CS 726: Programming Assignment 1

Deeptanshu Malu Deevyanshu Malu Neel Rambhia

1 Preprocessing

The preprocessing step involves converting the given data into a suitable format for further processing. We have done the following preprocessing steps:

1. **Clique Potentials:** Converted all the cliques and potentials data into a dictionary format. The keys of the dictionary are the cliques and the values are the potentials.
2. **Edges:** Converted all the cliques into a set of tuples, where each tuple consists of two nodes that are connected by an edge.
3. **Nodes:** Converted the edge set into a set of all the nodes in the graph.

2 Triangulation

Here are some of the functions we have defined for the triangulation and maximum clique finding process:

1. **is_simplicial:** This function checks if a given node is simplicial in the graph. A node is simplicial if the neighbors of the node form a clique.

```
function is_simplicial(adjacency_list, node):
    for every pairs of neighbors of node:
            if the pair is not connected:
                return False
    return True
```

2. **find_simplicial_vertex:** This function finds a simplicial vertex in the graph. If no simplicial vertex is found, it returns None.

```
function find_simplicial_vertex(adjacency_list):
    for each node in the graph:
        if the node is simplicial:
            return node
    return None
```

3. **make_vertex_simplicial:** This function makes a given node simplicial by adding edges between all the neighbors of the node.

```
function make_vertex_simplicial(adjacency_list, node):
    for every pair of neighbors of the node:
        if the pair is not connected:
            add an edge between the pair
    return adjacency_list
```

4. **chordal_graph_with_heuristic:** This function constructs a chordal graph from the given graph using the heuristic method. It returns a list of maximal cliques in the chordal graph.

```
function chordal_graph_with_heuristic(adjacency_list):
    create an temporary adjacency list from the given adjacency list
    create an empty list for elimination order
    create an empty list for cliques

    while the length of elimination order is less than
    the length of adjacency list:
        find a simplicial node in the graph
        if a simplicial node is found:
            add the simplicial node to the elimination order
            create a clique with the simplicial node
            add all the neighbors of the simplicial node to the clique
            add the clique to the list of cliques
            remove the simplicial node from the temporary adjacency list
        else:
            find the node with the least degree in the graph
            make the node simplicial
            add the node to the elimination order
            create a clique with the node
            add all the neighbors of the node to the clique
            add the clique to the list of cliques
            remove the node from the adjacency list

    create a list of maximal cliques from the list of cliques
    return the list of maximal cliques
```

```
def chordal_graph_with_heuristic(adjlist):
    chordal_adjlist = deepcopy(adjlist)
    elimination_order = []
    cliques = []

    while len(elimination_order) < len(adjlist):
        simplicial_node = find_simplicial_vertex(chordal_adjlist)

        if simplicial_node is not None:
            elimination_order.append(simplicial_node)
            clique = {simplicial_node}
```

```
            neighbours = chordal_adjlist[simplicial_node]
            for neigh in neighbours:
                if all(neigh in chordal_adjlist[i] for i in clique):
                    clique.add(neigh)
            cliques.append(clique)
            for neighbor in chordal_adjlist[simplicial_node]:
                chordal_adjlist[neighbor].remove(simplicial_node)
            del chordal_adjlist[simplicial_node]
        else:
            degrees = {
                node: len(neighbors)
                for node, neighbors in chordal_adjlist.items()
            }
            least_degree_node = min(degrees, key=degrees.get)
            chordal_adjlist = make_vertex_simplicial(
                chordal_adjlist, least_degree_node
            )
            elimination_order.append(least_degree_node)
            clique = {least_degree_node}
            neighbours = chordal_adjlist[least_degree_node]
            for neigh in neighbours:
                if all(neigh in chordal_adjlist[i] for i in clique):
                    clique.add(neigh)
            cliques.append(clique)
            for neighbor in chordal_adjlist[least_degree_node]:
                chordal_adjlist[neighbor].remove(least_degree_node)
            del chordal_adjlist[least_degree_node]

    max_cliques = []
    for clique in cliques:
        is_subset = False
        for m_clique in max_cliques:
            if clique.issubset(m_clique):
                is_subset = True
                break
        if not is_subset:
            max_cliques.append(clique)

    return max_cliques

max_cliques = chordal_graph_with_heuristic(self.adjlist)
self.max_cliques = max_cliques
print("Adjacency List:")
for node, neighbors in self.adjlist.items():
    print(f"{node}: {neighbors}")
print("Maximal Cliques:", self.max_cliques)
```