CS5340 - Lab 1 Report

Liu Yichao A0304386A

September 14, 2024

Abstract

This lab is about the implementation of both the *sum-product* and *max-product* algorithm for inference in tree-like graphical models. For inference here we mean

- computing the marginal probability of each variable given the observed values of some variables. (calculate the joint probability and marginalize)
- computing the max probability assignment of the unobserved variables given the observed values of some variables. (find MAP & configuration)

Specifically, with the given python framework, I implemented *compute_marginals_bp* and *map_eliminate* functions respectively, and some other helper functions also.

Difficulties

The main problem I encountered is to understand these 3 concept of increasing difficulty. I understand them during the implementation of functions:

1. **factor**: represents the joint probability of a set of variables. A factor object has 4 fields: var, card, val, val_argmax, and it can be multiplied with another factor object. The function *factor_marginalization* shows that a factor object can be marginalized over a set of variables.
2. **message**: the marginalization of the product of a list of factors. a message is acutally a factor object, which is the marginalization of the product of a list of factors like

$$m_{ji}(x_i) = \sum_{x_j} (\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{kj}(x_j))$$

3. **collect / distribute**: The function *collect* and *distribute* is acutally the process of message passing under some protocol. In a tree-like graph, after assigning a root, the message passing is done in a bottom-up and top-down way, with respect to the collect and distribute process, during which we compute the message for each edge in the graph, and finally get the marginal probability of each variable with

$$p(x_i) = \psi^E(x_i) \prod_{k \in N(i)} m_{ki}(x_i)$$

   This is a trade-off between the complexity of the computation and the memory usage. In *compute_marginal_naive* function, the largest size of elimination clique is decided by the elimination order, which is NP-hard to find the optimal order.
   In *compute_marginal_bp* we use auxiliary memory space to store messages, with the benefit of reducing the computation complexity from exponential to polynomial.
   The *max-product* algorithm is similar to the *sum-product* algorithm, but with the max operation instead of the sum operation.

$$m_{ji}^{max}(x_i) = \max_{x_j} (\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{kj}^{max}(x_j))$$