

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

```

1: cycles  $\leftarrow$  Find all cycles in graph
2: for each cycle in cycles do
3:   if length of cycle  $\geq 4$  then
4:     if there is no shortcut (vertices connected by non-cycle edge) then
5:       return False
6:     end if
7:   end if
8: end for
9: return True
  
```

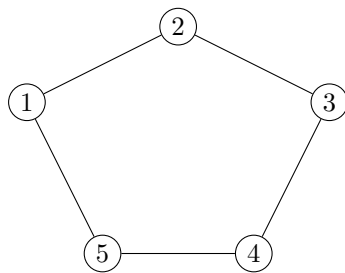
If the graph is already triangulated, we directly proceed with extracting the maximal cliques. If the graph is not triangulated, we first triangulate it using the minimum degree heuristic, as described in the pseudocode below:

Algorithm 2 Triangulation Process

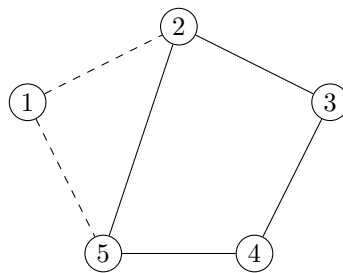
```

1: vertices_left  $\leftarrow$  Set of all vertices
2: while vertices_left is not empty do
3:   vertex  $\leftarrow$  Vertex in vertices_left with minimum degree
4:   for each pair  $(i, j)$  of neighbours of vertex do
5:     if the graph does not contain an edge between  $i$  and  $j$  then
6:       Add edge  $(i, j)$  to the original graph
7:     end if
8:   end for
9:   Remove vertex from vertices_left
10:  Update graph by removing vertex and updating degrees and edges
11: end while
  
```

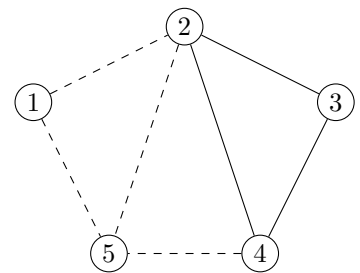
The figure below shows the run of the triangulation algorithm on an example graph:



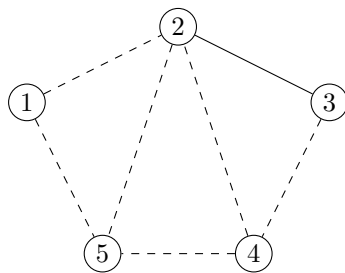
Step 1: Initial Graph



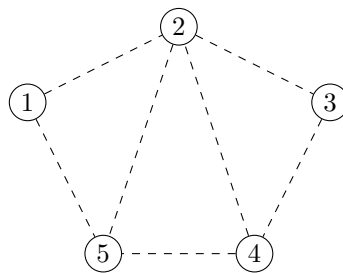
Step 2: Select 1, add (5,2), remove 1



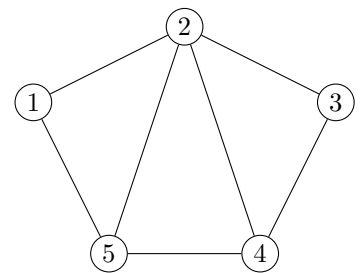
Step 3: Select 5, add (4,2), remove 5



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

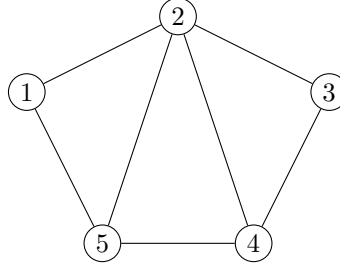
```

1: procedure BRON_KERBOSCH(current_clique, candidates, excluded, maximal_cliques)
2:   if candidates is empty and excluded is empty then
3:     Add current_clique to maximal_cliques
4:     return
5:   end if
6:   for each vertex  $v$  in candidates do
7:     BRON_KERBOSCH(current_clique  $\cup \{v\}$ ,
8:       candidates  $\cap$  Neighbors( $v$ ),
9:       excluded  $\cap$  Neighbors( $v$ ),
10:      maximal_cliques)
11:   Remove  $v$  from candidates
12:   Add  $v$  to excluded
13:   end for
14: 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:

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

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

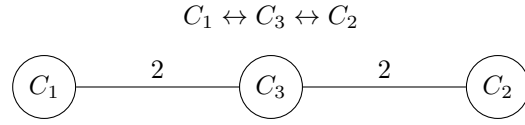
Require: Set of maximal cliques \mathcal{C}

Ensure: Junction tree satisfying the running intersection property

```
1: Initialize an empty priority queue  $T$ 
2: for all pairs  $(C_1, C_2)$  in  $\mathcal{C}$  do
3:   Compute the intersection set  $S = C_1 \cap C_2$ 
4:   if  $S \neq \emptyset$  then
5:     Compute weight  $w = |S|$ 
6:     Push  $(-w, C_1, C_2)$  onto  $T$  (negative weight for max heap)
7:   end if
8: end for
9:
10: Initialize  $parent[C] = C$  and  $rank[C] = 0$  for all  $C \in \mathcal{C}$ 
11:
12: function FINDPARENT( $C$ )
13:   if  $parent[C] \neq C$  then
14:      $parent[C] \leftarrow \text{FindParent}(parent[C])$ 
15:   end if
16:   return  $parent[C]$ 
17: end function
18:
19: function UNION( $C_1, C_2$ )
20:    $root_1 \leftarrow \text{FindParent}(C_1)$ 
21:    $root_2 \leftarrow \text{FindParent}(C_2)$ 
22:   if  $root_1 \neq root_2$  then
23:     if  $rank[root_1] > rank[root_2]$  then
24:        $parent[root_2] \leftarrow root_1$ 
25:     else if  $rank[root_2] > rank[root_1]$  then
26:        $parent[root_1] \leftarrow root_2$ 
27:     else
28:        $parent[root_2] \leftarrow root_1$ 
29:        $rank[root_1] \leftarrow rank[root_1] + 1$ 
30:     end if
31:     return True
32:   end if
33:   return False
34: end function
35:
36: Initialize an empty set  $MST$  (Minimum Spanning Tree)
37: while  $T$  is not empty do
38:   Pop  $(w, C_1, C_2)$  from  $T$ 
39:   if UNION( $C_1, C_2$ ) then
40:     Add edge  $(C_1, C_2)$  to  $MST$ 
41:   end if
42: end while
43: return  $MST$ 
```

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

Final Junction Tree:



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

```

1: Arbitrarily root junction tree at  $C_{root}$ 
2: Initialize dictionary clique_depths
3: Perform DFS from  $C_{root}$  to populate clique_depths
4: function SENDMESSAGE( $C_{from}, C_{to}$ )
5:   Compute separator set  $S = C_{from} \cap C_{to}$ 
6:   Initialize message vector  $M$  of size  $2^{|S|}$ 
7:   Modify clique potential based on incoming messages
8:   for each state assignment in  $C_{from}$  do
9:     Compute corresponding separator index
10:    Aggregate message value
11:   end for
12:   Store message  $M(C_{from} \rightarrow C_{to})$ 
13: end function
14: for each clique from deepest to root do
15:   Send messages to parent cliques
16: end for
17: for each clique from root to leaves do
18:   Send messages to child cliques
19: end for
20: Compute partition function  $Z$  using root clique potential and received messages
21: return  $Z$ 

```

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

Algorithm 6 Computation of Marginal Probabilities

```

1: Initialize adjacency list for junction tree
2: Retrieve partition function  $Z$  using previously computed values
3: Initialize marginal probability list  $M$  with zeros
4: for each variable  $X_i$  in the graphical model do
5:   Find a maximal clique  $C$  containing  $X_i$ 
6:   Extract the potential function for clique  $C$ 
7:   for each neighboring clique  $C'$  of  $C$  do
8:     Compute separator set  $S = C \cap C'$ 
9:     Retrieve message  $M(C' \rightarrow C)$ 
10:    for each assignment in  $C$  do
11:      Identify corresponding index in  $S$ 
12:      Multiply message values with clique potential
13:    end for
14:   end for
15:   Compute marginal probability for  $X_i$ 
16:   Normalize values using  $Z$ 
17: end for
18: return  $M$ 

```
