Inference and Message Passing
Advanced Machine Learning (Spring 2024)

Introduction 1

Probabilistic graphical models (PGMs) provide a powerful framework for representing complex joint distributions over many random variables. By exploiting the inherent structure of these distributions, PGMs enable efficient computation for both inference and learning. In many practical applications—from computer vision to natural language processing—exact inference is often computationally prohibitive, and one must resort to approximate methods.
The two primary inference tasks in PGMs are:

- **Marginal Inference:** Calculating the marginal probability $P(X_i \mid E = e)$ for a variable X_i given some evidence $E = e$.
- **Maximum a Posteriori (MAP) Inference:** Finding the most likely configuration $x^* = \arg\max_x P(X = x \mid E = e)$.

Preliminaries and Definitions 2

2.a Graphical Models

A **graphical model** is a probabilistic model for which a graph expresses the conditional dependence structure between random variables. There are two main types:

- **Bayesian Networks:** Directed acyclic graphs where edges represent causal influences.
- **Markov Random Fields (MRFs):** Undirected graphs where edges capture pairwise relationships between variables.

2.b Factors and Potentials

In PGMs, **factors** (or **potentials**) are functions that assign a non-negative value to each configuration of a subset of variables. The joint probability distribution over the variables $\mathbf{x}$ is typically written as:

$$P(\mathbf{x}) = \frac{1}{Z} \prod_{\phi \in \Phi} \phi(\mathbf{x}_\phi)$$

where Z is the normalization constant (partition function).

2.c Inference Tasks

Definition 2.1 (Marginal Inference). *Given a set of variables X and evidence $E = e$, marginal inference computes:*

$$P(X \mid E = e) = \sum_{\mathbf{x} \setminus X} P(\mathbf{x} \mid E = e)$$

Definition 2.2 (MAP Inference). *MAP inference finds the assignment x^* that maximizes the posterior probability:*

$$x^* = \arg\max_x P(x \mid E = e)$$

Factorization in Inference Queries 3

Inference in probabilistic graphical models often requires computing marginal probabilities or finding the most probable assignment of variables. The challenge is the exponential complexity involved in summing or maximizing over all possible values.

3.a Joint Probability Distribution

Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of random variables. The joint probability distribution is given by:

$$P(x_1, x_2, \dots, x_n)$$

Computing this directly is infeasible due to the exponential number of combinations of values.

3.b Factorization Using Potential Functions

Instead, we can express the joint distribution as a product of factors:

$$P(x_1, x_2, \dots, x_n) \propto \prod_{c \in \mathcal{F}} \psi_c(X_c)$$

where:

- $\propto$ denotes "proportional to" (ignoring normalization constants).
- $\mathcal{F}$ is the set of factor dependencies.
- $\psi_c(X_c)$ is a potential function (factor) that depends only on a subset X_c of variables.

3.c Example: Factorizing a Three-Variable Distribution

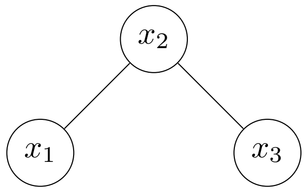
For three variables x_1, x_2, x_3, the joint probability can be factorized as:

$$P(x_1, x_2, x_3) \propto \psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)$$

where:

- $\psi_{12}(x_1, x_2)$ represents dependencies between x_1 and x_2.
- $\psi_{23}(x_2, x_3)$ represents dependencies between x_2 and x_3.

This factorization reduces computational complexity, making inference more efficient. Below is the graph for this Probability factorization.



3.d Example Of Efficient Inference

We compute the normalization constant Z using the given formula:

$$Z = \sum_{a,b} \psi_{AB}(a, b) \sum_{c} \psi_{BC}(b, c)$$

Assuming the states:

- A has values $\{a_1, a_2, a_3\}$
- B has values $\{b_1, b_2\}$
- C has values $\{c_1, c_2\}$

Let the potential functions be:

$$\psi_{AB} = \begin{bmatrix} 0.2 & 0.3 \\ 0.4 & 0.1 \\ 0.1 & 0.2 \end{bmatrix}$$

$$\psi_{BC} = \begin{bmatrix} 0.5 & 0.5 \\ 0.7 & 0.3 \end{bmatrix}$$

First, compute the inner sum:

$$\sum_c \psi_{BC}(b,c) = \begin{bmatrix} 0.5+0.5 \\ 0.7+0.3 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$$

Now, substitute into the outer sum:

$$Z = \sum_{a,b} \psi_{AB}(a,b) \times 1.0$$

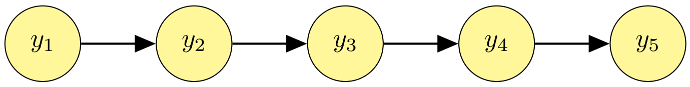
$$Z = \sum_{a,b} \psi_{AB}(a,b)$$

$$= 0.2 + 0.3 + 0.4 + 0.1 + 0.1 + 0.2 = 1.3$$

Thus, the normalization constant calculated according to this form is:

$$Z = 1.3$$

A similar correlation between the potentials can be used to calculate the cumulative probability for chain based mdoels.



- **Potentials:** $\psi_j(y_i, y_{i+1})$
- **Probability:**

$$Pr(y_1, \ldots, y_n) = \prod_i \psi_j(y_i, y_{i+1}) \cdot Pr(y_1)$$

This method however easy to compute and understand is not the efficient approach and a more efficient alternative exists
We aim to compute:

$$\Pr(y_5 = 1) = \sum_{y_1,\ldots,y_4} \Pr(y_1, \ldots, y_4, 1)$$

Expanding using the chain rule and factorizing:

$$\begin{aligned}
\Pr(y_5 = 1) &= \sum_{y_1}\sum_{y_2}\sum_{y_3}\sum_{y_4} \psi_1(y_1,y_2)\psi_2(y_2,y_3)\psi_3(y_3,y_4)\psi_4(y_4,1)P(y_1) \\
&= \sum_{y_1} P(y_1) \sum_{y_2} \psi_1(y_1,y_2) \sum_{y_3} \psi_2(y_2,y_3) \sum_{y_4} \psi_3(y_3,y_4)\psi_4(y_4,1) \\
&= \sum_{y_1} P(y_1) \sum_{y_2} \psi_1(y_1,y_2) B_2(y_2) \\
&= \sum_{y_1} B_1(y_1) P(y_1)
\end{aligned}$$

where:

$$B_i(y_i) = \sum_{y_{i+1}} \psi_i(y_i, y_{i+1}) B_{i+1}(y_{i+1})$$

with the base case:

$$B_4(y_4) = \sum_{y_5} \psi_4(y_4, y_5) = \psi_4(y_4, 1)$$

Hardness of Inference 4

Given a graphical model $P(x_1, \ldots, x_m)$ which is factorized efficiently in terms of potentials (e.g., polynomial number of potentials, with each potential containing a constant number of variables), can we always find $P(x_i)$ or Z in polynomial time? **No**
For example: The grid graph is an example of such a graph for which we cannot find in polynomial time. For proof refer Theorem 9.1 of KF textbook.
Further, we can map 3-SAT problem as a Bayesian network, and confirm that it is not always possible to find marginals in polynomial time.

4.a Example 3-SAT Instance

Consider the formula:

$$\Phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_4)$$

$$C_1 = x_1 \vee \neg x_2 \vee x_3 \quad C_2 = \neg x_1 \vee x_2 \vee \neg x_3 \quad C_3 = x_2 \vee x_3 \vee \neg x_4$$

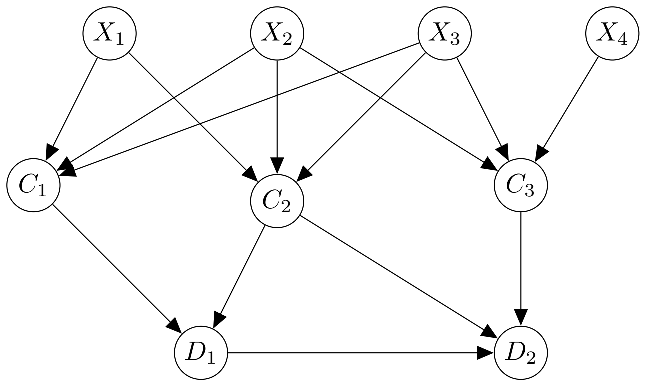
4.a.1 Mapping 3-SAT to a Bayesian Network

We construct a Bayesian network where:

- Each Boolean variable x_i is represented as a node.
- Each clause introduces a **constraint node** ensuring at least one literal is true.
- The probability of satisfaction corresponds to the probability that all constraint nodes are satisfied.
- We can introduce intermediate decision nodes and the probability of last decision node being satisfied is required.

4.a.2 Graphical Representation

The Bayesian network for our example formula is:



Each clause node C_i is a deterministic function of its parent variables. It is satisfied if at least one parent is true.

4.a.3 Query: Probability of Satisfaction

To determine whether the formula is satisfiable, we compute:

$$P(\text{D2} = 1) = P(C_1 = 1, C_2 = 1, C_3 = 1)$$

which involves summing over all possible truth assignments.
We thus notice that in general,the reduction from 3-SAT to Bayesian inference is not possible in polynomial time and can take exponential order time.

Exact Inference: Variable Elimination 5

Variable Elimination (VE) is a systematic algorithm that exploits the distributive law to efficiently compute marginal distributions. It works by summing out (or "eliminating") variables one at a time from the joint distribution.

5.a Algorithm Description

Given a set of factors Φ, the joint distribution is written as:

$$P(X) = \sum_{Y_1} \cdots \sum_{Y_n} \prod_{\phi \in \Phi} \phi$$

For example, in a chain-structured MRF $A - B - C - D$, one possible elimination order is to eliminate B first, then C, and finally A:

$$P(D) = \sum_{C} \phi(C, D) \left(\sum_{B} \phi(B, C) \left(\sum_{A} \phi(A, B) \right) \right)$$

5.b Pseudocode for Variable Elimination

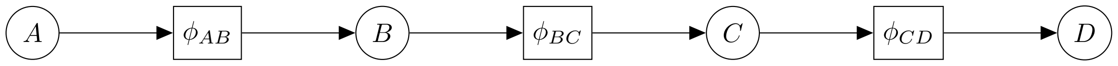
The following pseudocode outlines the Variable Elimination algorithm:

Algorithm 1 Variable Elimination

```
1: procedure VariableElimination($\Phi$, Query $Q$, Elimination Order $\{Y_1, Y_2, \dots, Y_n\}$, Evidence $E$)
2:     Input: Set of factors $\Phi$, query variable(s) $Q$, elimination order, evidence $E$
3:     Output: Marginal distribution $P(Q \mid E)$
4:     for all factor $\phi \in \Phi$ do
5:         Restrict $\phi$ with evidence $E$
6:     end for
7:     for each variable $Y$ in elimination order do
8:         $F_Y \leftarrow \{$factors in $\Phi$ that mention $Y\}$
9:         $\psi \leftarrow$ Product of factors in $F_Y$
10:        $\psi' \leftarrow \sum_Y \psi$
11:        Remove factors in $F_Y$ from $\Phi$
12:        Add factor $\psi'$ to $\Phi$
13:    end for
14:    $\psi_{\text{final}} \leftarrow$ Product of all remaining factors in $\Phi$
15:    return Normalize($\psi_{\text{final}}$)
16: end procedure
```

5.c Illustrative Example

Consider a chain-structured model represented in the figure below:



Here, eliminating B and C produces intermediate factors that integrate the contributions from their neighboring variables.

Approximate Inference Methods 6

When exact inference is infeasible due to computational constraints (for example, in loopy graphs), approximate inference techniques are used.

6.a Sampling Methods

Sampling methods, such as Markov Chain Monte Carlo (MCMC), approximate the target distribution by drawing samples. One common sampling technique is **Gibbs Sampling**.

6.a.1 Gibbs Sampling Pseudocode

Gibbs Sampling updates one variable at a time by sampling from its conditional distribution:

Algorithm 2 Gibbs Sampling

```
1: procedure GibbsSampling($P$, Initial State $x^{(0)}$, Number of Iterations $T$)
2:     $x \leftarrow x^{(0)}$
3:     for $t = 1$ to $T$ do
4:         for each variable $x_i$ in $x$ do
5:             Sample $x_i \sim P(x_i \mid x_{\setminus i})$
6:         end for
7:         Record sample $x^{(t)}$
8:     end for
9:     return $\{x^{(t)}\}_{t=1}^{T}$
10: end procedure
```

6.b Variational Inference

Variational Inference approximates the true distribution $P(X)$ with a simpler distribution $Q(X)$ by turning inference into an optimization problem. The goal is to minimize the Kullback-Leibler divergence $D_{KL}(Q\|P)$. This method is particularly useful in large-scale applications.

6.c Loopy Belief Propagation

In graphs containing cycles, standard Belief Propagation does not guarantee convergence. *Loopy Belief Propagation* uses the same message passing rules iteratively to provide an approximate solution. Despite the lack of guarantees, it often performs well in practice.

Variable Elimination on General Graphs: Generalizing Chain Inference 7

- **Given:** Potentials $\psi_C(x_C)$, where C are cliques in a graph G.
- **Find:**

$$Z = \sum_{x_1,\ldots,x_n} \prod_C \psi_C(x_C)$$

We can follow a simple procedure to achieve this. To perform variable elimination, the process begins with choosing a good ordering of variables, $x_1, x_2, \ldots, x_n$, as the order significantly affects the efficiency of computation. Next, a list of factors $\mathcal{F}$ is initialized, where each factor $\psi_C(x_C)$ corresponds to a clique (a group of connected nodes) in the graph. Variables are then eliminated one by one in the chosen order.
For each variable x_i, the relevant factors in $\mathcal{F}$ that involve x_i are identified and grouped as $\mathcal{F}_i$. These factors are multiplied together to compute M_i. The variable x_i is then removed by summing M_i over all possible values of x_i, resulting in a new factor m_i. The factor list $\mathcal{F}$ is updated by removing the old factors $\mathcal{F}_i$ and replacing them with m_i. This process continues until all variables have been processed. The final factor m_n contains the result Z, which represents the desired computation.

7.a Example:

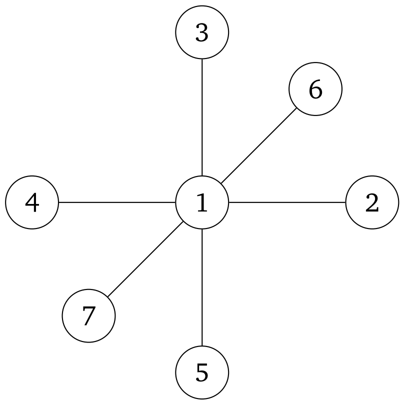
- **Given:** $\psi_{12}(x_1,x_2), \psi_{24}(x_2,x_4), \psi_{23}(x_2,x_3), \psi_{45}(x_4,x_5), \psi_{35}(x_3,x_5)$
- **Find:**

$$Z = \sum_{x_1,\ldots,x_5} \psi_{12}(x_1,x_2)\psi_{24}(x_2,x_4)\psi_{23}(x_2,x_3)\psi_{45}(x_4,x_5)\psi_{35}(x_3,x_5)$$

- $x_1 : \prod\{\psi_{12}(x_1,x_2)\} \to M_1(x_1,x_2) \xrightarrow{\sum_{x_1}} m_1(x_2)$
- $x_2 : \prod\{\psi_{24}(x_2,x_4), \psi_{23}(x_2,x_3), m_1(x_2)\} \to M_2(x_2,x_3,x_4) \xrightarrow{\sum_{x_2}} m_2(x_3,x_4)$
- $x_3 : \prod\{\psi_{35}(x_3,x_5), m_2(x_3,x_4)\} \to M_3(x_3,x_4,x_5) \xrightarrow{\sum_{x_3}} m_3(x_4,x_5)$
- $x_4 : \prod\{\psi_{45}(x_4,x_5), m_3(x_4,x_5)\} \to M_4(x_4,x_5) \xrightarrow{\sum_{x_4}} m_4(x_5)$
- $x_5 : \prod\{m_4(x_5)\} \to M_5(x_5) \xrightarrow{\sum_{x_5}} Z$

Choosing a Variable Elimination Order 8

Variable elimination (VE) has a complexity of $O(nm^w)$, where w is the maximum number of variables in any factor. Choosing a poor elimination order can create large intermediate factors, drastically increasing complexity. For example, eliminating x_2 first could result in a complexity of $O(nm^4)$. This highlights the importance of selecting an efficient order.



Finding the optimal elimination order is NP-hard for general graphs, but polynomial-time algorithms exist for chordal graphs. A graph is chordal if every cycle longer than three has a shortcut (an extra edge connecting non-adjacent nodes). Triangulation, which minimizes the size of the largest clique, reduces computation costs and is key to making VE more efficient. While optimal triangulation is also NP-hard, heuristics can provide good approximations.

Finding Optimal Order in a Triangulated Graph 9

Goal: Find the optimal ordering for $P(x_1)$ inference, where x_1 has to be the last in the ordering.
Input: Graph G, where $n =$ number of vertices in G.

```
1. For $i = 2, \dots, n$, do:
    (a) $\pi_{i-1} =$ pick any simplicial vertex in $G$ other than $1$.
    (b) Remove $\pi_{i-1}$ from $G$.
2. Return ordering $\pi_1, \pi_2, \dots, \pi_{n-1}$.
```

In the above algorithm, if no simplicial vertex is found at a particular step, it indicates that the graph is not triangulated. To address this, additional edges can be added to make the graph triangulated at that point. The process of adding these edges can follow specific heuristics. One heuristic is to choose the vertex with the smallest degree and connect all its neighbors. Another heuristic is to choose the vertex that will require the fewest additional edges to connect its neighbors. These strategies help ensure the graph becomes triangulated efficiently while minimizing the number of added edges.

Junction Trees 10

A junction tree (or clique tree) is a tree structure used for exact inference in graphical models by clustering variables into cliques and performing message passing efficiently.

10.a Why Junction Trees?

Junction trees provide a structured approach to inference by converting a graph into a tree structure that satisfies the running intersection property. This ensures that each variable appears in a connected subtree, allowing for efficient message passing without loops.

10.b Junction tree construction using VE algorithm

1. **Find Optimal Ordering:** Determine the optimal ordering of variable elimination using the simplicial vertices ordering. If the graph is not triangulated, perform triangulation during this process.
2. **Initialize Clique Graph:** Begin with an empty clique graph C_G.
3. **Simulate VE Algorithm:** Simulate the VE process. Each intermediate factor M_i becomes a node in C_G, connected to all factors in $\mathcal{F}_i$ that were multiplied to form M_i.
4. **Remove Subsumed Nodes:** Eliminate nodes in C_G that are subsumed by other nodes to reduce redundancy.
5. **Assign Potentials:** Assign each potential to any node in C_G that subsumes its variables.

10.c Example

We have 6 variables: $X_1, X_2, X_3, X_4, X_5, X_6$ with the following dependencies:

$$X_1 \to X_2, \quad X_2 \to X_3, \quad X_2 \to X_4, \quad X_3 \to X_5, \quad X_4 \to X_6, \quad X_5 \to X_6$$

This structure is represented by the following graph:

$$\begin{tikzcd}
X_1 \arrow[r] & X_2 \arrow[r] \arrow[d] & X_3 \arrow[d] & \\
& X_4 \arrow[drr] & X_5 \arrow[dr] & \\
& & & X_6
\end{tikzcd}$$

10.c.1 Step 1: Find Optimal Ordering

We begin by determining the optimal ordering of variables for variable elimination using **simplicial vertex ordering**. This step ensures efficient triangulation.
Optimal ordering: $X_2, X_1, X_3, X_4, X_5, X_6$.

10.c.2 Step 2: Initialize Clique Graph

Start with an empty **clique graph (CG)**. Initially, no cliques are formed.

$$\text{Clique Graph (CG)} = \emptyset$$

10.c.3 Step 3: Simulate Variable Elimination Process

Perform variable elimination in the order $X_2, X_1, X_3, X_4, X_5, X_6$.
1. Eliminate X_2: Combine factors involving X_2 and create an intermediate factor M_1.
Add M_1 to the clique graph and connect it to the neighboring factors.
2. Eliminate X_1: Combine factors involving X_1 and create another factor M_2, connected to the appropriate factors.
3. Repeat the process for X_3, X_4, X_5, X_6, creating intermediate factors M_3, M_4, M_5, M_6.

10.c.4 Step 4: Remove Subsumed Nodes

After performing variable elimination, **remove redundant nodes** that are subsumed by other nodes in the clique graph to reduce redundancy and simplify the structure.
1. Check for nodes that are included in other factors.
2. Remove these redundant nodes.

10.c.5 Step 5: Assign Potentials

After the elimination, assign potentials (probability distributions) to each clique in the junction tree.
- Each factor is assigned to the clique that subsumes its variables.
- Potentials represent the marginal distributions over the variables in each clique.

10.d Final Junction Tree

The final junction tree will consist of cliques connected by edges. Each clique represents a factor, and belief propagation can be used to perform inference efficiently.

$$\text{Junction Tree (CG)} = \{M_1, M_2, M_3, M_4, M_5, M_6\}$$

Junction trees provide an efficient structure for inference by converting a probabilistic graphical model into a tree. The transformation reduces computational complexity by ensuring that messages propagate efficiently without cycles.

Computation Graph of the VE Algorithm 11

A special graph over cliques — Junction Trees.

1. Find the optimal ordering of variable elimination using the simplicial vertices ordering, triangulating the graph in the process if necessary.
2. Initialize the clique graph as empty.
3. Simulate the running of the VE algorithm. Each M_i is a node in the graph. M_i is connected to all factors in M_i that were multiplied to form M_i.
4. Remove nodes that are subsumed by other nodes.
5. Assign each potential to any node in the graph that subsumes its variables.

11.a The Viterbi Algorithm

Let observations x_t take one of k possible values, and states y_t take one of m possible values.
Given n observations: $o_1, \ldots, o_n$
Given Potentials:

- $Pr(y_t|y_{t-1}) = P(y|y')$ (Table with m^2 values)
- $Pr(x_t|y_t) = P(x|y)$ (Table with mk values)
- $Pr(y_1) = P(y)$ (Start probabilities, Table with m values)

Find $\max_y Pr(Y|X = O)$
Initialization:

$$B_1[y] = P(y)P(o_1|y), \quad y \in \{1, \ldots, m\}$$

Recursion: For $t = n \ldots 2$:

$$\psi(y, y') = P(y|y')P(x_t = o_t|y)$$
$$B_{t-1}[y] = \max_{y'=1}^{m} \psi(y, y')B_t[y']$$

Termination: Return $\max_y B_1[y]P(y)P(x_t = o_t|y)$
Time taken: $O(nm^2)$

Message Passing in Probabilistic Graphical Models 12

Message passing is a fundamental algorithmic paradigm used for inference in probabilistic graphical models. It allows the computation of exact or approximate marginal probabilities efficiently.The message passing algorithm executes the variable elimination process on a junction tree, enabling the computation of all marginals or inference queries in a single run.

12.a Pseudocode for Belief Propagation

Below is the pseudocode for Belief Propagation on a tree-structured factor graph:

Algorithm 3 Belief Propagation for Trees

```
procedure BELIEFPROPAGATION($G(V, F)$)
    Input: Factor graph $G$ with variable nodes $V$ and factor nodes $F$
    Initialize messages $m_{i\to a}(x_i)$ and $m_{a\to i}(x_i)$ for all edges.
    Upward Pass:
    for each leaf variable node $i$ do
        Send message $m_{i\to a}(x_i)$ to its neighboring factor $a$
    end for
    for each non-leaf variable node in post-order traversal do
        for each neighboring factor $a$ do
            Compute $m_{i\to a}(x_i) = \prod_{b\in N(i)\setminus a} m_{b\to i}(x_i)$
            Send $m_{i\to a}(x_i)$ to factor $a$
        end for
    end for
    Downward Pass:
    for each factor node in pre-order traversal do
        for each neighboring variable node $i$ do
            Compute $m_{a\to i}(x_i) = \sum_{\mathbf{x}_a\setminus x_i} f_a(\mathbf{x}_a) \prod_{j\in N(a)\setminus i} m_{j\to a}(x_j)$
            Send $m_{a\to i}(x_i)$ to variable $i$
        end for
    end for
    Compute Beliefs:
    for each variable node $i$ do
        Belief($x_i$) $\propto \prod_{a\in N(i)} m_{a\to i}(x_i)$
    end for
    return {Belief($x_i$)}
end procedure
```

12.b Message Passing Algorithm

Given a clique graph (CG) where cliques are nodes connected to other cliques, the message passing algorithm follows these steps:

- Until convergence, repeat the following:
  - Each node c sends a message $m_{c\to c'}(.)$ to each of its neighbors c' once it has received messages from all other neighbors $N(c) - \{c'\}$.

The message update equation is given by:

$$m_{c\to c'}(x_s) = \sum_{x_c - x_s} \psi_c(x_c) \prod_{d\in N(c)-\{c'\}} m_{d\to c}(x_{d\cap c})$$

For MAP queries, the summation is replaced with maximization.
The algorithm operates as follows: Each clique node C in the junction tree continuously sends messages to its neighboring nodes. The process runs iteratively over time steps, from $t = 1$ to $t = t$.
At every time step, each clique node in the junction tree attempts to transmit a message to its neighbors. However, a clique can only send a message to a neighbor once it has received messages from all its other neighbors.
We denote the message sent from clique C to its neighboring clique C' as:

$$M(C \to C') = f(C, C')$$

A clique node C can only send a message after it has received messages from all its other neighboring cliques, which are represented as $N(C)$. Once all messages are received from $N(C) \setminus C'$, it can forward a message to C'. Mathematically, this condition can be expressed as:

$$M(C \to C') = \psi(C, C') \prod_{C''\in N(C)\setminus C'} M(C'' \to C)$$

where $\psi(C, C')$ represents the factor potential between cliques C and C', and $M(C'' \to C)$ represents the messages received from all other neighbors.

12.c Computing Marginal Probabilities

The marginal probability over all variables in a clique is computed as:

$$P(x_c) \propto \psi_c(x_c) \prod_{d \in N(c)} m_{d \to c}(x_{d \cap c})$$

From this, the probability of any variable x_i in the clique is determined by:

$$P(x_i) = \sum_{x_c - x_i} P(x_c)$$

12.d Example

Suppose we have the original graph as follows:

$$\begin{tikzcd}
X_7 \arrow[dr, no head] & & X_1 \arrow[dl, no head] \arrow[dr, no head] \arrow[drr, no head] & & & X_5 \arrow[dl, no head] \arrow[dr, no head] & \\
& X_2 \arrow[rr, no head] & & X_3 \arrow[r, no head] & X_4 & & X_6
\end{tikzcd}$$

The junction tree corresponding to this graph:

$$\begin{tikzcd}
(X_2, X_7) \arrow[r, leftrightarrow, bend left, "t=1", "t=4"'] & (X_1, X_2, X_3) \arrow[r, leftrightarrow, bend left, "t=2", "t=3"'] & (X_3, X_1, X_4) \arrow[r, leftrightarrow, bend left, "t=3", "t=2"'] & (X_4, X_1, X_5) \arrow[r, leftrightarrow, bend left, "t=4", "t=1"'] & (X_5, X_6) \\
\text{(A)} & \text{(B)} & \text{(C)} & \text{(D)} & \text{(E)}
\end{tikzcd}$$

The tree satisfies the running intersection property and is indeed correct.

$$m_{C \to C'}(C \cap C' = X_3, X_4) \to \text{message}$$

At time $t = 1$: Only cliques A and E can send messages:

$$m_{A \to B}(X_2) = \sum_{X_7} \psi_{27}(X_2, X_7)$$

$$m_{E \to D}(X_5) = \sum_{X_6} \psi_{56}(X_5, X_6)$$

At time $t = 2$:

$$m_{B \to C}(X_1, X_3) = \sum_{X_2} \psi_{123}(X_1, X_2, X_3) \cdot m_{A \to B}(X_2)$$

$$m_{D \to C}(X_4) = \sum_{X_5} \psi_{45}(X_4, X_5) \cdot m_{E \to D}(X_5)$$

At time $t = 3$:

$$m_{C \to D}(X_4) = \sum_{X_3, X_5} \psi_{134}(X_1, X_3, X_4) \cdot m_{B \to C}(X_1, X_3)$$

$$m_{C \to B}(X_1, X_3) = \sum_{X_4} \psi_{134}(X_1, X_3, X_4) \cdot m_{D \to C}(X_4)$$

At time $t = 4$: All nodes would have received messages.
Now we got all factors which are required to answer all marginal queries.
Suppose we have to find the marginal probability of x_i as: Let c be a clique in the function tree containing x_i, then:

$$P(x_i) \propto \sum_{X_c - x_i} \psi_c(X_c) \prod_{d \in N(c)} M_{d \to c}(X_{d \cap c})$$

Note: This algorithm is not efficient when we have to answer queries over a variable (For that, variable elimination is efficient).
This algorithm is optimal when we have to query probability over a subset of variables which appear in clique.

Baum-Welch algorithm 13

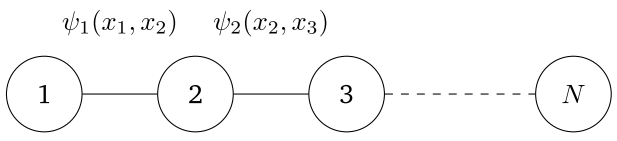
The Baum-Welch algorithm is a special case of sum-product message passing over chain graphs. It is widely used for inference in Hidden Markov Models (HMMs). In this section, we discuss how the message-passing scheme works when dealing with chain graphs.

13.a Problem Setup

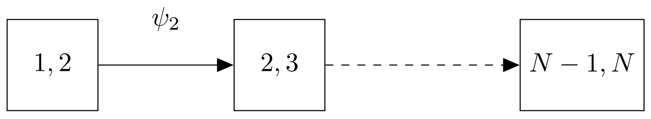
Given a set of potentials $\psi_i(x_i, x_{i+1})$ for $i = 1, \dots, N-1$, we aim to compute the marginal probabilities efficiently using a structured approach.

13.b Graph Representation

- The given graph G consists of nodes arranged in a chain-like structure:



- The corresponding clique graph C_G has nodes representing the edges of G. Each clique contains two connected nodes from G:



13.c Message Passing in Chain Graphs

13.c.1 Forward Pass

To compute marginals, we perform a forward pass where messages propagate from left to right along the chain:

$$m_{i\to i+1}(x_{i+1}) = \sum_{x_i} \psi_i(x_i, x_{i+1}) m_{i-1\to i}(x_i).$$

Example: Consider a simple case where $N = 3$, and the given potentials are:

$$\psi_1(x_1, x_2) = \begin{bmatrix} 0.8 & 0.2 \\ 0.4 & 0.6 \end{bmatrix}, \quad \psi_2(x_2, x_3) = \begin{bmatrix} 0.7 & 0.3 \\ 0.5 & 0.5 \end{bmatrix}.$$

Starting from an initial message $m_{0\to 1}(x_1) = [0.5, 0.5]$, we compute:

$$m_{1\to 2}(x_2) = \sum_{x_1} \psi_1(x_1, x_2) m_{0\to 1}(x_1).$$

This results in new messages being passed forward iteratively.

13.c.2 Backward Pass

In a similar manner, a backward pass can be performed to compute marginals efficiently.

13.d Complexity and Applications

Message passing has a worst-case complexity of $O(m^w N)$, where w is the size of the largest clique, and m is the number of values each discrete variable can take. It is particularly efficient for tree-structured graphs, where inference runs in linear time.
Popular applications include:

- Viterbi Algorithm for Hidden Markov Models (HMMs)
- Forward-Backward Algorithm for Kalman Filters