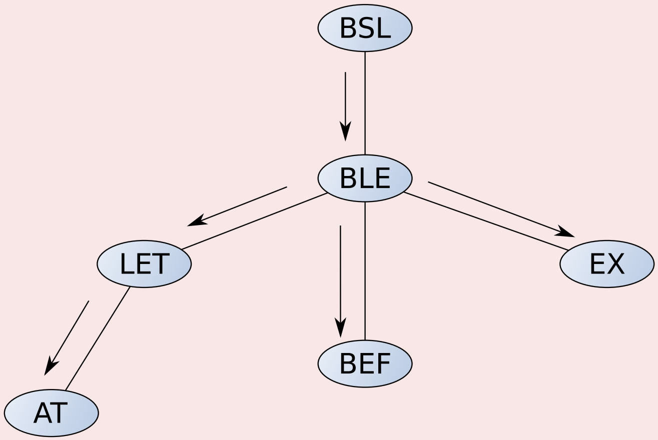
1. Select an arbitrary clique, C_0
2. Collection Phase – flows passed from periphery to C_0
3. Distribution Phase – flows passed from C_0 to periphery

Outline

Cinvestav

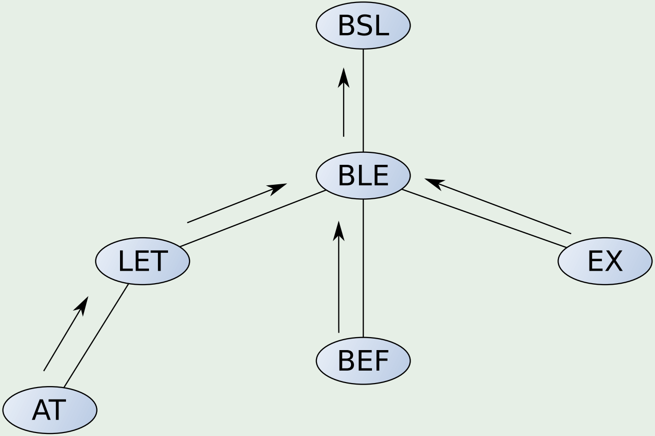
Example

Distribution



Example

Collection



The Full Propagation

After the two propagation phases have been carried out

- The Junction tree will be in equilibrium with each clique containing the joint probability distribution for the variables it contains.
- Marginal probabilities for individual variables can then be obtained from the cliques.

Cinvestav

The Full Propagation

After the two propagation phases have been carried out

- The Junction tree will be in equilibrium with each clique containing the joint probability distribution for the variables it contains.
- Marginal probabilities for individual variables can then be obtained from the cliques.

Now, some evidence E can be included before propagation

- By selecting a clique for each variable for which evidence is available.
- The potential for the clique is then set to 0 for any configuration which differs from the evidence.

The Full Propagation

After the two propagation phases have been carried out

- The Junction tree will be in equilibrium with each clique containing the joint probability distribution for the variables it contains.
- Marginal probabilities for individual variables can then be obtained from the cliques.

Now, some evidence E can be included before propagation

- By selecting a clique for each variable for which evidence is available.
- The potential for the clique is then set to 0 for any configuration which differs from the evidence.

Cinvestav

The Full Propagation

After the two propagation phases have been carried out

- The Junction tree will be in equilibrium with each clique containing the joint probability distribution for the variables it contains.
- Marginal probabilities for individual variables can then be obtained from the cliques.

Now, some evidence E can be included before propagation

- By selecting a clique for each variable for which evidence is available.
- The potential for the clique is then set to 0 for any configuration which differs from the evidence.

The Full Propagation

After propagation the result will be

$$P\left(x, E\right) = \frac{\prod_{c \in C} \phi_c\left(x_c, E\right)}{\prod_{s \in S} \phi_s\left(x_s, E\right)}$$

After normalization

$$P\left(x | E\right) = \frac{\prod_{c \in C} \phi_c\left(x_c | E\right)}{\prod_{s \in S} \phi_s\left(x_s | E\right)}$$

The Full Propagation

After propagation the result will be

$$P\left(x, E\right) = \frac{\prod_{c \in C} \phi_c \left(x_c, E\right)}{\prod_{s \in S} \phi_s \left(x_s, E\right)}$$

After normalization

$$P\left(x|E\right) = \frac{\prod_{c \in C} \phi_c \left(x_c|E\right)}{\prod_{s \in S} \phi_s \left(x_s|E\right)}$$