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

2.1 triangulate_and_get_cliques

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.
2. **find_simplicial_vertex:** This function finds a simplicial vertex in the graph. If no simplicial vertex is found, it returns None.
3. **make_vertex_simplicial:** This function makes a given node simplicial by adding edges between all the neighbors of the node.
4. **chordal_graph_with_heuristic:** In this function we do the following:
   (a) Find a simplicial vertex in the graph.
   (b) If a simplicial vertex is found, take its neighbors and store it in the maximum clique list.
   (c) If no simplicial vertex is found, take a node based on the heuristic that the node has the minimum degree. Make this node simplicial and take its neighbors and store it in the maximum clique list.
   (d) Repeat the process until all the nodes are covered.
   (e) If a clique found is already a subset of a previously found clique, discard it.
   (f) Return the maximal cliques found.

3 Junction Tree Construction

3.1 get_junction_tree

In this section, we have defined the following functions for constructing the junction tree:

1. **create_junction_graph**: This function creates a junction graph from the found maximal cliques. We iterate over all the maximal cliques and add an edge between two maximal cliques if they have a common node. We also assign a weight to each edge which is the size of the intersection of the two maximal cliques.

2. **make_junction_tree**: This function creates a junction tree from the junction graph. We first find the maximum spanning tree of the junction graph using Kruskal's algorithm. This tree is the junction tree.

3.2 assign_potentials_to_cliques

In this section, we have defined the following functions for assigning potentials to the cliques:

1. **multiply_potentials**: This function takes two potentials and there corresponding cliques and multiplies them to get a new potential and a new clique.

After this we iterate over all the maximal cliques and for each clique, we find all the clique potentials that are subsets of the maximal clique. We multiply all these potentials to get the potential for the maximal clique.

4 Marginal Probability

4.1 get_z_value

In this section, we have defined the following functions for finding the Z value:

1. **create_empty_message_dict**: This function creates an empty message dictionary for all the cliques in the junction tree. The dictionary has the sender clique as the key and the value is a dictionary with the receiver clique as the key and its value is the message from the sender clique to the receiver clique.

2. **multiply_messages**: This function takes a potential of the clique or a precomputed value (potential is already multiplied with some messages) and a message to be multiplied as input and it multiplies them to get a new potential.

3. **condense_message**: This function takes a potential and the nodes to be marginalized on and marginalizes the potential to get a new potential.

4. **send_message**: In this function, we do a forward pass of message passing. These are the steps we follow:

    (a) We maintain a list of nodes who have to send messages to their neighbors. This is initialized with the leaf nodes of the junction tree.

   (b) We iterate over all the nodes in the list and send messages from the node to all its neighbors using the `multiply_messages` and `condense_message` functions.
   (c) When a message is sent from a node to its neighbor, we update the message dictionary.
   (d) After iterating over all the nodes in the list, we remove the nodes from the list and add their neighbors to whom messages were sent to the list.
   (e) We repeat the process until the list is empty and final non-empty value of the list is the root of the junction tree.

5. **receive_message**: In this function, we do a backward pass of message passing. These are the steps we follow:
   (a) We maintain a list of nodes who have to send messages to their neighbors. This is initialized with the root of the junction tree.
   (b) We iterate over all the nodes in the list and send messages from the node to all its neighbors using the `multiply_messages` and `condense_message` functions.
   (c) When a message is sent from a node to its neighbor, we update the message dictionary.
   (d) After iterating over all the nodes in the list, we remove the nodes from the list and add their neighbors to whom messages were sent to the list.
   (e) We repeat the process until the list is empty.

6. **calc_z**: This function calculates the Z value using the fully updated message dictionary. We simply choose any maximal clique and marginalize it on all it component nodes to get the Z value.

5 MAP Assignment

6 Top k Assignments