600.676 Machine Learning Homework 2

2.1.1 Deliverables [14 points]

In order to construct a minimal I-map for the marginal distribution, we should remain all dependencies of original network in the new network. Denote the original network as $G = (V, E)$, the new network as $G' = (V', E')$, where $V' = V \setminus \{FluRate\}$. As Figure 1 shown, eliminating node "FluRate" is equivalent to node "FluRate" is unobserved.
Initially, let $E(G') = \emptyset$. Then, we should preserve all the active paths given "FluRate" unobserved.
For IsFluSeaon in G, the paths

$$IsFluSeaon \rightarrow FluRate \rightarrow MaryGetsFlu,$$
$$IsFluSeaon \rightarrow FluRate \rightarrow JohnGetsFlu$$

are active when FluRate is unobserved.
Hence, we should add edges $IsFluSeason \rightarrow MaryGetsFlu$ and $IsFluSeason \rightarrow JohnGetsFlu$ to $E(G')$ to hold the dependencies
Similarly, for PreviousFluRate in G, the paths

$$PreviousFluRate \rightarrow FluRate \rightarrow JohnGetsFlu,$$
$$PreviousFluRate \rightarrow FluRate \rightarrow MaryGetsFlu$$

are active when FluRate is unobserved.
Then, we should add edges $PreviousFluRate \rightarrow MaryGetsFlu$ and $PreviousFluRate \rightarrow JohnGetsFlu$ to

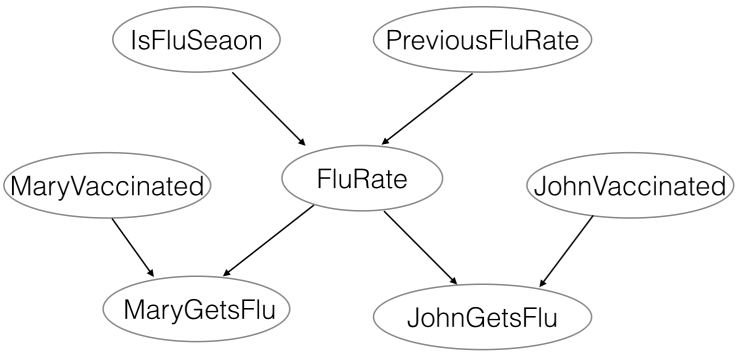

Figure 1: Orignal bayes-network

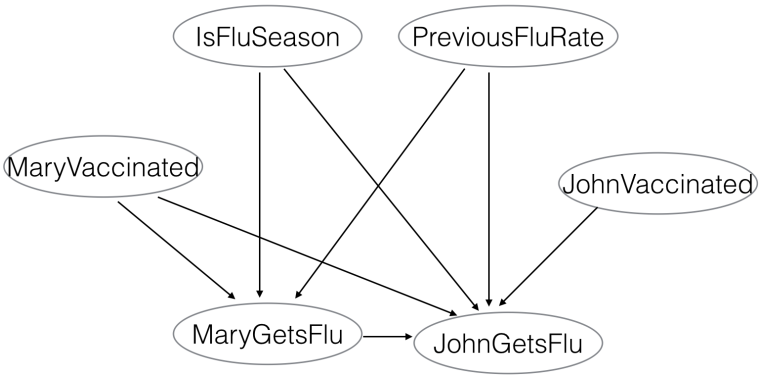

Figure 2: Modified bayes-network

$E(G')$ to hold dependencies.
Next, in G, $MaryGetsFlu \leftarrow FluRate \rightarrow JohnGetsFlu$ is also active when FluRate is unobserved. Then, we should add edge between $MaryGetsFlu$ and $JohnGetsFlu$. Either direction is okay, since G is a "symmetric" graph. In G', we choose to add an edge $MaryGetsFlu \rightarrow JohnGetsFlu$.

For MaryVaccinated, in G, the path $MaryVaccinated \rightarrow MaryGetsFlu \leftarrow FluRate \rightarrow JohnGetsFlu$ is active when $MaryGetsFlu$ is observed and $FluRate$ is unobserved. But the present G', the only path from $MaryVaccinated$ to $JohnGetsFlu$, $MaryVaccinated \rightarrow MaryGetsFlu \rightarrow JohnGetsFlu$ is inactive when $MaryGetsFlu$ is observed. Thus, in order to preserve the dependence, we should add edge $MaryVaccinated \rightarrow JohnGetsFlu$ to $E(G')$.

Then, in G, consider the path $JohnVaccinated \rightarrow JohnGetsFlu \leftarrow FluRate \rightarrow MaryGetsFlu$, this path is active if and only if $JohnGetsFlu$ is observed and $FluRate$ is unobserved. In present G', since we have added an edge $MaryGetsFlu \rightarrow JohnGetsFlu$, hence there is a v-structure $MaryGetsFlu \rightarrow JohnGetsFlu \leftarrow JohnVaccinated$, and this v-structure is active when $JohnGetsFlu$ is observed. Therefore, the dependence holds in G', we don't need to add an edge $JohnVaccinated \rightarrow MaryGetsFlu$.

Let's consider path $MaryVaccinated \rightarrow MaryGetsFlu \leftarrow FluRate \leftarrow PreviousFluRate$, and $JohnVaccinated \rightarrow JohnGetsFlu \leftarrow FluRate \leftarrow IsFluSeaon$ in G. Such paths are active if and only if $FluRate$ is unobserved and $MaryGetsFlu$ is observed. In present G' when $MaryGetsFlu$ is observed, we can see the v-structures $MaryVaccinated \rightarrow MaryGetsFlu \leftarrow PreviousFluRate$, and $JohnVaccinated \rightarrow JohnGetsFlu \leftarrow IsFluSeaon$ are active as well. We don't need to add more edges to $E(G')$ since the dependences holds.

Finally, let's consider $MaryVaccinated \rightarrow MaryGetsFlu \leftarrow FluRate \leftarrow IsFluSeason$, and $JohnVaccinated \rightarrow JohnGetsFlu \leftarrow FluRate \leftarrow PreviousFluRate$ in G. Such paths are active if and only if $FluRate$ is unobserved and $MaryGetsFlu$ is observed. In present G', when $MaryGetsFlu$ is observed, the v-structures $MaryVaccinated \rightarrow MaryGetsFlu \leftarrow IsFluSeason$, and $JohnVaccinated \rightarrow JohnGetsFlu \leftarrow PreviousFluRate$ are also active. Therefore, the dependences hold in G', we don't need to add any more edges to $E(G')$

Above all, we have obtained our modified network structure, shown in Figure 2.

To generate the corresponding cpd, network file, we run our bayes-query, the network-F2.txt, and cpd-F2.txt is stored in the zip file.

2.1.2 Analytical Questions [6 points]
1. [6 points] Generalize the procedure you used to solve the preceding problem into a node elimination algorithm. That is, define an algorithm that transforms the structure $\mathcal{G}$ into $\mathcal{G}'$ such that one of the nodes X_i of $\mathcal{G}$ is not in $\mathcal{G}'$ and $\mathcal{G}'$ is an I-map of the marginal distribution over the remaining variables as defined by $\mathcal{G}$.

We can use variable elimination algorithm to transform a structure $\mathcal{G}$ into $\mathcal{G}'$. The pseudocode is offered. Besides the pseudocode, we also provide a mathematical proof based on the knowledge of graphical model.

The pseudocode is the following:

```
#Construct network
def constructNetwork():
for node in network.node:
node.parent=getParent(node,network)
    node.cpd=getCpd(node,cpdfile)
    node.var=getVar(node,network)

#Assume var_eliminate is the parameter that we want to eliminate
#Eliminate var_eliminate in the cpd of node by marginalization

def elimination():
newCpd={}
for entry in newCpd:
     newkey=oldkey.remove(var_eliminate)
        #If the variables share the same key, then
        #sum them up.
#Update the newCpd
```

Proof of Mathematics Since there exists a probability distribution $\mathbb{P}$ that factorizes structure $\mathcal{G}$, and suppose that there exists a variable ordering $X_1, X_2, \ldots, X_n$ that the original structure is the minimal I-map generated by this variable ordering. Then in order to generate a new structure $\mathcal{G}'$ such that one of the nodes X_i of $\mathcal{G}$ is not in $\mathcal{G}'$ and $\mathcal{G}'$ is an I-map of the marginal distribution over the remaining variables as defined by $\mathcal{G}$, consider the property of minimal I-map and construct the structure by the variable ordering

$$X_1, X_2, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n,$$

we have this new structure is the $\mathcal{G}'$ that we desire.

Claim 1: In the new structure $\mathcal{G}'$ as we construct above, each parent of X_i is directly connected to all directed descents of X_i and each directed descent of node X_i is directly connected from all parents of X_i. And all the other edges remain the same.

Proof. Consider any nodes X_j that belongs to the set of X_i directed descents, since the original structure $\mathcal{G}$ is generated by $X_1, X_2, \ldots, X_n$, then we have $\mathrm{Pa}(X_i) \subseteq \{X_1, \ldots, X_{j-1}\}$. Since in $\mathcal{G}$, the node $X_i \in U(X_j)$ by X_i is the parent of X_j. In $\mathcal{G}'$, if $Y \in \mathrm{Pa}(X_i)$ and $Y \notin U(X_j)$, then we have $X_j \perp Y | U(X_j)$. However, back to $\mathcal{G}$, there exists a path: $Y \to X_i \to X_j$, and X_i is not observed, then this path should be active. Therefore, $X_j \perp Y | U(X_j)$ is false. Then we have $\mathrm{Pa}(X_i) \subseteq U(X_j)$, that is to say each directed descent of node X_i is directly connected with all parents of X_i. □

Claim 2: The marginal distribution $\mathbb{P}_{\mathcal{G}}$ remains the same as the marginal distribution $\mathbb{P}_{\mathcal{G}'}$. (i.e., Any queries to $\mathcal{G}$ which do not include the variable X_i should return the same result as the same query to $\mathcal{G}'$)

Proof. By **Theorem 4.9**,that is if a distribution P_B factorizes according to $\mathcal{G}$, then $\mathcal{G}$ is a I-map for P, in Probabilistic Graphical Models by Koller, since $\mathcal{G}'$ is constructed from $\mathcal{G}$ by algorithm minimal I-map without changing factorization $\mathbb{P}$, therefore, $\mathbb{P}_{\mathcal{G}} = \mathbb{P}_{\mathcal{G}'} = \mathbb{P}$, then any queries to $\mathcal{G}$ which do not include the variable X_i should return the same result as the same query to $\mathcal{G}'$. □

Node Elimination Algorithm:

1. Input original structure $\mathcal{G}'$ and eliminating node X_i.
2. Search for the set of parents of node X_i in $\mathcal{G}$, say $\text{Pa}(X_i)$.
3. Search for the set of directed descent of node X_i in $\mathcal{G}$, say $\text{DDes}(X_i)$.
4. Eliminate node X_i and all the edges incident with X_i.
5. Link all the nodes in $\text{Pa}(X_i)$ to all the nodes in $\text{DDes}(X_i)$, the direction is from $\text{Pa}(X_i)$ to $\text{DDes}(X_i)$. Then we get $\mathcal{G}'$.

2.2.1 Analytical Questions [16 points]
1. [8 points] Prove that the above strategy is a sound criterion for determining whether Z is a requisite probability node for $P(X|Y)$ in $\mathcal{G}$. Specifically, show that if this criterion fails to identify Z as a requisite node, then for all pairs of networks $\mathcal{B}_1, \mathcal{B}_2, P_{\mathcal{B}_1}(X|Y) = P_{\mathcal{B}_2}(X|Y)$.

Proof. If this criterion fails to identify Z as a requisite node, i.e., there exists no active path from $\hat{Z}$ to X given Y. More precisely, $Z \notin Y$, otherwise, Z will be observed and the CPDs of Z are meaningless.
Now consider $\hat{Z}, Z, X$ in $\mathcal{G}'$, define G as the subgraph with all the nodes in $\mathcal{G}'$ except $\hat{Z}, Z, X$ and $Y \subseteq G$. Then we have

$$\hat{Z} \to Z \leftrightarrow G \leftrightarrow X.$$

There are two cases:

- $\hat{Z} \to Z \to G \leftrightarrow X$. Since we have there exists no active path from $\hat{Z}$ to X given $Y \subseteq G$, and $\hat{Z} \to Z \to G_1 \in G$ is an active triple, then the block must occur in G that block $\hat{Z}$ to X, and also block Z to X, which implies $P_{\mathcal{B}_1}(X|Y) = P_{\mathcal{B}_2}(X|Y)$.
- $\hat{Z} \to Z \leftarrow G \leftrightarrow X$. Since we know $Z \notin Y$, then by the property of V-structure, when Z is not observed, $\hat{Z} \to Z \leftarrow G_1 \in G$ is an inactive triple. Then **Claim:** in the rest part $Z \leftarrow G \leftrightarrow X$, $P_{\mathcal{B}_1}(X|Y) = P_{\mathcal{B}_2}(X|Y)$.
  Without loss of generality, prove this by showing it does not hold in each path with the first triple with structure $\hat{Z} \to Z \leftarrow G_1$. Since Z is not directly connected with X, then $X \notin N(Z)$, therefore, all each $G_1 \in N(Z)$, we have $G_1 \neq X$. By conditions of our problem, $P(X|\text{Pa}(X))$ remains the same. Then since Z is a descent of G_1 and even there exists an active path between Z and X, Z still not be an ascent of X by structure $Z \leftarrow G_1$, therefore, we have

  $$P(X|Y) = P(X|(Y \cap \text{Pa}(X))) = \sum_{y \in \text{Pa}(X) \backslash Y} P(X|\text{Pa}(X)) P(\text{Pa}(X) \backslash Y)$$

  Since $P_{\mathcal{B}_1}(X|\text{Pa}(X)) = P_{\mathcal{B}_2}(X|\text{Pa}(X))$ by condition and Z does not affect the parents of X by Z is not X's ascent, therefore, $P_{\mathcal{B}_1}(\text{Pa}(X) \backslash Y) = P_{\mathcal{B}_2}(\text{Pa}(X) \backslash Y)$. Then we have $P_{\mathcal{B}_1}(X|Y) = P_{\mathcal{B}_2}(X|Y)$.

Combine two cases above, we have the above strategy is a sound criterion for determining whether Z is a requisite probability node for $P(X|Y)$ in $\mathcal{G}$. □

2. [8 points] Show that this criterion is weakly complete (like d-separation) in the sense that, if it identifies Z as a requisite in $\mathcal{G}$, there exists some pair of networks $\mathcal{B}_1, \mathcal{B}_2$ such that $P_{\mathcal{B}_1}(X|Y) \neq P_{\mathcal{B}_2}(X|Y)$.

Proof. If it identifies Z as a requisite in $\mathcal{G}$, then there exists an active path from $\hat{Z}$ to X, i.e., there also exists an active path from Z to X. Then in order to that there exists some pair of networks $\mathcal{B}_1, \mathcal{B}_2$ such that $P_{\mathcal{B}_1}(X|Y) \neq P_{\mathcal{B}_2}(X|Y)$, construct $\mathcal{G}$ as

$$\hat{Z} \to Z \to G_1 \to X$$

and $Y = \emptyset$, then it is obvious that there exists an active path from $\hat{Z}$ to X, so as Z to X. Then for some altering CPD of Z, we have $P_{\mathcal{B}_1}(X|Y) \neq P_{\mathcal{B}_2}(X|Y)$. Then this criterion is weakly complete. □

4.1.4 Parameter Estimation [40 points]

I create a estimate-param to train the data. The result of 'test-cpd-grid10x10-t10.txt' match 'cpd-grid10x10-t10.txt' CPD file within 2 decimal places.

```
./estimate-params network-grid10x10-t10.txt training-grid10x10-t10.txt test-cpd-grid10x10-t10.txt
```

Then, we use it to generate other cpd files with different parameters.

4.1.5 Analytical Questions [10 points]
1. [3 points] An important advantage of sharing parameters is the lessening of the effects of *overfitting*. Explain why. Why is this issue important when doing parameter estimation in this assignment?

Proof. Overfitting effects occur when the number of parameters is too much larger than the observations. However, overfitting may not make the model more accurate. Sometimes, overfitting has negative effects on model. For example, consider a linear model with n samples, for the correct model, it is enough to have two parameters; if we construct a model with n parameter as a polynomial P, then P is sure to go through these n points, but when we predict another points, the points computed by polynomial model P will be away from the observing points.
Sharing parameters will reduce the number of parameters, particularly, if our samples are much more than the necessary samples to train the parameters , then such model with too many will cause too much noise when predicting new points.
In this assigment, for Observation model, our sharing parameters is T (time) with different status. Since if we do not have enough samples when $t = t_0$, the probability of $P(Observation\_t|PositionRow\_t, PositionCol\_t)$, we compute **without sharing parameter**, will not be accuracy compare to the probability of $P(Observation\_t|PositionRow\_t, PositionCol\_t)$, we compute **with sharing parameter**, since by our experience particular time value should hardly affect the probability of making a particular observation, we can use the probability at other t to estimate it. Therefore, the distributions you learn should be the same across all values of t.
For Motion model, our sharing parameter are T (time) and particular positions (i and j values). Since we assume the probability of ending up in row i after moving forward from row $i-1$ should be the same as moving from row $j-1$ to j. Thus, you will learn parameters for the following templates:

$$P(PositionRow\_t = i|PositionRow\_t - 1 = i - 1, Action\_t - 1)$$

$$P(PositionRow\_t = i | PositionRow\_t - 1 = i + 1, Action\_t - 1)$$
$$P(PositionRow\_t = i | PositionRow\_t - 1 = i, Action\_t - 1)$$
$$P(PositionCol\_t = j | PositionRow\_t - 1 = j - 1, Action\_t - 1)$$
$$P(PositionCol\_t = j | PositionRow\_t - 1 = j + 1, Action\_t - 1)$$
$$P(PositionCol\_t = j | PositionRow\_t - 1 = j, Action\_t - 1)$$

□

2. [3 points] Describe at least one property of the robot localization setting (with regard to either the motion model or observation model) that you cannot model with the shared parameterization in this assignment, but that you could model if we used an explicit parameterization(i.e., we learn the entire CPT for particular positions or time steps). It does not have to be a property of the environment that we describe in this handout; you can construct something relevant that you might like to model in this setting.

Proof. In Motion model, we should not model by using sharing position parameter (i, j). For example: $P(PositionRow\_t = i | PositionRow\_t - 1 = i - 1, Action\_t - 1)$ arbitrarily assume that all the columns at time $t - 1$ are independent with rows. However, in our model, it does not make sense, more precisely, by using explicit parameterization, the CPD of $PositionRow\_t$ should be

$$P(PositionRow\_t = i | PositionRow\_t - 1 = i - 1, PositionCol\_t - 1 = j, Action\_t - 1).$$

Sharing parameter will cause the effect of underfitting. □

3. [4 points] Describe a way to combine the advantages of both approaches. That is, come up with a model design that can potentially learn unique values for all CPT entries, but also has shared parameters which reduce overfitting. Sketch the idea in a few sentences, and explain the advantages over other models.

Basic idea: Combine part 1,2 above, we could sharing parameter T and use explicit parameter of position (i, j). That is to consider the probability:

$$P(Observation\_t | PositionRow\_t; PositionCol\_t) = P(Observation | PositionRow; PositionCol),$$

the above formula sharing the parameter t, and the probability:

$$P(PositionRow\_t = i | PositionRow\_t - 1 = i - 1, PositionCol\_t - 1 = j, Action\_t - 1).$$

which take the influence from column into consideration.

4.2.1 Analytical Questions: Clique Tree [20 points]
1. [2 points] Describe the process you followed to make the clique tree. Note one or two points in the processes where you could have obtained a (slightly) different clique tree by making different choices.

In Step 2: Create a chordal graph by triangulation, we have more than one choice to remove any chordless cycles with 4 or more nodes. For example, in this assignment, we have

$$PositionRow\_0, PositionCol\_0, PositionRow\_1, PositionCol\_1$$

forming a chordless cycle with 4 nodes, then we could either connect $PositionRow\_0$ with $PositionCol\_1$ or $PositionRow\_1$ with $PositionCol\_0$ to triangulation, and in this process we could obtain different clique trees.
In Step 5: Since we could not ensure that the maximal spanning tree over this cluster tree is unique, then we could obtain different clique trees.

2. [2 points] Demonstrate that the running intersection property holds in the resulting clique tree. A formal proof is not required, simply explain how this property holds.

Suppose we have G is a chordal graph, then define $\mathcal{C}^*$ as the set of maximal cliques in G. Since by **Proposition 32** in **Lecture Notes from Laurent Younes**, we have a cluster graph $(\mathcal{C}^*, \mathbb{E})$ is connected if and only if G is connected. Now consider the relation between maximal spanning tree of $(\mathcal{C}^*, \mathbb{E})$ and junction tree in $(\mathcal{C}^*, \mathbb{E})$, we claim that:

$$\{\mathcal{T} : \mathcal{T} \text{ is a clique tree over } (\mathcal{C}^*, \mathbb{E})\} = \{\mathcal{T}' : \mathcal{T}' \text{ is maximal spanning tree over } (\mathcal{C}^*, \mathbb{E})\}.$$

Here is the proof: (**Proposition 33** in **Lecture Notes from Laurent Younes**)

The lecture notes of Laurent Younes has been uploaded on our github:

`https://github.com/motian12ps/machine-learning-data-to-model/tree/master/hw2`

The corresponding proof is on page 60.

Proof. Since $\mathcal{T}$ has maximum weight, we know that it can be obtained via Prim's algorithm, and that there exists a sequence $\mathcal{T}_1, \ldots, \mathcal{T}_n = \mathcal{T}$ of trees constructed by this algorithm. Let $\mathcal{T}_k = (\mathcal{C}^*, \mathbb{E}_k)$.
We proceed by contradiction. Let k be the largest index such that $\mathcal{T}_k$ can be extended to a junction tree for $\mathcal{C}_G^*$, and let $\mathcal{T}'$ be a junction tree extension of $\mathcal{T}_k$. Assume that $k < n$, and let $e_{k+1} = (C_{k+1}, C')$ be the edge that has been added when building $\mathcal{T}_{k+1}$, with $\mathcal{C}_{k+1} = \{C_{k+1}\} \cup \mathcal{C}_k$. This edge is not in $\mathcal{T}'$, and there therefore exists a unique edge $e = (B, B')$ in the path between C_k and C' in $\mathcal{T}'$ such that $B \in \mathcal{C}_k$ and $B' \notin \mathcal{C}_k$. We must have $\omega(e) = |B \cap B'| \leq \omega(e_{k+1}) = |C_{k+1} \cap C'|$. But, since the running intersection property is true for $\mathcal{T}'$, both B and B' must contain $C_{k+1} \cap C'$ so that $B \cap B' = C_{k+1} \cap C'$. This implies that, if one modifies $\mathcal{T}'$ by replacing edge e by edge e_{k+1}, yielding a new spanning tree $\mathcal{T}''$, the running intersection property is still satisfied in $\mathcal{T}'$. Indeed if a vertex $s \in V$ belongs to both extremities of a path containing B and B' in $\mathcal{T}'$, then it must belong to $B \cap B'$, and hence to $C_{k+1} \cap C'$, and therefore to any set in the path in $\mathcal{T}'$ that linked C_{k+1} and C'. So we found a junction tree extension of Tk+1, which contradicts our assumption that k was the largest. So we must have $k = n$ and $\mathcal{T}$ is a junction tree.
Let us now consider the converse statement and assume that $\mathcal{T}$ is a junction tree. Let k be the largest integer such that there exists a sequence of subgraphs of $\mathcal{T}$ that is provided by Prim's algorithm. Denote such a sequence by $(\mathcal{T}_1, \ldots, \mathcal{T}_k)$, with $\mathcal{T}_j = (\mathcal{C}_j, \mathbb{E}_j)$. Assume (to get a contradiction) that $k < n$, and consider a new step for Prim's algorithm, adding a new edge $e_{k+1} = \{C_{k+1}, C'\}$ to $\mathcal{T}_k$. Take as before the path in $\mathcal{T}$ linking C' to C_{k+1} in $\mathcal{T}$, and select the edge e at which this path leaves $\mathcal{C}_k$. If $e = (B, B')$, we must have $\omega(e) = |B \cap B'| \leq \omega(e_k) = |C_{k+1} \cap C'|$, and the running intersection property in $\mathcal{T}$ implies that $C_{k+1} \cap C' \subset B \cap B'$, which implies that $\omega(e) = \omega(e_{k+1})$. This implies that adding e instead of e_{k+1} at step $k+1$ is a valid choice for Prim's algorithm, and contradicts the fact that k was the largest number of such steps that could provide a subtree of $\mathcal{T}$. So $k = n$ and $\mathcal{T}$ is maximal. □

3. [2 points] Describe how you produced the cliquetree file.

In order to get the clique tree structure, we implement the five steps of forming a clique tree from any graph shown in this time homework. We should point out that generating a clique tree by using these five steps is a NP-hard problem. Even one more edge can increase the runtime dramatically. Since the whole graph is a like an iteration structure, we can use part of the whole graph, then expand it to the whole graph.

```
./calculate-cliquetree   networkStructureFile(optional)
```

I create a file called "simple-clique-network-grid10x10-t10.txt" to store a part of the whole graph. The $network - structure - file$ is a optional file. Running "calculate-cliquetree" can calculate the corresponding clique tree of this network structure. The result clique tree is identical to the clique tree structure shown in control file "cliquetree-grid10x10-t10.txt". It will run for about 20s to calculate the clique-tree structure.

```
ChenTianyis-MacBook-Pro:RobotLocalization chentianyi$ ./calculate-cliquetree
simple-network-grid10x10-t10.txt

clique tree structure

clique ('PositionRow_t', 'PositionCol_t', 'PositionRow_t+1', 'Action_t')
clique ('PositionCol_t', 'PositionRow_t+1', 'PositionCol_t+1', 'Action_t')
edge weight 3

clique ('PositionRow_t', 'PositionCol_t', 'PositionRow_t+1', 'Action_t')
clique ('PositionRow_t', 'PositionCol_t', 'ObserveWall_t')
edge weight 2

clique ('PositionRow_t', 'PositionCol_t', 'PositionRow_t+1', 'Action_t')
clique ('PositionRow_t', 'PositionCol_t', 'ObserveLandmark1_t')
edge weight 2

clique ('PositionRow_t', 'PositionCol_t', 'PositionRow_t+1', 'Action_t')
clique ('PositionRow_t', 'PositionCol_t', 'ObserveLandmark2_t')
edge weight 2
[Finished in 16.6s]
```

After verifying the clique tree structure, we write a python script to generate the cliquetree files of different parameters, called "generate-clique-tree.py".

```
./generate-clique-tree-file network-gridYYxYY-tmmm.txt cliquetree-gridYYxYY-tmmm.txt
```

We first generate a test cliquetreefile under 10x10, t=10, and compare it with 'cliquetree-grid10x10-t10.txt'.

```
ChenTianyis-MacBook-Pro:RobotLocalization chentianyi$ ./generate-clique-tree-file network-gr
id10x10-t10.txt test-cliquetree-grid10x10-t10.txt
ChenTianyis-MacBook-Pro:RobotLocalization chentianyi$ cmp test-cliquetree-grid10x10-t10.txt
cliquetree-grid10x10-t10.txt
cmp: EOF on cliquetree-grid10x10-t10.txt
```

As the result shown, our output clique tree file is identical to the original one provided by the instructors which verifies the correctness of our code. Then we can use it to generate the clique tree file with different parameters.

4. [4 points] For a grid of size N by M and a sequence of T actions and observations (assume they are fully observed), what would be the computational complexity of computing the distribution over the final position at time T if we simply marginalized over a joint CPT? What is thee computational complexity of computing the distribution over the final position at time T using message passing in your clique tree?

If we simply marginalized over a joint CPT, that is to say we need to compute $P(PositionRow\_T)$ and $P(PositionCol\_T)$ by our condition, we have the joint probability of all variables as:

$$P(PR\_0, PC\_0, AC\_0, \ldots, PR\_T-1, PC\_T-1, AC\_T-1, PR\_T)$$

Then the complexity is: $O(Simply\ marginal) = O((MN)^T)$.
If using message passing in your clique tree, by our construction $PositionRow\_t$ belongs to some clique as $C_i = PositionRow\_t, PositionRow\_t-1, Action\_t-1$ which is connected with clique $C_j = PositionCol\_t-1, PositionRow\_t-1, Action\_t-1$, since running intersection property implies $S_{i,j} = \{PositionRow\_t-1, Action\_t-1\}$, then we have:

$$P(PR\_t = j_t) = \sum_{AC\_t-1, PR\_t-1} P(PR\_t = j_t | AC\_t-1, PR\_t-1)\delta_{i\to j}(AC\_t-1, PR\_t-1)$$

and $1 \le j_t \le N$, then we have computational complexity is: $4N^2$. Since we have T steps in our clique tree, then we have computational complexity is : $4N^2T$. So as computational complexity of $P(PC_t)$, it is $4M^2T$. Therefore, the complexity of computing final position is $O(\max(M^2, N^2)T)$

5. Suppose we wanted to query for the robot state at time 5 and time 15.
(a) [7 points] How would you modify the clique tree so that you could make this query? Prove that your modification is guaranteed to give the correct marginal probability. Hint: to ask a query, there must exist a cluster which contains all of the query variables.
(b) [3 points] Why is it not always valid to answer this type of query using two different clusters where each contains one of the query variables?

(a) Modification Consider the query for the robot state at time 5 and at time 15. We have the variable related to these should be $Position_Row\_5$, $Position_Col\_5$, $Action\_4$ and $Position_Row\_15$, $Position_Col\_15$, $Action\_14$. Then construct a clique tree with these two nodes, then pick one of them to be root, then do a belief update, since clique with $t = 5$ is ready if and only if all its neighbor are ready. Therefore, we could get root for passing the message forward and backward, then we get the query of $t = 5$ and $t = 15$.
Proof
By probabilistic Graphical Model by Koller, we have a clique satisfies the family preservation and Running intersection property. Then we could find a path in our clique tree that contains all the variables only in this path. By variable eliminate algorithm, we could get a clique tree, so that the query variables must exists in the same cluster graph that contains all the query variables.
(b) By calibration tree, we need to check whether

$$\sum_{C_i - S_{i,j}} \mu_{i,j} = \sum_{C_j - S_{i,j}} \mu_{i,j}$$

holds or not.
If a clique tree is not calibrated, once different clusters contains each query variables, they may send message to their neighbors and update belief in different ways.The belief update will be different which cause noise to our probability calculation. Hence,the answer to this query will not be valid.

4.2.4 Deliverables: Message Passing [60 points]

We implement belief update inference based on chapter 10.3 Koller book. A $bayes - query - bp$ is in RobotLocalization File.

The following is some details of our implemented algorithm:
Variables:
1. C_i , a clique in the clique tree.
2. β_i, belief of the clique C_i.

Since the clique tree has not cycles, one iteration of belief propagation should be able to calibrate the entire tree.
In this algorithm, we took advantage of the tree structure, and apply a recursively message passing function.

Algorithm 1 Calibration using belief propagation

```
procedure CALIBRATION
    Read Data from files
    Build clique tree CT
    Initialize belief β
    Initialize message μ = 1
    root ← any clique in CT
Run 2 iterations.
    BeliefPropagation(root, Null)
    BeliefPropagation(root, Null)
```

While the clique tree may have no directions, for two adjacent cliques, we define the clique which is next to *root* to be the father of the other.
BeliefPropagation is the procedure that takes as input two adjacent cliques, C_i and C_j, where C_j is the father of C_i. Let $S_{i,j}$ be the intersection of C_i and C_j, BeliefPropagation passes messages between the two cliques to make sure that

$$\sum_{C_i - S_{i,j}} \beta_i = \mu_{i,j} = \sum_{C_j - S_{i,j}} \beta_j$$

Since we use recursive definition we are allowed to pass the message downstream and upstream in the same procedure.

For query, if there are nodes that are not in the same clique then we need to apply the chain rule. In the code, the RecQuery is a recursive function that finds the probability of X, and which root we start does not matter.

Algorithm 2 BeliefPropagation

procedure BELIEFPROPAGATION(C_i, C_j)
  $S_{i,j} \leftarrow C_i \cap C_j$
  $\sigma_{i,j} \leftarrow \mathbf{0}$
  `// Downstream`
  for all $b \in \beta_j$ **do**
    $\sigma_{i,j}[b \cap S_{i,j}] \leftarrow \sigma_{i,j}[b \cap S_{i,j}] + \beta_j[b]$
  for all $b \in \beta_i$ **do**
    $\beta_i[b] \leftarrow \beta_i[b] \times \frac{\sigma_{i,j}[b \cap S_{i,j}]}{\mu_{i,j}[b \cap S_{i,j}]}$
  $\mu_{i,j} \leftarrow \sigma_{i,j}$
  `// For all children`
  for all $C_{nb} \in Neighbor(C_i)$ **do**
    if $C_{nb} != C_j$ **then**
      BeliefPropagation(C_{nb}, C_i)
  `// Upstream`
  for all $b \in \beta_i$ **do**
    $\sigma_{i,j}[b \cap S_{i,j}] \leftarrow \sigma_{i,j}[b \cap S_{i,j}] + \beta_i[b]$
  for all $b \in \beta_j$ **do**
    $\beta_j[b] \leftarrow \beta_j[b] \times \frac{\sigma_{i,j}[b \cap S_{i,j}]}{\mu_{i,j}[b \cap S_{i,j}]}$
  $\mu_{i,j} \leftarrow \sigma_{i,j}$

Algorithm 3 Query

procedure QUERY
  for all X, Y **do**
    Initialize belief β
    Initialize message $\mu = \mathbf{1}$
    update β using Y
    Calibration()
    $root \leftarrow$ any clique in CT
    $result$ = RecQuery(root, X)

4.2.5 Extra Credit: Max-product Message Passing [20 points]

Max-product belief update message passing algorithm is analogous to belief update os sum-product message passing, the max-product message passing update beliefs of cliques by choosing the maximal marginal probability.

$$\beta_i(c_i) = MaxMarg_{\widetilde{p}_\phi}(c_i)$$

The sepset belief will choose the maximal belief, in particular when

$$max_{c_i - s_{i,j}} \beta_i = max_{c_j - s_{i,j}} \beta_j = \mu_{i,j}(S_{i,j})$$

Another different is the calculation of message, in max-product,the message $\sigma_{i \to j}$

$$\sigma_{i \to j} \leftarrow max_{c_i - s_{i,j}} \beta_i$$

The rest things are not changed comparing with the algorithm in Chapter 10.3

We implement the max-product algorithm based on these changes, a bayes-query-mp is offered in the Robot-Localization File.

But the results seems a little wierd. We will continue to debug it, and update it in our github.