Part I
TWO MESSAGE PASSING ALGORITHMS
**Sum-Product Message Passing Algorithm**

**Clique tree**

- **Claim:** for each clique \( C_i \):
  - \( \pi_{i}[C_i] = P(C_i) \)
  - **Variable elimination**, treating \( C_i \) as a root clique

- **Compute** \( P(X) \)
  - Find belief \( \pi \) of a clique that contains \( X \) and eliminate other RVs.
  - If \( X \) appears in multiple cliques, they must agree

---

**Clique Tree Calibration**

- A clique tree with potentials \( \pi_i[C_i] \) is said to be **calibrated** if for all neighboring cliques \( C_i \) and \( C_j \):

  \[
  \sum_{C_j \in S} \pi_i[C_i] = \sum_{C_j \in S} \pi_j[C_j] \quad \text{“Sepset belief”}
  \]

- **Key advantage** the clique tree inference algorithm
  - Computes marginal distributions for all variables \( P(X_1), \ldots, P(X_n) \) using only twice the computation of the upward pass in the same tree.
Calibrated Clique Tree as a Distribution

- At convergence of the clique tree algorithm, we have that:

$$P_{\theta}(X) = \frac{\prod_{i \in C_i} \pi_i(C_i)}{\prod_{(i \in C_i) \cup \mathcal{F}} \mu_i(S_{ij})}$$

- Proof:

$$\mu_i(S_{ij}) = \sum_{C \in \mathcal{C}_i} \pi(C) = \sum_{C \in \mathcal{C}_i} \pi'(C) \prod_{k \in \mathcal{N}_j(i)} \delta_k$$

$$= \sum_{C \in \mathcal{C}_i} \pi'(C) \delta_j(S_{ij}) \prod_{k \in \mathcal{N}_j(i)} \delta_k$$

$$= \delta_j(S_{ij}) \prod_{k \in \mathcal{N}_j(i)} \delta_k$$

- Clique tree invariant: The clique beliefs \(\pi\)'s and sepset beliefs \(\mu\)'s provide a re-parameterization of the joint distribution, one that directly reveals the marginal distributions.

Distribution of Calibrated Tree

- For calibrated tree

Bayesian network

\(P(C|B) = \frac{P(B,C)}{P(B)} = \frac{\pi_3[B,C]}{\pi_2[B]}\)

Joint distribution can thus be written as

$$P(A,B,C) = P(A,B)P(C|B) = \frac{\pi_2[A,B] \pi_3[B,C]}{\mu_2[B]}$$

Clique tree invariant

$$P_{\theta}(X) = \frac{\prod_{i \in C_i} \pi_i}{\prod_{(i \in C_i) \cup \mathcal{F}} \mu_i}$$
An alternative approach for message passing in clique trees?

Message Passing: Belief Propagation

- Recall the clique tree calibration algorithm
  - Upon calibration the final potential (belief) at $i$ is:
    $\pi_i = \pi_i \prod_{k \in N_i} \delta_{k \rightarrow i}$
  - A message from $i$ to $j$ sums out the non-sepset variables from the product of initial potential and all messages except for the one from $j$ to $i$:
    $\delta_{i \rightarrow j} = \sum_{C \subseteq S_i \backslash \{j\}} \pi_i \prod_{k \in N_i \backslash \{j\}} \delta_{k \rightarrow i}$
  - Can also be viewed as multiplying all messages and dividing by the message from $j$ to $i$:
    $\delta_{i \rightarrow j} = \sum_{C \subseteq S_i \backslash \{j\}} \pi_i \prod_{k \in N_i \backslash \{j\}} \delta_{k \rightarrow i} \delta_{j \rightarrow i}^{-1}$

- "Sepset belief"

- Forms a basis of an alternative way of computing messages
Message Passing: Belief Propagation

Based on the observation above,

- Different message passing scheme, belief propagation
- Each clique $C_i$ maintains its fully updated beliefs $\pi_i$
  - product of initial clique potentials $\pi_i^0$ and messages from neighbors $\delta_{i\rightarrow j}$
- Each sepset also maintains its belief $\mu_{i,j}$
  - product of the messages in both direction $\delta_{i\rightarrow j}$ and $\delta_{j\rightarrow i}$
- The entire message passing process is executed in an equivalent way in terms of the clique and sepset beliefs – $\pi_i$’s and $\mu_{i,j}$’s.

Basic idea ($\mu_{i,j}=\delta_{i\rightarrow j}\delta_{j\rightarrow i}$)

- Each clique $C_i$ initializes the belief $\pi_i$ as $\pi_i^0$ (=$\prod \phi_i$) and then updates it by multiplying with message updates received from its neighbors.
- Store at each sepset $S_{i,j}$ the previous sepset belief $\mu_{i,j}$ regardless of the direction of the message passed
- When passing a message from $C_i$ to $C_j$ divide the new sepset belief $\sigma_{i,j} = \sum_{x_j} \pi_j$ by previous $\mu_{i,j}$
- Update the clique belief $\pi_i$ by multiplying with $\sigma_{i,j}$

This is called belief update or belief propagation
Message Passing: Belief Propagation

- Initialize the clique tree
  - For each clique $C_i$ set
  - For each edge $C_i - C_j$ set

- While uninformed cliques exist
  - Select $C_i - C_j$
  - Send message from $C_i$ to $C_j$
    - Marginalize the clique over the sepset
    - Update the belief at $C_j$
    - Update the sepset belief at $C_i - C_j$

- Equivalent to the sum-product message passing algorithm?
  - Yes – a simple algebraic manipulation, left as PS#2.

Clique Tree Invariant

- Belief propagation can be viewed as reparameterizing the joint distribution
  - Upon calibration we showed

- Initially this invariant holds since
  - At each update step invariant is also maintained
    - Message only changes $\pi_i$ and $\mu_{ij}$ so most terms remain unchanged
    - We need to show that for new $\pi'$, $\mu'$
    - But this is exactly the message passing step

→ Belief propagation reparameterizes $P_{\phi}$ at each step
Answering Queries

- Posterior distribution queries on variable $X$
  - Sum out irrelevant variables from any clique containing $X$

- Posterior distribution queries on family $X, \text{Pa}(X)$
  - The family preservation property implies that $X, \text{Pa}(X)$ are in the same clique.
  - Sum out irrelevant variables from clique containing $X, \text{Pa}(X)$

- Introducing evidence $(Z = z)$
  - Compute posterior of $X$ where $X$ appears in clique with $Z$
    - Since clique tree is calibrated, multiply clique that contains $X$ and $Z$ with indicator function $I(Z = z)$ and sum out irrelevant variables.
  - Compute posterior of $X$ if $X$ does not share a clique with $Z$
    - Introduce indicator function $I(Z = z)$ into some clique containing $Z$ and propagate messages along path to clique containing $X$
    - Sum out irrelevant factors from clique containing $X$

\[
P_\phi(X) = \prod_{\phi \in \Phi} \phi \quad P_\phi(X, Z = z) = I(Z = z) \prod_{\phi \in \Phi} \phi
\]

So far, we haven’t really discussed how to construct clique trees...
Constructing Clique Trees

- Two basic approaches
  1. Based on variable elimination
  2. Based on direct graph manipulation

- Using variable elimination
  - The execution of a variable elimination algorithm can be associated with a cluster graph.
  - Create a cluster $C_i$ for each factor used during a VE run
  - Create an edge between $C_i$ and $C_j$ when a factor generated by $C_i$ is used directly by $C_j$ (or vice versa)

→ We showed that cluster graph is a tree satisfying the running intersection property and thus it is a legal clique tree

Direct Graph Manipulation

- Goal: construct a tree that is family preserving and obeys the running intersection property
  - The induced graph $I_{F,\alpha}$ is necessarily a chordal graph.
    - The converse holds: any chordal graph can be used as the basis for inference.
    - Any chordal graph can be associated with a clique tree (Theorem 4.12)

Reminder: The induced graph $I_{F,\alpha}$ over factors F and ordering $\alpha$:
  - Union of all of the graphs resulting from the different steps of the variable elimination algorithm.
  - $X_i$ and $X_j$ are connected if they appeared in the same factor throughout the VE algorithm using $\alpha$ as the ordering
Constructing Clique Trees

- The induced graph $I_{F,\alpha}$ is necessarily a chordal graph.
  - Any chordal graph can be associated with a clique tree (Theorem 4.12)

- **Step I: Triangulate** the graph to construct a chordal graph $H$
  - Constructing a chordal graph that subsumes an existing graph $H_0$
  - NP-hard to find a minimum triangulation where the largest clique in the resulting chordal graph has minimum size
  - Exact algorithms are too expensive and one typically resorts to heuristic algorithms. (e.g., node elimination techniques; see K&F 9.4.3.2)

- **Step II: Find cliques** in $H$ and make each a node in the clique tree
  - Finding maximal cliques is NP-hard
  - Can begin with a family, each member of which is guaranteed to be a clique, and then use a greedy algorithm that adds nodes to the clique until it no longer induces a fully connected subgraph.

- **Step III: Construct a tree** over the clique nodes
  - Use maximum spanning tree algorithm on an undirected graph whose nodes are cliques selected above and edge weight is $|C_i \cap C_j|$
  - We can show that resulting graph obeys running intersection → valid clique tree

---

Example

![Diagram showing the process of constructing a clique tree](image-url)
Learning Introduction

- So far, we assumed that the networks were given

- Where do the networks come from?
  - Knowledge engineering with aid of experts
  - Learning: automated construction of networks
    - Learn by examples or instances
Learning Introduction

- **Input**: dataset of instances $D = \{d[1], ... d[m]\}$
- **Output**: Bayesian network

**Measures of success**
- How close is the learned network to the original distribution?
  - Use distance measures between distributions
  - Often hard because we do not have the true underlying distribution
  - Instead, evaluate performance by how well the network predicts new unseen examples ("test data")
- Classification accuracy
- How close is the structure of the network to the true one?
  - Use distance metric between structures
  - Hard because we do not know the true structure
  - Instead, ask whether independencies learned hold in test data

Prior Knowledge

- Prespecified structure
  - Learn only CPDs
- Prespecified variables
  - Learn network structure and CPDs
- Hidden variables
  - Learn hidden variables, structure, and CPDs
- Complete/incomplete data
  - Missing data
  - Unobserved variables
Learning Bayesian Networks

- Four types of problems will be covered

- Data
- Prior information

I. Known Structure, Complete Data

- Goal: Parameter estimation
- Data does not contain missing values
II. Unknown Structure, Complete Data

- **Goal:** Structure learning & parameter estimation
- **Data** does not contain missing values

Input Data

<table>
<thead>
<tr>
<th>X₁</th>
<th>X₂</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁&lt;sup&gt;0&lt;/sup&gt;</td>
<td>x₂&lt;sup&gt;0&lt;/sup&gt;</td>
<td>y&lt;sup&gt;0&lt;/sup&gt;</td>
</tr>
<tr>
<td>x₁&lt;sup&gt;0&lt;/sup&gt;</td>
<td>x₂&lt;sup&gt;1&lt;/sup&gt;</td>
<td>y&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>x₁&lt;sup&gt;1&lt;/sup&gt;</td>
<td>x₂&lt;sup&gt;0&lt;/sup&gt;</td>
<td>y&lt;sup&gt;0&lt;/sup&gt;</td>
</tr>
<tr>
<td>x₁&lt;sup&gt;1&lt;/sup&gt;</td>
<td>x₂&lt;sup&gt;1&lt;/sup&gt;</td>
<td>y&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

Initial network

| X₁ | X₂ | P(Y|X₁,X₂) |
|----|----|-----------|
| x₁<sup>0</sup> | x₂<sup>0</sup> | 1 0       |
| x₁<sup>0</sup> | x₂<sup>1</sup> | 0.2 0.8   |
| x₁<sup>1</sup> | x₂<sup>0</sup> | 0.1 0.9   |
| x₁<sup>1</sup> | x₂<sup>1</sup> | 0.02 0.98 |

III. Known Structure, Incomplete Data

- **Goal:** Parameter estimation
- **Data** contains missing values (e.g. Naïve Bayes)

Input Data

<table>
<thead>
<tr>
<th>X₁</th>
<th>X₂</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>x₂&lt;sup&gt;0&lt;/sup&gt;</td>
<td>?</td>
</tr>
<tr>
<td>x₁&lt;sup&gt;0&lt;/sup&gt;</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>x₁&lt;sup&gt;1&lt;/sup&gt;</td>
<td>x₂&lt;sup&gt;0&lt;/sup&gt;</td>
<td>?</td>
</tr>
<tr>
<td>x₁&lt;sup&gt;1&lt;/sup&gt;</td>
<td>x₂&lt;sup&gt;1&lt;/sup&gt;</td>
<td>?</td>
</tr>
<tr>
<td>x₁&lt;sup&gt;1&lt;/sup&gt;</td>
<td>?</td>
<td>y&lt;sup&gt;0&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

Initial network

| X₁ | X₂ | P(Y|X₁X₂) |
|----|----|----------|
| x₁<sup>0</sup> | x₂<sup>0</sup> | 1 0       |
| x₁<sup>0</sup> | x₂<sup>1</sup> | 0.2 0.8   |
| x₁<sup>1</sup> | x₂<sup>0</sup> | 0.1 0.9   |
| x₁<sup>1</sup> | x₂<sup>1</sup> | 0.02 0.98 |
IV. Unknown Structure, Incomplete Data

- Goal: Structure learning & parameter estimation
- Data contains missing values

Parameter Estimation

- Input
  - Network structure
  - Choice of parametric family for each CPD $P(X_i|Pa(X_i))$

- Goal: Learn CPD parameters

- Two main approaches (CHLE)
  - Maximum likelihood estimation
  - Bayesian approaches
Biased Coin Toss Example

- Coin can land in two positions: Head or Tail

- Estimation task
  - Given toss examples $x[1],...,x[m]$ estimate $P(X=h)=\theta$ and $P(X=t)=1-\theta$
  - Denote by $P(H)$ and $P(T)$ to mean $P(X=h)$ and $P(X=t)$, respectively.

- Assumption: i.i.d samples
  - Tosses are controlled by an (unknown) parameter $\theta$
  - Tosses are sampled from the same distribution
  - Tosses are independent of each other

Goal: find $\theta \in [0,1]$ that predicts the data well

“Predicts the data well” = (likelihood) of the data given $\theta$

$\mathcal{L}(\theta; D) = P(D | \theta) = \prod_{i=1}^{m} P(x[i] | x[i-1], \theta) = \prod_{i=1}^{m} P(x[i] | \theta)$

Example: probability of sequence $H,T,T,H,H$

$\mathcal{L}(\{H,T,T,H,H\}; \theta) = P(H | \theta)P(T | \theta)P(T | \theta)P(H | \theta)P(H | \theta) = \theta^3(1-\theta)^2$
Maximum Likelihood Estimator

- Parameter $\theta$ that maximizes $L(D; \theta) = p(D|\theta)$
- In our example, $\theta = 0.6$ maximizes the sequence $H,T,T,H,H$

In our example, $\theta = 0.6$ maximizes the sequence $H,T,T,H,H$

$\theta_{MLE} = 0.6$

![Graph showing the likelihood function $L(D; \theta)$ vs $\theta$]

Maximum Likelihood Estimator

- General case
  - Observations: $M_H$ heads and $M_T$ tails
  - Find $\theta$ maximizing likelihood
    
    $L(M_H, M_T; \theta) = \theta^{M_H} (1 - \theta)^{M_T}$

  - Equivalent to maximizing log-likelihood
    
    $l(M_H, M_T; \theta) = M_H \log \theta + M_T \log(1 - \theta)$

  - Differentiating the log-likelihood and solving for $\theta$, we get that the maximum likelihood parameter is:

    $\theta_{MLE} = \arg \max_{\theta} L(M_H, M_T; \theta)$

    $\theta_{MLE} = \frac{M_H}{M_H + M_T}$

    $\frac{\partial l}{\partial \theta} |_{\theta = \theta_{MLE}} = 0$
Acknowledgement

- These lecture notes were generated based on the slides from Prof Eran Segal.