CS5340 - Lab2 Part1 Report

Liu Yichao A0304386A

September 23, 2024

factor_utils.py

The functions *factor_product*, *factor_marginalization* are just the same as lab1. The function *factor_evidence* omits the observed variables based on the implementation in lab1.

jt_construction.py

The function *_get_jt_clique_and_edges* contains these steps:

1. Construct graph from nodes and edges, stored in an adjacency list
2. Choose an elimination order, perform the elimination process to reconstitute the graph by adding edges to the original MRF. (This process is also called triangulation)
3. Use elimination cliques as the clique nodes of the junction tree. The weight of the edge between two cliques is the size of the intersection of the two cliques.
4. Use *Prim* algorithm to find the maximum spanning tree, which is also junction tree. Here *Prim* is better than *Kruskal* algorithm because we are generating from a complete graph.

The function *_get_clique_factors* constructs the factors for each clique node in the junction tree.

main.py

The function *_update_mrf_w_evidence*, before constructing the junction tree, updates the graph with evidences, by removing observed nodes and keeping probabilities corresponding to evidences.

The function *_get_clique_potentials* uses sum-product algorithm to get the potentials for each clique node in the junction tree, which is almost the same as lab1.

The function *_get_node_marginal_probabilities* compute marginal probabilities for each node in the junction tree. It is a brute-force method here as nodes inside a clique are fully-connected. Since each node may belong to multiple cliques with the same marginal probability, we start this process **from the smallest clique** to minimize the calculation complexity, which in practice is maintaining a bool list to track whether a node has been calculated in another smaller clique. This would notably reduce the complexity of the algorithm.