CS 726 Coding Assignment 1
A Message Passing Approach

Hrishikesh Deore(22b1832), R Rushikesh(22b0948), Subhashini(22b0013)

February 20, 2025

Contents

1 Acknowledgment of Tools and Sources

This project was developed collaboratively by the team using Python and Jupyter Notebook for implementation and testing. To enhance efficiency and clarity, we utilized the following tools and resources:

- **ChatGPT:** Used for generating structured explanations, pseudocode, and refining the LaTeX report format.
- **Python Libraries:** Standard Python libraries such as `itertools`, `collections`, and `heapq` were employed for graph operations, clique detection, and top-k assignment computation.
- **Textbook References:** Concepts and algorithms were referenced from:
  - Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

All code was written by the team unless explicitly mentioned. Code snippets inspired by external sources are properly cited within the comments.

—

2 Introduction

Probabilistic inference is a fundamental task in graphical models, enabling the computation of marginal probabilities and the most probable assignments given observed evidence. This report presents a structured solution using message-passing algorithms in undirected graphical models. The key tasks include:

- Triangulating the graph to obtain a chordal structure.
- Constructing a junction tree from the triangulated graph.
- Computing marginal probabilities using the sum-product algorithm.
- Determining the Maximum A Posteriori (MAP) assignment using the max-product algorithm.
- Finding the top k most probable assignments.

—

3 Graph Triangulation

3.1 Definition

Graph triangulation involves converting an undirected graph into a **chordal graph**, where every cycle of length greater than three has a chord (an edge connecting non-consecutive nodes). This transformation is crucial for efficient inference.

3.2 Method Used

We employed the **minimum degree heuristic** for triangulation:

1. Select a node with the minimum degree.
2. Connect its neighbors to form a clique.
3. Remove the node and repeat until the graph is fully triangulated.

3.3 Pseudocode

Algorithm 1 Graph Triangulation using Minimum Degree Heuristic

Require: Undirected graph $G = (V, E)$
Ensure: Triangulated graph G'
Initialize an empty list for elimination order
while Graph G is not empty **do**
    Select node v with the minimum degree
    Connect all non-adjacent neighbors of v
    Remove v from the graph
    Append v to the elimination order
end while
return Triangulated graph G'

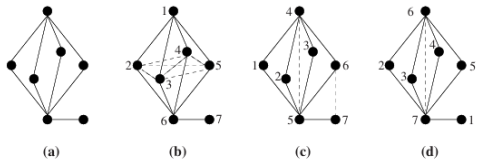

Fig. 1. (a) A graph G, and various triangulations of G by EG through the given orderings: (b) A minimal triangulation with $O(n^2)$ fill edges. (c) A non-minimal triangulation of G with less fill. (d) A minimum triangulation of G.

Figure 1

3.4 Example Illustration

—

4 Junction Tree Construction

4.1 Definition

A **Junction Tree** is a tree structure where each node represents a maximal clique of the triangulated graph, and edges represent shared variables between cliques, maintaining the **running intersection property**:

$$S_i = C_i \cap C_j$$

where C_i and C_j are adjacent cliques and S_i is their separator.

4.2 Methodology

1. Identify **maximal cliques** using the Bron-Kerbosch algorithm.The Bron-Kerbosch algorithm works by recursively exploring potential cliques through three sets

   - R: The current clique being constructed.
   - P: The set of potential candidates that can extend R.
   - X: The set of excluded vertices that would form duplicate cliques if added.

2. Construct a **weighted graph**, where nodes represent cliques and edge weights reflect the size of their intersections.

3. Apply **Kruskal's algorithm** to obtain a **maximum spanning tree** (MST).Kruskal's algorithm operates on an edge-weighted graph and follows these steps:

   - Sort Edges: Arrange all edges in ascending order of weights. (For Maximum Spanning Tree, sort in descending order.)
   - Initialize Forest: Treat each node as a separate component (disjoint sets).
   - Iterate Through Edges: For each edge: If it connects two different components, add the edge to the MST. Merge the two components (using the Union-Find data structure). Stop: Once the MST contains n-1 edges (for n nodes), the process ends.

   Kruskal's algorithm was chosen because it efficiently constructs a maximum spanning tree of cliques, prioritizing stronger connections and ensuring the running intersection property. Its edge-based approach aligns perfectly with the task of forming a junction tree from weighted clique intersections.

4.3 Pseudocode

Algorithm 2 Junction Tree Construction

Require: Triangulated graph G'
Ensure: Junction Tree T
1: Identify maximal cliques $C_1, C_2, \ldots, C_n$
2: Construct edges weighted by clique intersections
3: Apply Kruskal's MST algorithm
4: Ensure the running intersection property
5: **return** Junction Tree T

4.4 Example

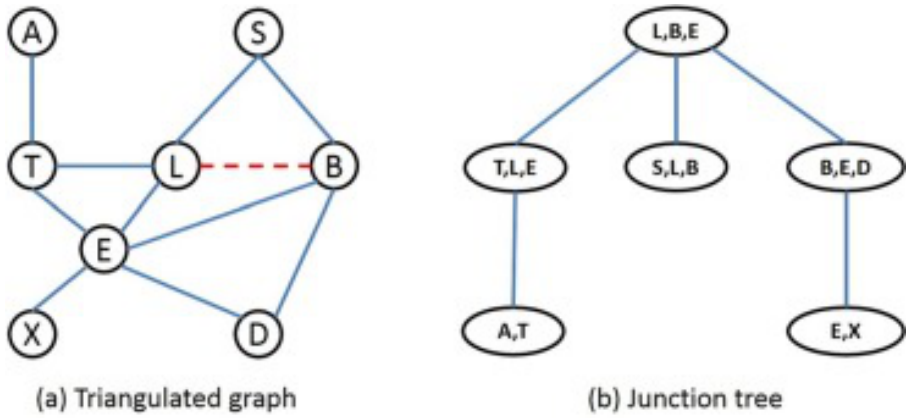

Figure 2: Junction Tree constructed from the triangulated graph.

—

5 Assigning Potentials to Cliques

5.1 Definition

After constructing the junction tree, the next step is to assign the input factor potentials to the appropriate cliques. This ensures that the factorization of the graphical model is preserved.

5.2 Methodology

1. **Identify Input Potentials:** Each potential is associated with a subset of variables, such as $\phi(X_i, X_j)$ for a pairwise factor.
2. **Assign to Cliques:** For each potential:
   - Identify the **smallest clique** in the junction tree that contains all variables in the potential's scope.
   - Assign the potential to that clique.
3. **Multiply Potentials:** Initialize each clique's potential to 1 and multiply by all assigned factors:

$$\psi_{C_i}(X_{C_i}) = \prod_{\phi \in \text{Factors}(C_i)} \phi(X_{C_i})$$

4. **Normalization(optional):** To avoid numerical instability, potentials can be normalized:

$$\psi_C(X_C) \leftarrow \frac{\psi_C(X_C)}{\sum_{X_C} \psi_C(X_C)}$$

5.3 Pseudo code

Algorithm 3 Assign Potentials to Cliques

Require: Triangulated cliques, input factors with scopes and potential values
Ensure: Clique potentials assigned and updated
Initialize clique potentials:
for each clique C in triangulated cliques **do**
  Sort variables in C
  Initialize $\psi_C(X_C) \leftarrow 1$ for all assignments
end for
Assign factors to cliques:
for each input factor $\phi(X_F)$ with scope F and potential values **do**
  Identify cliques containing scope F
  if such cliques exist **then**
    Choose the smallest clique C^* containing F
    Sort C^* and F variables lexicographically
    Multiply $\phi(X_F)$ into $\psi_{C^*}(X_{C^*})$
  end if
end for
Optional: Normalize clique potentials:
for each clique C **do**
  Normalize $\psi_C(X_C) \leftarrow \frac{\psi_C(X_C)}{\sum_{X_C} \psi_C(X_C)}$
end for
Output: Assigned clique potentials ψ_C for all cliques

This assignment ensures that the junction tree retains the original factorization of the graphical model, facilitating efficient message passing for marginalization and MAP inference.

—

6 Marginal Probability Computation

6.1 Definition

The **marginal probability** of a variable X_i is the sum of probabilities of all possible configurations involving X_i:

$$P(X_i) = \sum_{\mathbf{x} \setminus x_i} P(\mathbf{x})$$

6.2 Method Used

We use the **sum-product algorithm** to propagate messages along the junction tree:

1. Compute Partition Function (Z):

   The partition function Z is obtained using the sum-product algorithm on the junction tree, ensuring proper normalization of probability distributions.

2. Obtain Clique Beliefs:

   Messages are passed between cliques in the junction tree to compute beliefs for each clique. These beliefs represent the marginal probability distributions over the variables in that clique.

3. Extract Marginals for Individual Variables:

   The marginal probability of each variable is obtained by summing out all other variables from the relevant clique belief:

$$P(X_i) = \sum_{X_{C_i} \setminus X_i} \psi_{C_i}(X_{C_i})$$

   Each variable's marginal is normalized using Z to ensure that the probabilities sum to 1.

6.3 Pseudocode

Algorithm 4 Sum-Product Algorithm for Marginal Computation

Require: Junction tree T, clique potentials ψ
Ensure: Marginal probabilities $P(X_i)$
1: Initialize messages m_{ij} for each edge
2: **for** each edge (i, j) in T **do**
3: $m_{ij} = \sum_{C_i \setminus S_{ij}} \psi_i \times \prod_{k \neq j} m_{ki}$
4: **end for**
5: Compute beliefs: $b_i = \psi_i \times \prod_j m_{ji}$
6: Normalize to obtain marginals
7: **return** Marginal probabilities

—

7 Maximum A Posteriori (MAP) Assignment

7.1 Definition

The **MAP assignment** finds the most probable configuration of variables:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} P(\mathbf{x})$$

7.2 Method Used

1. Max-Product Algorithm:

   Instead of summing out variables, we replace summation with maximization during message passing:

$$\psi_C(X_C) = \max_{X_C} \prod_{\phi \in \text{Factors}(C)} \phi(X_C)$$

   The junction tree propagates max-product messages to find the highest-probability configuration.

2. Backtracking for Assignment Recovery:

   After computing max-marginals, the optimal assignment is recovered using backtracking, selecting the maximizing value for each variable.

7.3 Pseudocode

Algorithm 5 Max-Product Algorithm for MAP Assignment

Require: Junction tree T, clique potentials ψ
Ensure: MAP assignment $\mathbf{x}^*$
1: **for** each edge (i, j) in T **do**
2: $\quad m_{ij} = \max_{C_i \setminus S_{ij}} \psi_i \times \prod_{k \neq j} m_{ki}$
3: **end for**
4: Backtrack to determine optimal assignment
5: **return** MAP assignment $\mathbf{x}^*$

—

8 Top-k Most Probable Assignments

8.1 Definition

The **top-k assignments** are the k most probable variable configurations, ranked by their joint probabilities:

$$P(\mathbf{x}_1) \geq P(\mathbf{x}_2) \geq \cdots \geq P(\mathbf{x}_k)$$

8.2 Method Used

1. Enumerate Possible Assignments:

   Generate all possible assignments of variables (for small models) or use a structured search approach for large models.

2. Compute Joint Probabilities:

   For each assignment, compute the probability by multiplying the assigned values through the factorized model.

3. Sort and Extract Top-k:

   Use a max-heap to efficiently retrieve the k highest probability assignments.

8.3 Pseudocode

Algorithm 6 Top-k Assignment Computation

Require: Clique potentials ψ, k
Ensure: Top-k assignments
1: **for** each assignment $\mathbf{x}$ **do**
2: $\quad$ Compute joint probability $P(\mathbf{x})$
3: **end for**
4: Sort assignments by probability
5: Select top k
6: **return** Top-k assignments

—

9 Results and Discussion

9.1 Example Output

The implemented algorithm produces the following results:

- **Marginal probabilities:** $P(X_i)$ for each variable.
- **MAP assignment:** $\mathbf{x}^*$, the most probable configuration.
- **Top-k assignments:** The k highest-probability assignments.

9.2 Significance

- **Marginal probabilities** aid in understanding individual variable distributions.
- **MAP assignment** identifies the most likely scenario.
- **Top-k assignments** provide alternative solutions for robust decision-making.

—

10 Contributions of Team Members

This project was completed collaboratively, with each member contributing to specific tasks:

- **Rushikesh 22b0948: Final code for functions(Triangulation, Junction Tree, Assigning potentials to clique, Marginal Probability Computation, MAP assignmnet, Top-k)**
- **Hrishikesh 22b1832: Helped with junction tree and Marginals and coded final report.** .
- **Subhashini 22b0013 : Helped with marginals and report**

—

11 Conclusion

This report demonstrates the application of message-passing algorithms for probabilistic inference in graphical models. The key takeaways are:

- Efficient graph triangulation ensures tractable inference.
- Junction tree construction maintains clique structure.
- Message passing enables computation of marginals, MAP, and top-k assignments.

—

References

1. Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.
2. Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

—