# CS 726 Report

Sambhav Jain, Aryan Katiyar, Adwai Krishna

November 19, 2025

## 1 Triangulation

The method employed in this implementation is the **minimum fill-in heuristic**, which efficiently selects the order of variable elimination to ensure that the resulting graph is chordal.

### 1.1 Minimum Fill-In Heuristic

The minimum fill-in heuristic operates by iteratively selecting a node for elimination based on the following criteria:

1. **Check for simplicial nodes**: A node is considered *simplicial* if all its neighbors form a fully connected clique. If such a node exists, it is eliminated first, as its removal does not introduce any additional edges.

2. **Select the node with minimal fill-in**: If no simplicial node is found, the algorithm selects a node that, upon elimination, introduces the fewest additional edges between its neighbors. This is determined by:

$$\text{FillIn}(v) = \frac{d(v)(d(v)-1)}{2} - \frac{1}{2} \sum_{\substack{u,w \in N(v) \\ u \neq w \\ w \in N(u)}} 1$$

   where $d(v)$ is the degree of node $v$, and $N(v)$ represents its neighbors.

3. **Complete the neighbors**: Before eliminating the chosen node, all its neighbors are fully connected, ensuring the graph remains chordal.

### 1.2 Advantages of Minimum Fill-In Heuristic

As this metric tries to greedily introduce the minimum number of edges required at any iteration, it keeps the graph structure close to the original, thereby ensuring that the elimination order minimizes the complexity of the subsequent inference steps in the junction tree algorithm.

# 2 Finding Maximal Cliques

A key step in constructing the junction tree is identifying the **maximal cliques** of the triangulated graph. A clique is a subset of vertices that are fully connected, and a clique is **maximal** if no additional vertex can be included without breaking its completeness.

## 2.1 Bron-Kerbosch Algorithm

To extract maximal cliques efficiently, we use the **Bron-Kerbosch algorithm**, which finds all maximal cliques in an undirected graph. It operates recursively while maintaining three sets:

- $R$ — The current clique being expanded.
- $P$ — The set of potential nodes that can still be added to $R$.
- $X$ — The set of nodes that must be excluded to avoid duplicate cliques.

At each step, the algorithm selects a pivot node to reduce the number of recursive calls, improving efficiency. The process continues until both $P$ and $X$ are empty, at which point $R$ forms a maximal clique.

## 2.2 Algorithm Pseudocode

The Bron-Kerbosch algorithm with pivoting is outlined below:

**Algorithm 1** Bron-Kerbosch with Pivoting

```
1: procedure BronKerbosch(R, P, X, G)
2:     if P = ∅ and X = ∅ then
3:         Output R as a maximal clique return
4:     end if
5:     Choose a pivot vertex u ∈ P ∪ X
6:     for each vertex v ∈ P \ N(u) do
7:         BronKerbosch(R ∪ {v}, P ∩ N(v), X ∩ N(v), G)
8:         P ← P \ {v}
9:         X ← X ∪ {v}
10:    end for
11: end procedure
```

Here, $N(v)$ represents the neighbors of $v$ in the graph $G$. Using a pivot $u$ reduces the number of recursive calls, making the algorithm more efficient.

# 3 Junction Tree Construction

Now to create a tree using the maximal cliques obtained above as nodes, such that it also satisfies running intersection property, we have used the Maximum Spanning tree algorithm (in terms of the separator set). We have used **Kruskal's algorithm** for this purpose.

## 3.1 Algorithm for Junction Tree Construction

Given a set of maximal cliques obtained from a triangulated graph, the junction tree is constructed as follows:

1. Compute the **separator set** for each pair of cliques:

$$S(i,j) = C_i \cap C_j, \quad \forall i,j \text{ where } i \neq j \tag{1}$$

   where $C_i$ and $C_j$ are maximal cliques, and $S(i,j)$ is the set of shared variables.

2. Compute the **size** of each separator:

$$\text{size}(S(i,j)) = |S(i,j)| \tag{2}$$

3. Sort separator sets in descending order of size to prioritize stronger connections.

4. Use the **Union-Find** data structure to construct a maximum-weight spanning tree(**Kruskal's Algorithm**):

   (a) Initialize each clique as its own component.

   (b) Iteratively add the highest-weight separator if it does not form a cycle.

5. Construct an adjacency list representation of the tree:

$$T = \{(C_i, C_j, S(i,j))\} \tag{3}$$

   ensuring that the graph remains a tree.

6. Store the junction tree structure, associating each clique with its variables and potential functions.

# 4 Computation of Marginals

After performing sum-product message passing in the junction tree, we compute the belief for each clique as the product of its local potential and all incoming messages. The marginal distribution for each variable is then computed by summing over the assignments of the other variables and normalizing the result.

### Steps

- **Belief Computation:** For each clique, compute the belief by multiplying the local potential with the messages received from neighboring cliques.
- **Marginalization:** For each variable within a clique, sum the belief over all assignments of the other variables to obtain its marginal distribution.
- **Normalization:** Normalize the marginal probabilities so that they sum to 1.

## 5 Top-$k$ Inference via Max-Product Message Passing

To obtain the $k$ most probable assignments (top-$k$ configurations), a max-product variant of message passing is used. In this approach, each clique maintains a priority queue (heap) that stores candidate assignments and their probabilities.

- **Initialization:** For each assignment in a clique, initialize a heap with a single candidate having the local potential as its score.
- **Message Computation:** For each message sent from a sender clique to a receiver clique, compute the product of the sender's local potential and the incoming messages (excluding the receiver). For each separator assignment, update the heap to retain only the top-$k$ candidates.
- **Final Assignment Extraction:** Once message passing is complete, extract the top-$k$ assignments from the root clique and normalize them using the partition function $Z$ computed during belief propagation.

## 6 References


- Rosetta Code - Bron–Kerbosch Algorithm
- Ermon Group CS228 Notes on Inference and Junction Tree
- Simplilearn Tutorial on Kruskal's Algorithm