Lab4 Report

Name: WU YIHANG Email: e1216288@u.nus.edu. Student ID: A0285643W

Part 1

- `_sample_step(nodes, proposal_factors)`

  In this function, nodes are sampled and during the sampling process, the distribution of proposal factors will be updated according to current node's sampling value by `factor_evidence` function.

- `_get_conditional_probability(target_factors, proposal_factors, evidence, num_iterations)`

  Some helper functions are added to realize this function

  - `find_order(proposal_factors)`: In this function, a nx Graph are built to produce topological order of proposal factors
  - `get_probability_from_factor(factor, sample)`: This function will return a probability of factor in current sample by using `assignment_to_index` to find the exact index in factor's value list
  - `find_factor_for_node(node,factors)`: This function will return a factor that represent the node's conditional distribution, I realize this function by a simple iteration and search
  - `calculate_weight(sample, evidence,target_factors_pos,proposal_factors_pos)`: This function will calculate weight based on one sample. It will use the 2 functions above to find the target probability and proposal probability based on current sample and evidence, and then compute and return the weight.
  - Based on the four helper function above, we can implement the `_get_conditional_probability`:
    - First, update the proposal factors according to evidence and modify the order into topological order
    - Second, I prepare 2 dictionary: `target_factors_pos,proposal_factors_pos` to save model's time in interation.
    - Third, we conduct the sampling process and calculate weight iteratively. I use a dictionary to record the different sample, if current sample appeared before, the model won't compute it again and this can help us save time.
    - Last, the normalization will be done and finally we will get the result.

Part 2

- `_sample_step(nodes, factors, in_samples)`

  In this function, for each node, I fix other node's sampling value, and iterate current node's value to compute current node's probability distribution, and based on the distribution I conduct the current node's random sampling.

- `_get_conditional_probability`:

  This function is easier then the importance sampling, and my implementation pipeline is as follows:

  - First, the the initial samples and factors are updated according to the evidence

- Second, conditional_prob is initialized
- Third, the sampling process is conducted, and if the iteration steps are larger than the threshold(num_burn_in), we will record the sample's occurrance time.
- Last, we will compute the probability of all samples according the occurrence time.