# CS726 - Advanced Machine Learning
# Programming Assignment 1

**Saksham Rathi (22B1003)**
**Sharvanee Sonawane (22B0943)**
**Deeksha Dhiwakar (22B0988)**

February 2025

# 1 Triangulation

This step is implemented in the function `triangulate_and_get_cliques`. We first check if the graph is already triangulated, using the function `whether_triangulated` described below:

**Algorithm 1** Check if Graph is already Triangulated

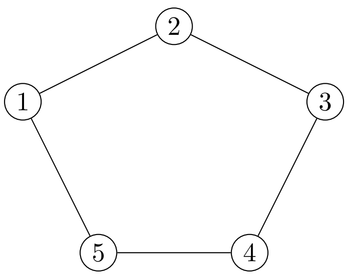
```
cycles ← Find all cycles in graph
for each cycle in cycles do
    if length of cycle ≥ 4 then
        if there is no shortcut (vertices connected by non-cycle edge) then
            return False
        end if
    end if
end for
return True
```

If the graph is already triangulated, we directly proceed with extracting the maximal cliques. If the graph is not triangulated, we first triangulate it using the minimum degree heuristic, as described in the pseudocode below:

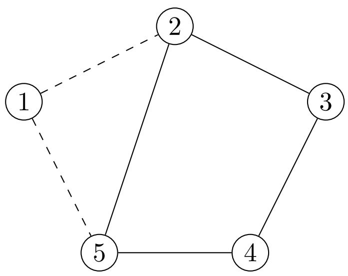
**Algorithm 2** Triangulation Process

```
vertices_left ← Set of all vertices
while vertices_left is not empty do
    vertex ← Vertex in vertices_left with minimum degree
    for each pair (i, j) of neighbours of vertex do
        if the graph does not contain an edge between i and j then
            Add edge (i, j) to the original graph
        end if
    end for
    Remove vertex from vertices_left
    Update graph by removing vertex and updating degrees and edges
end while
```

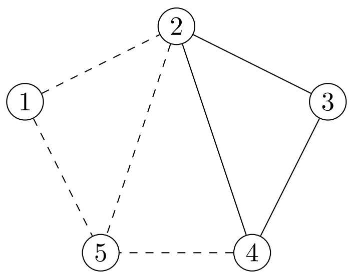
The figure below shows the run of the triangulation algorithm on an example graph:



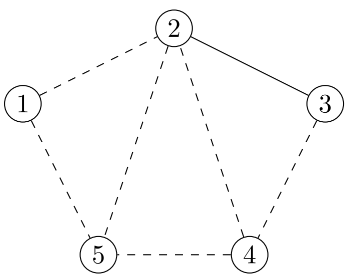

Step 1: Initial Graph



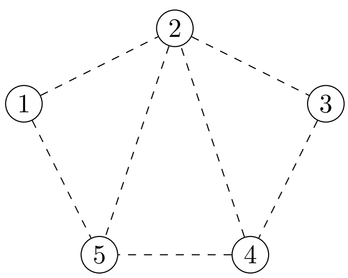

Step 2: Select 1, add (5,2), remove 1



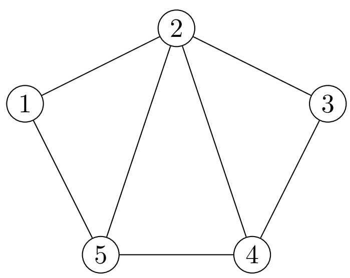

Step 3: Select 5, add (4,2), remove 5




Step 4: Select 4, remove 4




Step 5: Select 2, remove 2




Final Triangulated Graph

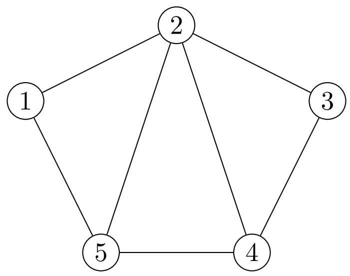
Once we obtain the triangulated graph, we extract the maximal cliques from it using the function `get_maximal_cliques`, which uses the Bron-Kerbosch algorithm described below:

**Algorithm 3** Bron–Kerbosch Algorithm for Finding Maximal Cliques

```
procedure BRON_KERBOSCH(current_clique, candidates, excluded, maximal_cliques)
    if candidates is empty and excluded is empty then
        Add current_clique to maximal_cliques
        return
    end if
    for each vertex v in candidates do
        Bron_Kerbosch(current_clique ∪ {v},
                candidates ∩ Neighbors(v),
                excluded ∩ Neighbors(v),
                maximal_cliques)
        Remove v from candidates
        Add v to excluded
    end for
end procedure
```

## 2 Junction Tree Construction

This step is implemented in the `get_junction_tree` function. We use the maximal cliques obtained after the triangulation process to create the junction tree while maintaining the running intersection property. Each clique in this tree retains its assigned potential values. Consider the given triangulated graph:




Final Triangulated Graph

From the triangulated graph, we obtain the maximal cliques:

$$C_1 = \{1, 2, 5\}, \quad C_2 = \{2, 3, 4\}, \quad C_3 = \{2, 4, 5\},$$

The intersection sets are as follows:

$$\begin{aligned} C_1 \cap C_3 = \{2, 5\}, &\quad |C_1 \cap C_3| = 2 \\ C_1 \cap C_2 = \{2\}, &\quad |C_1 \cap C_2| = 1 \\ C_3 \cap C_2 = \{2, 4\}, &\quad |C_3 \cap C_2| = 2 \end{aligned}$$

We construct edges with weights corresponding to intersection sizes:

$$(C_1, C_3, 2), \quad (C_3, C_2, 2), \quad (C_1, C_2, 1)$$

Using Kruskal's algorithm:

1. Sort edges: $(C_1, C_3, 2)$, $(C_3, C_2, 2)$, $(C_1, C_2, 1)$.
2. Add $(C_1, C_3)$ (weight 2).
3. Add $(C_3, C_2)$ (weight 2).

**Algorithm 4** Constructing a Junction Tree from Maximal Cliques

**Require:** Set of maximal cliques $\mathcal{C}$
**Ensure:** Junction tree satisfying the running intersection property
Initialize an empty priority queue $T$
**for all** pairs $(C_1, C_2)$ in $\mathcal{C}$ **do**
  Compute the intersection set $S = C_1 \cap C_2$
  **if** $S \neq \emptyset$ **then**
    Compute weight $w = |S|$
    Push $(-w, C_1, C_2)$ onto $T$ (negative weight for max heap)
  **end if**
**end for**

Initialize $parent[C] = C$ and $rank[C] = 0$ for all $C \in \mathcal{C}$

**function** FINDPARENT($C$)
  **if** $parent[C] \neq C$ **then**
    $parent[C] \leftarrow$ FindParent($parent[C]$)
  **end if**
  **return** $parent[C]$
**end function**

**function** UNION($C_1, C_2$)
  $root_1 \leftarrow$ FindParent($C_1$)
  $root_2 \leftarrow$ FindParent($C_2$)
  **if** $root_1 \neq root_2$ **then**
    **if** $rank[root_1] > rank[root_2]$ **then**
      $parent[root_2] \leftarrow root_1$
    **else if** $rank[root_2] > rank[root_1]$ **then**
      $parent[root_1] \leftarrow root_2$
    **else**
      $parent[root_2] \leftarrow root_1$
      $rank[root_1] \leftarrow rank[root_1] + 1$
    **end if**
    **return** True
  **end if**
  **return** False
**end function**

Initialize an empty set $MST$ (Minimum Spanning Tree)
**while** $T$ is not empty **do**
  Pop $(w, C_1, C_2)$ from $T$
  **if** UNION($C_1, C_2$) **then**
    Add edge $(C_1, C_2)$ to $MST$
  **end if**
**end while**
**return** $MST$

4. Ignore $(C_1, C_2)$ (weight 1) as it would form a cycle.

**Final Junction Tree:**

$$C_1 \leftrightarrow C_3 \leftrightarrow C_2$$

$$\begin{tikzcd} C_1 \arrow[r, dash, "2"] & C_3 \arrow[r, dash, "2"] & C_2 \end{tikzcd}$$

For any two cliques $C_i$ and $C_j$ containing the same variable $X$, all cliques along the unique path in the tree must contain $X$. This holds for all intersections.

# 3 Marginal Probability

Here is the pseudocode for sharing messages between the maximal cliques of the graph.

Firstly, we show how to calculate the Z value for the given graph.

**Algorithm 5** Computation of Partition Function $Z$

Arbitrarily root junction tree at $C_{root}$<br>
Initialize dicionary `clique_depths`<br>
Perform DFS from $C_{root}$ to populate `clique_depths`<br>
**function** SENDMESSAGE($C_{from}, C_{to}$)<br>
    Compute separator set $S = C_{from} \cap C_{to}$<br>
    Initialize message vector $M$ of size $2^{|S|}$<br>
    Modify clique potential based on incoming messages<br>
    **for** each state assignment in $C_{from}$ **do**<br>
        Compute corresponding separator index<br>
        Aggregate message value<br>
    **end for**<br>
    Store message $M(C_{from} \to C_{to})$<br>
**end function**<br>
**for** each clique from deepest to root **do**<br>
    Send messages to parent cliques<br>
**end for**<br>
**for** each clique from root to leaves **do**<br>
    Send messages to child cliques<br>
**end for**<br>
Compute partition function $Z$ using root clique potential and received messages<br>
**return** $Z$

Here is the pseudocode for computing the marginal probabilities in the graphical model using message passing.

**Algorithm 6** Computation of Marginal Probabilities

Initialize adjacency list for junction tree<br>
Retrieve partition function $Z$ using previously computed values<br>
Initialize marginal probability list $M$ with zeros<br>
**for** each variable $X_i$ in the graphical model **do**<br>
    Find a maximal clique $C$ containing $X_i$<br>
    Extract the potential function for clique $C$<br>
    **for** each neighboring clique $C'$ of $C$ **do**<br>
        Compute separator set $S = C \cap C'$<br>
        Retrieve message $M(C' \to C)$<br>
        **for** each assignment in $C$ **do**<br>
            Identify corresponding index in $S$<br>
            Multiply message values with clique potential<br>
        **end for**<br>
    **end for**<br>
    Compute marginal probability for $X_i$<br>
    Normalize values using $Z$<br>
**end for**<br>
**return** $M$