# Implementation of Clique Tree Algorithm for Conditional Linear Gaussian Model

Technical Report

Wei Sun

December 2009

## ABSTRACT

This report presents in details an implementation of Clique Tree algorithm, also known as Junction Tree algorithm, for one particular type of hybrid Bayesian network called Conditional Linear Gaussian (CLG). From constructing a clique tree from the original Bayesian network, to finding a strong root to make sure the correct message propagation flow, to message passing protocal between cliques, to finally computing the marginal posterior distributions for hidden variables, I thoroughly describe almost all of details of procedures involved. I aim to making this report a self-complete document, by which programmer should be able to implement Clique Tree algorithm for CLG into automatic inference engine. In this report, I also summarize and discuss some important theoretical and implementation issues including strong/weak marginalization for conditional Gaussian (CG) potential, complexity of Clique Tree algorithm with clique size, and complexity comparision with other inference algorithms.

**Keywords:** Bayesian networks, Clique Tree algorithm, Conditional Linear Gaussian.

## 1. INTRODUCTION

Bayesian network (BN) [1] [12] [5] [11] is a directed acyclic graph (DAG) consisting of nodes and arrows (directed edges). Node represents random variable and arrow represents the casual dependence relationship between nodes. Every node in BN has a pre-defined conditional probability distribution (CPD) for the random variable it represents. CPD is a functional relationship describing the probabilistic dependence between the node and all of its parent nodes. BN with both discrete and continuous random variable is called hybrid model. The simplest hybrid BN model is Conditional Linear Gaussian (CLG) [14] [8], in which discrete node could have Gaussian child, but no continuous parent node for any discrete variable.

An important task with BN model is to do probabilistic inference for decision support. Over the last several decades, a number of inference algorithms for Bayeisan networks has been developed [4]. Among them, Clique Tree algorithm, also known as Junction tree, [10] [7] [9], is one of most popular algorithms to comput exact posterior distributions for discrete random variables or exact first two moments for continuous Gaussian variables for particular types of Bayesian networks, namely, pure discrete Bayesian networks or CLG. Unfortunately in the literature, there is not much detailed descriptions about Clique Tree algorithm from the implementation point of view. Popular commercial software in the market having Clique Tree algorithm implemented include Hugin (Http://www.hugin.com/), Netica (http://www.norsys.com/), etc. Hugin has the capability to handle both pure discrete model and CLG, while Netica always discretizes continuous variables. In addition, there are some BN research tools implementing Clique Tree algorithm such as Matlab BN Toolbox (BNT), Genie Smile (http://genie.sis.pitt.edu/), etc.

This report aims to becoming a self-complete technical documentation to help both researcher and software developer to understand Clique Tree algorithm thoroughly. It is organized as follows. Section 1.1 summarizes all of notations I used in this report to make the reader easy to understand. To illustrate the algorithm, I introduce an example CLG model in Section 1.2. Section 1.3 outlines the framework and general mechanism of Clique Tree algorithm. Section 2 focuses on describing how to construct a clique tree from an original Bayesian network, and how to find a strong root clique from the clique tree. Several topics related to graph transformation will be presented such as moralization, triangulation, building maximum spanning tree, etc. Explanation of why a strong elimination order is needed, how it related to strong marginalization in theory, will also be discussed in this section. In section 3, I present all of computations in details for message passing between cliques in order to compute eventually the marginal distributions for hidden variables. Topics include conditional Gaussian (CG) potential representation in canonical form and moment form, potential initialization for nodes and cliques, basic operations of CG potentials such as multiplication, division, marginalization, message propagation protocol between cliques, calibration of clique tree, etc. Finally in section 4, I take the chance to discuss both implementation and theoretical issues for Clique Tree algorithm, including the algorithm complexity.

## 1.1. Notation

The notations used in this report follow conventions in the literature. In general, I use capital letters such as $A, B, X$ to denote random variables or nodes in Bayesian networks. Bold capital letters (e.g., $\mathbf{A}, \mathbf{X}, \mathbf{Y}$) are used to denote set of random variables or set of nodes. The corresponding lower case letters such as $a, b, x$ and $\mathbf{a}, \mathbf{x}, \mathbf{y}$ are representing the particular instantiations of the random variables or set of random variables. Some variable may have subscript index such as $A_i, C_j$. Generally, clique is a set of nodes, and so it is denoted by bold letter. Union of two sets $\mathbf{C_i}$ and $\mathbf{C_j}$ is denoted as $\mathbf{C_i} \cup \mathbf{C_j}$. Set difference, e.g., set of variables in $\mathbf{C_i}$ but not in $\mathbf{C_j}$, is denoted as $\mathbf{C_i} \backslash \mathbf{C_j}$. Set intersection, common variables between two sets $\mathbf{C_i}, \mathbf{C_j}$, is denoted as $\mathbf{C_i} \cap \mathbf{C_j}$.

Vector and function variable are usually denoted as lower case bold letters. Matrix is denoted as capital bold letter. Transform of vector or matrix will be represented using $T$ in the superscript position, such as $\mathbf{h}^T, \mathbf{K}^T$. $P(A = A_i)$ denotes the probability of discrete random variable $A$ taking one of its states $A_i$. $p(x)$ denotes the probability density function of a continuous random variable $X = x$. Other than random variable that usually represented by capital letter, general scalar variable will be denoted as lower case letter. A scalar Gaussian variable $X$ with mean $u$, and variance $\sigma^2$, is denoted as $X \sim \mathcal{N}(u, \sigma^2)$. A multi-variate Gaussian variable $\mathbf{X}$ with mean vector $\mathbf{u}$, and covariance matrix $\mathbf{\Sigma}$ is denoted as $\mathbf{X} \sim \mathcal{N}(\mathbf{u}, \mathbf{\Sigma})$. A discrete random variable with probability mass function for all of its possible states is denoted by a special stacked symbol $\overset{d}{\sim}$ meaning distributed discretely, followed by a list of probabilities, each number for one state. For example, a binary discrete random variable $B$ with the probabilities being in state 1, and state 2 as $0.3, 0.7$, is denoted as $B \overset{d}{\sim} D(1 : 0.3\,; 2 : 0.7)$. A conditional probability distribution of random variable $A$ given $B$ is denoted as $P(A|B)$. Also, I usually use low case letter to denote function variable, and $f, v, \phi$, etc. to denote functions.

## 1.2. Example BN model for illustration

I use a 8-node poly tree CLG model called Poly8CLG as an illustration example. Figure 1 shows its structure. As you can see, Poly8CLG consists of 3 discrete nodes $A, B, C$, depicted by rectangles, and 5 continuous nodes $U, X, Y, W, Z$, depicted by ellipses.

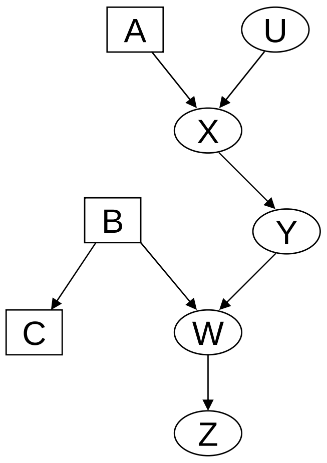


**Figure 1.** Poly8CLG: a 8-node poly tree CLG Bayesian network.

The CPDs defined in Poly8CLG are listed as below:

$$\begin{aligned}
P(A) &\overset{d}{\sim} D(1:0.8\,;2:0.2)\\
P(B) &\overset{d}{\sim} D(1:0.5\,;2:0.5)\\
P(C|B=1) &\overset{d}{\sim} D(1:0.5\,;2:0.5)\\
P(C|B=2) &\overset{d}{\sim} D(1:0.3\,;2:0.7)\\
P(U) &\sim \mathcal{N}(30,\ 2)\\
P(X|A=1,U) &\sim \mathcal{N}(-5+U,\ 1)\\
P(X|A=2,U) &\sim \mathcal{N}(5+U,\ 1)\\
P(Y|X) &\sim \mathcal{N}(-2X,\ 1)\\
P(W|B=1,Y) &\sim \mathcal{N}(5+Y, 1)\\
P(W|B=2,Y) &\sim \mathcal{N}(20+2Y, 2)\\
P(Z) &\sim \mathcal{N}(0.5W,\ 1)
\end{aligned}$$

I will refer to this example model for concrete illustrations when I describe the algorithm details in the following sections.

## 1.3. Outline of Clique Tree algorithm

Basically, Clique Tree algorithm consists of two processes: first one is a series of graph transformations to build a clique tree from the original Bayeisan network represented in a directed acyclic graph (DAG); another process is the computation of message passing in the clique tree. Each process includes lots of detailed procedures. Graph transformations include moralization, triangulation, clique identification, finding strong root clique, building directed clique tree, etc. Message passing relates to message representation as potential in canonical form or moment form, potential marginalization, multiplication, and division, etc. Figure 2 shows a basic framework of Clique Tree algorithm in the order of procedures.

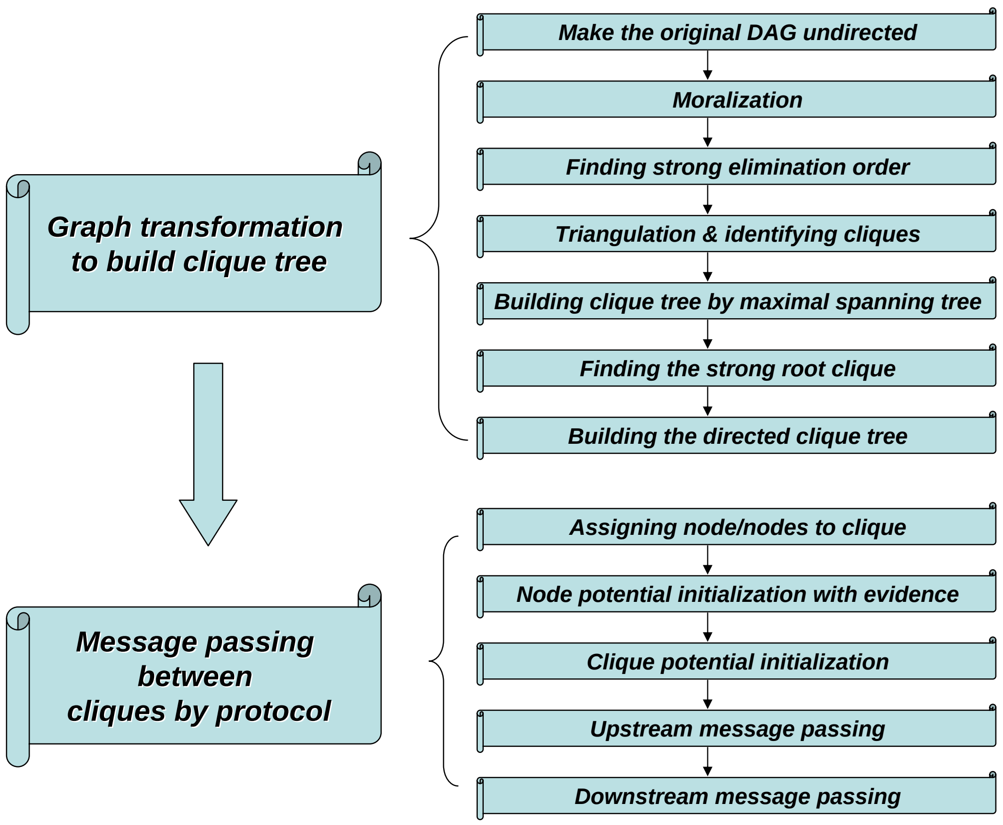

Graph transformation to build clique tree
Make the original DAG undirected
Moralization
Finding strong elimination order
Triangulation & identifying cliques
Building clique tree by maximal spanning tree
Finding the strong root clique
Building the directed clique tree
Message passing between cliques by protocol
Assigning node/nodes to clique
Node potential initialization with evidence
Clique potential initialization
Upstream message passing
Downstream message passing

**Figure 2.** Outline of Clique Tree Algorithm

Let me briefly review the procedures listed in the figure, detailed explanations will be presented in later sections. The first step is just to remove all of directions for arcs in the original DAG, and so make the graph undirected. Moralization means that marrying parents of common child node, namely, have a link connecting every pair of parent nodes. Elimination order is all about the order of variables being summing/integrating out when computing posterior distributions. For CLG, we need a strong elimination order to make sure strong marginalization, and further guarantee the correct first two moments for hidden Gaussian variables. Conceptually, strong elimination order always has continuous node eliminated prior to discrete node, and node in the moralized graph that requiring less number of fill-in edges for all of its neighboring nodes has the priority to be eliminated first. Triangulation is to make any loop in the moralized graph has no more than 3 nodes/vertices. Equivalently, for any node in the moralized graph, triangulation is making all of its neighboring nodes connected each other if they are not connected yet. By definition, clique is a maximal complete set of nodes connected each other. The set of a node and all of its neighboring nodes forms a potential clique. And a potential clique that is not a subset of any other potential cliques is a real clique. Based on the strong elimination order, triangulation make it easy to identify all of real cliques. And then, next is to build a clique tree using maximal spanning tree algorithm. Finding a strong root clique after constructing a clique tree is especially important in implementing Clique Tree algorithm for CLG, because it is the key to guarantee the correct message passing order, and so to make sure the correct first two moments of Gaussian variables. The final step in graph transformation is to add directions in the clique tree by letting the strong root clique pointing out to its neighboring cliques, and then these neighboring nodes further point out to its other neighboring cliques that one step further away from the strong root clique, repeating until reaching the leave cliques.

The computatation part of Clique Tree algorithm starts with assigning node/nodes to clique. Note that I use node to mean variable in the original Bayesian network, but clique to mean the clique in the clique tree, which is a set of nodes. I would like to remind the reader not being confused by them. Potential functions are used to represent CPDs for nodes, and further, to represent messages for cliques, in either canonical form or moment form, or both. After we initialize potentials for all of nodes, clique potential is initialized as the product of potentials of nodes that assigned to the clique. Then, based on message passing protocol for sending message from one clique to its neighboring clique (presented in Section 3.3), we need to do upstream message passing first, then followed by downstream message passing. Upstream message passing is the message sending process from leave cliques to the strong root clique. On the opposite, downstream message passing is to send message back to leave cliques from the strong root clique. After these two iterations of message flow, all of cliques have the correct joint posterior distributions. Marginal distribution for single variable could be computed by further marginalizing the clique. During the message passing process, we need to do a few potential operations, such as potential marginalization, multiplication, and division. Among all of potential operations, marginalization is the most complicated one, especially when marginalizing over both continuous and discrete variables. Detailed descriptions for all of potential operations will be presented in Section 3.2.

## 2. CONSTRUCTING CLIQUE TREE FROM BAYESIAN NETWORK

From the original Bayesian network, we first need to construct a clique tree using a series of graph transformations. Clique is set of nodes chosen from the original network. Clique tree, also known as Junction tree, are single-connected tree of cliques. The most important property of clique tree is the running intersection property. - any clique in the path of two arbitrary cliques $\mathbf{C_1}, \mathbf{C_2}$ in clique tree, contains the common variables in $\mathbf{C_1}, \mathbf{C_2}$. Graph transfromations include moralization, triangulation, etc. A strong elimination order of nodes from original network is very critical in the process of building the clique tree for CLG. Unfortunately, finding the optimal strong elimination order is NP-hard in general. Usually, clique tree is not unique, resulting in different computational efficiency. For CLG, clique tree must have a strong root for message passing to ensure exact posterior distributions.

The very first step for graph transformation is to remove directions for all of arcs between nodes in the original network, and so make the original directed acyclic graph undirected. Next, we will need to do moralization, triangulation, and decomposition, etc.

### 2.1. Moralization

Moralization is the procedure to marry parent nodes — have an edge between every pair of parents for a common child. For examples, we need to link $A, U$, and $B, Y$ to moralize Poly8CLG. The moralized undirected graph of Poly8CLG is shown in Figure 3.

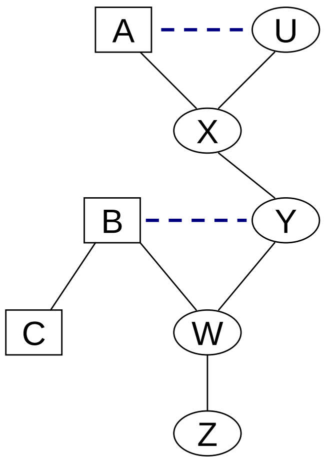


**Figure 3.** Moralized undirected graph of Poly8CLG: dark blue dashed lines show the moralization edges added between $A, U$ and $B, Y$.

## 2.2. Strong elimination order, triangulation and identifying cliques

Here the word 'strong' may sound wierd to the reader at the first glance. It is actually originated from the meaning of exact marginalization without any approximation, called strong marginalization. On the opposite, weak marginalization involves collapsing Gaussian mixture into less number of components, and so approximating distribution with the first two moments of Gaussian. In this sense, it is called 'weak' or 'strong'. Furthermore, in order to ensure the strong marginalization, the corresponding elimination order, triangulation, and the root clique for correct message flow, even the clique tree constructed consequently, are also named with this descriptive adjective 'strong'. All of these 'strong' things are for hybrid model only because of the heterogeneity of variable types in the network.

To better explain, let me present some definitions first:

- Cluster — a set of nodes, such that all nodes in the set are connected each other. Cluster is also called a complete set of nodes.
- Clique — a set of nodes, such that all nodes in the set are connected each other, and there is no other node in the original network is connected to all nodes of this set. In short, clique is the maximal complete set of nodes. Being complete is in the sense of all of its element are connected each other; being maximal is in the sense of fact that no any other node is connected to all its elements.

- Potential clique — a candidate clique consisting of a node and all of its neighboring nodes, of which at least one pair of neighboring nodes is not connected each other. A potential clique becomes a real clique if we add edges between unconnected neighboring nodes.

- Size of clique — product of sizes of nodes in this clique. Size of discrete variable is the number of states it can have; size of continuous variable is the dimensionality of this variable.

- Separator — set of common nodes/variables between two neighboring cliques.

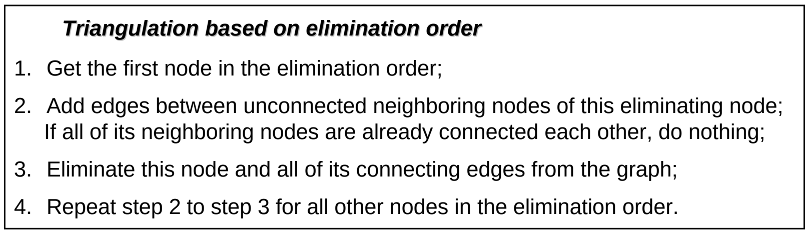
- Triangulation — a process to make a graph triangulated, namely, adding chord to break any cycle of loop that has more than 3 nodes/vertices.

- Discrete parent — discrete node that has at least one continuous child.

- Strong triangulation (for CLG only) — a process to (1) add edges between unconnected discrete parent nodes; (2) add chord to make the graph triangulated.

By definition, a cluster is a complete, but not necessarily maximal, set of nodes. And a clique is a maximal cluster. We know that once a graph is triangulated, cliques are determined and could be induced from the graph by identifying all of maximal clusters.

Triangulation could be done arbitrarily, as long as chord is added to make a cycle of loop in the graph no more than 3 nodes/vertices. But essentially, it is the elimination order of nodes to determine the way of how to triangulate a graph. It is achieved by Triangulation Procedure in Figure 4:



***Triangulation based on elimination order***

1. Get the first node in the elimination order;
2. Add edges between unconnected neighboring nodes of this eliminating node;
   If all of its neighboring nodes are already connected each other, do nothing;
3. Eliminate this node and all of its connecting edges from the graph;
4. Repeat step 2 to step 3 for all other nodes in the elimination order.

**Figure 4.** Triangulation Procedure based on an elimination order of nodes.

Different elimination orders result in different triangulation, and so different computational efficiency. Note that finding the optimal elimination order itself is a NP-hard problem.

In this report, I have no intention to find the optimal elimination order. Instead, the correctness of implementing the algorithm is more important in our consideration.

For CLG, we need a strong elimination order to achieve strong triangulation. And it is the key to maintain the correct computation of joint distribution during message passing. The genenral rules for finding a good strong elimination order are:

1. Continuous variables must be eliminated before discrete variables.
2. Looking at the undirected moralized graph, node that requires less number of fill-in edges for its neighbors, has the priority to be eliminated;
3. Node, that has smaller size of potential clique or real clique associate with, has the priority to be eliminated;
4. Node that has lower index in the data structure is eliminated first if there are multiple candidate nodes satisfying the above conditions.

The first rule shown above is simply sufficient to guarantee the adding edges between unconnected discrete parent nodes in the graph. And then, it ensure the strong triangulation. Other rules help to improve computational efficiency.

Applied to our example model Poly8CLG, suppose that we index nodes in the data structure as $A - B - C - U - X - Y - W - Z$, then a strong elimination order could be $Z - U - W - X - Y - C - A - B$. It will be a very good exercise for the reader to find why this order is strong. Correspondingly, the strongly triangulated graph of Poly8CLG is shown in Figure 5.

It is straightforward then to find all of cliques from the strongly triangulated graph. A pseudo code is present in Figure 6 for implementor to program the procedure to find all cliques from a moralized and strongly triangulated graph. For our example network Poly8CLG, based on the strong elimination order mentioned earlier, all of cliques identified are $(W, Z), (A, U, X), (B, Y, W), (A, X, Y), (A, B, Y), (B, C)$.

## 2.3. Building clique tree using maximal spanning tree

Now it is ready to give the formal definition of Clique Tree:

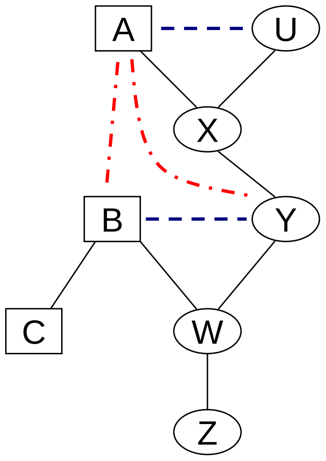


**Figure 5.** Moralized and strongly triangulated graph of Poly8CLG: dark blue dashed lines show the moralization edges added between $A, U$ and $B, Y$; red dashed lines show the strong triangulation edges added between $A, B$ and $A, Y$, of which the edge $A - B$ is to connect the unconnected discrete parents $A$ and $B$.

```
    Identifying cliques from moralized and triangulated graph
For nodes in elimination order,
   1. look at the moralized and triangulated graph G.
   2. identifying clique as the set of the node itself and all of its neighboring nodes.
   3. if clique identified in step 1 is not a subset of any previously induced clique,
      then save it as a clique; otherwise, discard it.
   4. eliminate the node and all edges to it from the graph G.
   5. save the remaining graph as G.
End-For
```

**Figure 6.** Pseudo-code to find all cliques from the moralized and triangulated graph.

Definition 2.1 (Clique Tree). *A Clique Tree over a Bayesian network $\mathcal{B}$ with random variables $\mathbf{X}$ is an undirected and singly-connected graph $\mathcal{T} = \{\mathcal{V}, \mathcal{E}\}$, of which $\mathcal{V}$ is a set of cliques, and $\mathcal{E}$ is a set of edges between cliques. Any clique $\mathbf{C}_i \in \mathcal{V}$ is a subset $\mathbf{X_i} \subset \mathbf{X}$. Any edge $E_i \in \mathcal{E}$ is associated with a nonempty set of common variables between two cliques connected by the edge $E_i$. Clique Tree obeys two properties:*

- *For any CPD in $\mathcal{B}$ over the variables $\mathbf{Y} \subset \mathbf{X}$, there exists an unique clique $\mathbf{C}$ such that $\mathbf{Y} \subset \mathbf{C}$.*
- *Let $\mathbf{C}_i, \mathbf{C}_j$ be two arbitrary cliques such that their intersection $\mathbf{M} = \mathbf{C}_i \cap \mathbf{C}_j$, then any clique $\mathbf{C}_k$ in the path of $\mathbf{C}_i$ to $\mathbf{C}_j$ has $\mathbf{M}$ in its domain, namely, $\mathbf{M} \subset \mathbf{C}_k$. This property is called the running intersection property.*

Usually there are a number of ways to connect cliques to be a clique tree, and so clique tree is not necessarily to be unique. In our implementaion, we use maximal spanning tree algorithm to construct the clique tree satisfying the running intersection property.

Obviously, a clique connects to another clique only if these two cliques have common variables. We call the number of common variables in the intersection set as the weight of the connecting edge. [6] presented an algorithm called maximal spanning tree algorithm to build the optimal clique tree. It maximizes weights of edges, while minimizing the sum of sizes of two connected cliques. We use a name called edge triple ($\mathcal{ET}$) to represent the association between edge and cliques. $\mathcal{ET}$ always has three items as the two connecting cliques $\mathbf{C}_i, \mathbf{C}_j$, the edge in between $\mathcal{E}_{ij}$, and two parameters as weight of edge $W_{ij}$, sum of sizes of cliques $S_{ij}$, denoted as $\mathcal{ET} = \{\mathbf{C}_i, \mathbf{C}_i, \mathcal{E}_{ij};\ W_{ij}, S_{ij}\}$.

The algorithm starts with an arbitrary chosen clique, then take it as a partial tree. For each of the remaining cliques, if it has a non-empty intersection set with the current clique in the partial tree, the algorithm calculate its edge triple. Among all of candidate edge triples, then choose the one maximizing the weight of edge connecting to the current partial tree, and if there is a tie, choose the one with minimum sum of sizes. After that, remove the chosen clique from set of candidate edge triples and take the chosen clique as the current clique. Repeat the process till all cliqus have been added to the tree. See Table 1 for the pseudo code.

Refer to our example model Poly8CLG, its clique tree is illustrated in Figure 7. By looking at a so-called cluster graph, it is straightforward to verify the clique tree and very helpful to demonstrate the algorithm. As shown in Figure 8, cluster graph shows all of possible connections between cliques. Furthermore, intersection set of connecting cliques and sum of clique sizes are shown along the edges in blue and red colors respectively. A step by step constructing process is demonstrated as the following, :

1. Arbitrarily choose $(A, X, Y)$ as the starting clique;

2. Candidate cliques are $(B, Y, W), (A, B, Y)$, and $(A, U, X)$;

3. By comparing their edge triples, clique $(A, U, X)$ is chosen to be added;

4. Remove edge triple $(A, X, Y) - (A, U, X)$;

**Table 1.** Pseudo code for building the optimal clique tree using maximal spanning tree algorithm.

```
$\mathcal{V}$ = set of all cliques;
$\mathcal{U} = \emptyset$;
arbitrarily choose $\mathbf{C}_0 \in \mathcal{V}$;
$\mathcal{U} = \mathcal{U} + \mathbf{C}_0$;
$\mathcal{R} = \mathcal{V} \backslash \mathcal{U}$;        % set of remaining cliques
$\mathcal{PT} = \mathbf{C}_0$          % partial clique tree
$\mathcal{CT} = \emptyset$             % set of candidate edge triples
while $\mathcal{R} \neq \emptyset$
    for $\mathbf{C}_i$ in $\mathcal{R}$
        if $\mathbf{C}_0 \cap \mathbf{C}_i \neq \emptyset$
            $\mathcal{ET}(i) = \{\mathbf{C}_0, \mathbf{C}_i, \mathcal{E}_{i0};\ W_{i0} = length(\mathbf{C}_0 \cap \mathbf{C}_i),\ S_{i0} = sizeof(\mathbf{C}_0) + sizeof(\mathbf{C}_i)\}$
            $\mathcal{CT} = \mathcal{CT} + \mathcal{ET}(i)$
        end-if
    end-for
    $\mathcal{ET}(kc) = arg_W\ max(\mathcal{CT})$          % choose maximum edge weight.
    $\mathcal{ET}(k) = arg_S\ min(\mathcal{ET}(kc))$       % choose the minimum sum of sizes.
    $\mathcal{PT} = \mathcal{PT} + \mathcal{ET}(k)$              % add the chosen edge triple $k$ into the partial tree.
    $\mathcal{CT} = \mathcal{CT} \backslash \mathcal{ET}(k)$         % remove the chosen edge triple $k$ from the candidate set.
    $\mathcal{U} = \mathcal{U} + \mathbf{C}_k$         % add clique $k$ into used set.
    $\mathbf{C}_0 = \mathbf{C}_k$              % make clique $k$ as the current clique.
    $\mathcal{R} = \mathcal{V} \backslash \mathcal{U}$             % update set of remaining cliques.
end-while

return $\mathcal{PT}$ as the clique tree.
```

5. Take clique $(A, U, X)$ as the current clique, and edge triple $(A, U, X) - (A, B, Y)$ is added into candidate set;

6. Edge triple $(A, X, Y) - (A, B, Y)$ is chosen and clique $(A, B, Y)$ is added into the tree;

7. Clique $(A, B, Y)$ is the current clique, and two more edge triples $(A, B, Y) - (B, C)$ and $(A, B, Y) - (B, Y, W)$ are added as candidates;

8. Edge triple $(A, B, Y) - (B, Y, W)$ is chosen and clique $(B, Y, W)$ is added into the tree;

9. Clique $(B, Y, W)$ is the current clique, and two more edge triples $(B, Y, W) - (B, C)$ and $(B, Y, W) - (W, Z)$ are added as candidates;

10. Edge triple $(B, Y, W) - (B, C)$ is chosen and clique $(B, C)$ is added into the tree;

11. Clique $(B, C)$ is the current clique, and no more edge triples is added as candidate;

12. Edge triple $(B, Y, W) - (W, Z)$ is chosen from the remaining candidates and clique $(W, Z)$ is added into the tree;

13. No more clique remaining, return the clique tree.

The interested reader is welcome to try with other starting clique. It will end with the same clique tree using the algorithm.

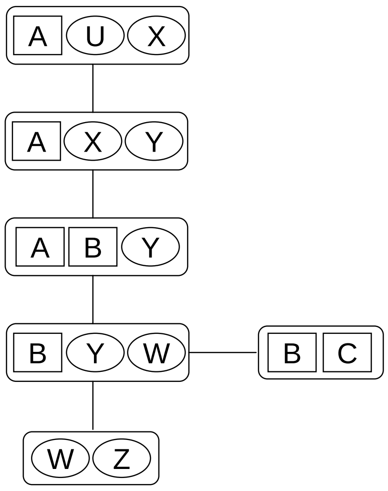


**Figure 7.** The clique tree of Poly8CLG.

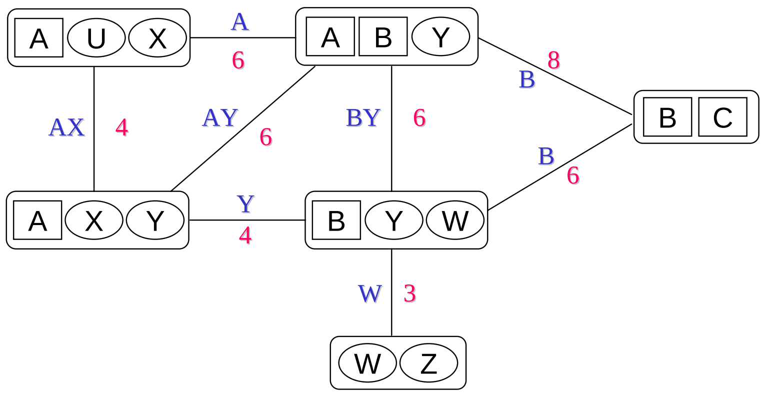


**Figure 8.** The cluster graph of Poly8CLG — blue letters are intersections of possible connecting cliques; and red numbers are the sum of sizes of both cliques connected.

## 2.4. Finding the strong root clique and building the directed clique tree for message passing

For pure Bayesian network, any clique could be a root for message passing. But for hybrid model such as CLG, the root clique has to be 'strong' to ensure strong marginalizations (no approximation) when sending messages towards the root clique from other cliques. This is why it is called strong root.

Basically, a clique represents a joint space. The joint density over the clique space is the products of CPDs of nodes in the clique, with the factors not necessarily being densities. For example, it could be unnormalized. For computational convenience, we use a function called potential to represent this data structure. When a clique $\mathbf{C}_i$ sends message to its neighboring clique $\mathbf{C}_j$, the first thing is to marginalize its potential over the variables only in $\mathbf{C}_i$ but not in $\mathbf{C}_j$, namely, $\mathbf{C}_i \backslash \mathbf{C}_j$, onto the domain of the separator between these two cliques, as $\mathbf{C}_i \cap \mathbf{C}_j$. If clique $\mathbf{C}_i$ has both discrete and continuous variables in the domain, its potential is a mixture of Gaussian potentials. Unless its potential represents a real joint density, marginalizing over any of its discrete variables results in incorrect collapsing of Gaussian mixture, consequently, incorrect computations of posterior distribution. This is why marginalizing over discrete variables for hybrid clique is prohabited when sending messages towards the strong root.

Formally, strong marginalization happens only in one of the following cases:

**Table 2.** Pseudo code for finding a strong root in a clique tree.

```
CLG $\mathcal{B}$ over set of discrete variables $\Delta$ and set of continuous variables $\Gamma$
Clique tree of $\mathcal{B}$: $\mathcal{T} = \{\mathcal{V}$: set of cliques, $\mathcal{E}$: set of edges $\}$
for $\mathcal{E}_{ij}$ in $\mathcal{E}$        % check every edge $\mathcal{E}_{ij}$ connecting $\mathbf{C}_i$ and $\mathbf{C}_j$ in the clique tree.
    if $\mathbf{C}_i \cap \mathbf{C}_j \subseteq \Delta$
        $A(i,j) = 1, A(j,i) = 1$
    else if $\mathbf{C}_i \backslash \mathbf{C}_j \subseteq \Gamma$
        $A(i,j) = 1$
    else if $\mathbf{C}_j \backslash \mathbf{C}_i \subseteq \Gamma$
        $A(j,i) = 1$
    end-if
end-for
strong_ root = clique $\mathcal{V}_k$ such that $sum(A(k,:)) = 0$
```

- Domain variables are pure discrete.
- Marginalize over continuous variables only.

Marginalization that involves any approximation such as collapsing Gaussian mixture into less number of component is called weak marginalization.

To guarantee correct computations, cliques must follow a certain order for passing messages in CLG. We know that in the way sending messages toward the strong root, strong marginalization is a necessity. Therefore, a message sending order from $\mathbf{C}_i$ to $\mathbf{C}_j$ on the way towards the strong root is allowable if and only if

$$\mathbf{C}_i \backslash \mathbf{C}_j \subseteq \Gamma \quad or \quad \mathbf{C}_i \cap \mathbf{C}_j \subseteq \Delta$$

where $\Gamma$ is the set of continuous variables and $\Delta$ is the set of discrete variables in the CLG respectively.

By checking every edge in the clique tree, at least one strong root clique will be identified such that it is not allowed to send message out to any of its neighboring cliques. Table 2 presents a psuedo code for finding the strong root clique in a clique tree.

In later sections, you will see that the strong root clique does send messages back to other cliques. This is done after the strong root receives all of messages sending from other cliques, so it has a correct joint density function. Then even though weak marginalization is involved in the sending process and it collapses the Gaussian mixture into one single Gaussian, it still provides the correct first two moments of Gaussian distribution.

A directed clique tree could be built to guide message passing directions after finding a strong root. Following the graph convention — root is always pointing out other cliques, we then have arrows from root to leaves in the directed clique tree. For our example model Poly8CLG, the strong root clique found is $(A, B, Y)$, and its directed clique tree is illustrated in Figure 9. The first round of message passing will be from leave cliques towards the strong root, called upstream message passing. After the strong root $(A, B, Y)$ receives all messages, it will send message back to all of other cliques for updating their joint distributions, known as downstream message passing. Detailed message passing computations will be described in next section.

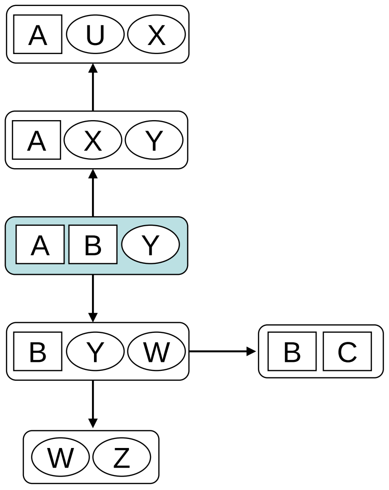


**Figure 9.** Directed clique tree of Poly8CLG — clique $(A, B, Y)$ is the strong root.

# 3. MESSAGE PASSING BETWEEN CLIQUES AND COMPUTING POSTERIOR DISTRIBUTION

## 3.1. Clique Initialization

### 3.1.1. Assigning node to clique

The very first step of calibrating a clique tree, is to assign node to the lightest clique in terms of clique size, that containing the node itself and all of its parents. For the example network Poly8CLG, the following is the list of node association with cliques:

A — (A,U,X) 2

B — (B,Y,W) 3

C — (B,C) 6

U — (A,U,X) 2

X — (A,U,X) 2

Y — (A,X,Y) 4

W — (B,Y,W) 3

Z — (W,Z) 1

where the numbers to the right of cliques are the index of the cliques found in the process of strong triangulation based on the strong elimination order.

### 3.1.2. Computing potential for each node

We represent the basic Gaussian potential for node $\mathbf{x}$ using canonical form as below,

$$\phi(\mathbf{x}) = exp(g + \mathbf{h}^T\mathbf{x} - \frac{1}{2}\mathbf{x}^T\mathbf{K}\mathbf{x})$$

where $g$, $\mathbf{h}$, and $\mathbf{K}$ are canonical characteristics. And in general, $\mathbf{x}$, $\mathbf{h}$, are column vectors; $\mathbf{K}$ is square matrix, $g$ is always a scalar. However, for different type of nodes, these parameters have very different assignments.

- For discrete root node $A$: obviously, its $\mathbf{h}$ and $\mathbf{K}$ will be zeros. And each $g(i)$ will be assigned as value of $log(P(A_i))$ for each of its state, where $P(A_i)$ is the probability of discrete node $A$ being in state $A_i$.

- In CLG, discrete node is limited to have discrete parents only. For this case, similarly, its $g(i)$ will be $log(P(A_i))$ but given its parents. Again, its $\mathbf{h}$ and $\mathbf{K}$ will be zeros.

- For continuous root node $\mathbf{X}$: suppose $X$ is $n$-dimension multi-variate Gaussian $\mathcal{N}(\mathbf{u}, \mathbf{\Sigma})$, we have equations to compute $g$, $\mathbf{h}$, and $\mathbf{K}$ as following,

$$\begin{aligned} g &= -0.5\mathbf{u}^T \Sigma^{-1}\mathbf{u} + log(2\pi)^{(-n/2)} + log(det(\mathbf{\Sigma}))^{(-1/2)} \quad (1) \\ \mathbf{h} &= \mathbf{\Sigma}^{-1}\mathbf{u} \quad (2) \\ \mathbf{K} &= \mathbf{\Sigma}^{-1} \quad (3) \end{aligned}$$

where $det(\mathbf{\Sigma})$ is the matrix determinant of $\mathbf{\Sigma}$.

- For continuous node with continuous parents only: denoted as $\mathbf{X}|\mathbf{Z}$, $\mathbf{X}$ is a linear combination of all of its continuous parents $\mathbf{Z}$. Here $\mathbf{Z}$ is set of all continuous parents variables. Linear coefficients are called weights, denoted as $\mathbf{w}$, and all of weights form the weight vector if $\mathbf{X}$ is a scalar Gaussian variable or weight matrix if $\mathbf{X}$ is a multi-variate Gaussian. Without loss of generality, let us assume $\mathbf{w}$ is a vector, with length equal to number of continuous parents (this is assuming that every continuous variable in the model is scalar Gaussian). So,

$$X = \mathbf{w}^T\mathbf{Z} + \mathcal{N}(u, \sigma^2),$$

where node $X$'s variance $\sigma^2$ does not depend on its parents. We then need to transform the linear Gaussian CPD (conditional probability distribution) into the canonical form. It results that computing $g$ is similar as in equation (1), but $\mathbf{h}$, $\mathbf{K}$ are calculated differently, all listed as below:

$$\begin{aligned} g &= -0.5u^2\sigma^{-2} - 0.5log(2\pi\sigma^{-2}) \quad (4) \\ \mathbf{h} &= \begin{pmatrix} \sigma^{-2}u \\ -\mathbf{w}\sigma^{-2}u \end{pmatrix} \quad (5) \\ \mathbf{K} &= \begin{pmatrix} \sigma^{-2} & -\sigma^{-2}\mathbf{w}^T \\ -\mathbf{w}\sigma^{-2} & \mathbf{w}\sigma^{-2}\mathbf{w}^T \end{pmatrix} \quad (6) \end{aligned}$$

- For continuous node with discrete parents only: denoted as $\mathbf{X}|\mathbf{I}$, $\mathbf{X}$ is a $n$-dimension multi-variate Gaussian $\mathcal{N}(\mathbf{u}_i, \mathbf{\Sigma}_i)$ given each instantiation of discrete parents $\mathbf{I} = i$. Then, given each $\mathbf{I} = i$, $g_i, \mathbf{h}_i$, and $\mathbf{K}_i$ are computed similarly as equation (1), (2), and

(3), shown as below:

$$\begin{aligned} g_i &= -0.5\mathbf{u}_i{}^T\boldsymbol{\Sigma}_i\mathbf{u}_i + log(2\pi)^{(-n/2)} + log(det(\boldsymbol{\Sigma}_\mathbf{i}))^{(-1/2)} & (7) \\ \mathbf{h}_i &= \boldsymbol{\Sigma}_i{}^{-1}\mathbf{u}_i & (8) \\ \mathbf{K}_i &= \boldsymbol{\Sigma}_i^{-1} & (9) \end{aligned}$$

- For continuous node with both discrete and continuous parents: this is a combination of the two cases above, denoted as $\mathbf{X}|\mathbf{I},\mathbf{Z}$. Again, assuming continous variable is scalar Gaussian, then for each realization of its discrete parents $\mathbf{I} = i$, $X$ is a linear combination of its continous parents $\mathbf{Z}$ -

  $$X = \mathbf{w}_i{}^T\mathbf{Z} + \mathcal{N}(u_i, \sigma_i{}^2).$$

  Given each instantiation of its discrete parents $\mathbf{I} = i$, $g_i$, $\mathbf{h}_i$, and $\mathbf{K}_i$ are computed similarly as equation (4), (5), and (6), shown as below:

  $$\begin{aligned} g_i &= -0.5u_i^2\sigma_i{}^{-2} - 0.5log(2\pi\sigma_i^{-2}) & (10) \\ \mathbf{h}_i &= \begin{pmatrix} \sigma_i{}^{-2}u_i \\ -\mathbf{w}_i\sigma_i{}^{-2}u_i \end{pmatrix} & (11) \\ \mathbf{K}_i &= \begin{pmatrix} \sigma_i{}^{-2} & -\sigma_i{}^{-2}\mathbf{w}_i{}^T \\ -\mathbf{w}_i\sigma_i{}^{-2} & \mathbf{w}_i\sigma_i{}^{-2}\mathbf{w}_i{}^T \end{pmatrix} & (12) \end{aligned}$$

### 3.1.3. Initialization of potentials for cliques and separators

Clique potential is the product of nodes potentials for all nodes assigned to it. If there is no node assigned to a clique, it is initialized as $\phi(\mathbf{C}) = 1$, namely, it canonical characteristics $(g, \mathbf{h}, \mathbf{K}) = (0, 0, 0)$. Same initializtion is applied to every separator, $\phi(\mathbf{S}) = 1$.

Multiplication of potentials and other basic operations applied to potentials are described in the next section.

## 3.2. Basic operations for CG potential

### 3.2.1. Extension

Basically, CG potential is a representation of CPD (for node) or product of CPDs (for clique, which is a set of nodes). Therefore, the domain of the potential goes with the variables involved in CPDs. Let us use node potential for explanation, because clique potential is just the product of node potentials. We know $g$ in any case is always a scalar, $\mathbf{h}$ and $\mathbf{K}$ are column vector and square matrix respectively. Size of $\mathbf{h}$ and $\mathbf{K}$ depend on how many variables in the domain of CPD. For a node with both discrete and continuous parents, given a particular realization of discrete parents, size of $\mathbf{h}$ will be $n_p \times 1$, where $n_p$ is the number of continuous parents of this node; and size of $\mathbf{K}$ is $n_p \times n_p$.

Sometimes, we need to operate on one potential with another one with bigger domain (for example, when we do multiplication of two potentials with different domain sizes). This is when we need to extend potential to make them in the same size. And it is simply done by just adding corresponding 0s to the positions. For example, let $\phi(i, y)$ has $g(i), \mathbf{h}(i), \mathbf{K}(i)$. If for some reason, we need to extend it to have one more variable $z$ included, the new potential $\phi(i, y, z)$ will have $\tilde{g}(i), \tilde{\mathbf{h}}(i)$, and $\tilde{\mathbf{K}}(i)$ as below:

$$\tilde{g}(i) = g(i), \quad \tilde{\mathbf{h}}(i) = \begin{pmatrix} \mathbf{h}(i) \\ 0 \end{pmatrix}, \quad \tilde{\mathbf{K}}(i) = \begin{pmatrix} \mathbf{K}(i) & 0 \\ 0 & 0 \end{pmatrix}.$$

### 3.2.2. Entering evidence

There are two types of evidence in hybrid Bayesian networks. One is the observed state of discrete variable. Another is a fixed observed value for continuous variable. Once a variable is observed regardless of discrete or continuous, its potential is changed and further the potentials of all cliques/separators that contain this variable are affected and need to be modified.

- Discrete evidence — when discrete variable is observed to be in one particular state, first we know that all of its other states are excluded from the potential. If there is any continuous variable in the same domain, we need to handle it accordingly. Also, we enter this discrete evidence by modifying its potential to be the one with characteristic $g = log(P(A_{i^*}))$, where $A_{i^*}$ is the observing state.

- Continuous evidence — Let us assume that a potential with domain $(\mathbf{x}, \mathbf{y})$, has canonical characteristcs $g, \mathbf{h}, \mathbf{K}$ and,

$$\mathbf{h} = \begin{pmatrix} \mathbf{h_x} \\ \mathbf{h_y} \end{pmatrix}, \quad \mathbf{K} = \begin{pmatrix} \mathbf{K_{xx}} & \mathbf{K_{xy}} \\ \mathbf{K_{yx}} & \mathbf{K_{yy}} \end{pmatrix}.$$

Considering that $\mathbf{y}$ is observed to be a specific value $\mathbf{y}^*$, the original potential has to integrate this finding and transform to be a new potential with domain variable $\mathbf{x}$ only. The transformed potential will have canonical characteristics $g^*, \mathbf{h}^*, \mathbf{K}^*$ as the following:

$$g^* = g + \mathbf{h}_\mathbf{y}^T \mathbf{y}^* - \frac{1}{2}\mathbf{y}^{*T}\mathbf{K_{yy}}\mathbf{y}^* \tag{13}$$

$$\mathbf{h}^* = \mathbf{h_x} - \mathbf{K_{xy}}\mathbf{y}^* \tag{14}$$

$$\mathbf{K}^* = \mathbf{K_{xx}} \tag{15}$$

In specail case, if $\mathbf{X}$ is a root continuous node and it is observed as $\mathbf{x}^*$, then the original potential for $\mathbf{X}$ $(g, \mathbf{h}, \mathbf{K})$ will be changed to have canonical characteristics as $(g^*, 0, 0)$, where $g^*$ is computed as $g^* = g + \mathbf{h}^T\mathbf{x}^* - \frac{1}{2}\mathbf{x}^{*T}\mathbf{K}\mathbf{x}^*$.

### 3.2.3. Multiplication

Multiplication of potentials is simple as addition of the canonical characteristics, shown as below:

$$(g_1, \mathbf{h}_1, \mathbf{K}_1) * (g_2, \mathbf{h}_2, \mathbf{K}_2) = (g_1 + g_2, \mathbf{h}_1 + \mathbf{h}_2, \mathbf{K}_1 + \mathbf{K}_2).$$

### 3.2.4. Division

Similarly as multiplication, dividing potentials is computed as subtraction of the canonical characteristics.

$$(g_1, \mathbf{h}_1, \mathbf{K}_1) / (g_2, \mathbf{h}_2, \mathbf{K}_2) = (g_1 - g_2, \mathbf{h}_1 - \mathbf{h}_2, \mathbf{K}_1 - \mathbf{K}_2).$$

However, we need to make sure that the potential in the position of denominator should not be zero. And if $\phi_1(\mathbf{x}) = 0$, then $\phi_1(\mathbf{x})/\phi_2(\mathbf{x}) = 0$.

### 3.2.5. Marginalizing potential onto smaller domain

Marginalization shall be the most complicated operations for CG potential. The potential may have both discrete and continuous variables in the domain. If we use $\mathbf{i}$ to denote the set of discrete variables involved in the potential, it generally has a set of canonical characteristics $g_i, \mathbf{h_i}$ and $\mathbf{K_i}$ for each of combination of instantiations of discrete variables. Marginalization of CG potential has to be handled differently depending on what type of variable we are marginalizing over.

**Case 1 - marginalizing over continuous variable only:**

Without loss of generality, let us assume that in the potential, it has discrete variables $\mathbf{i}$, and set of continuous variables $\mathbf{y}$ consisting of two sets $\mathbf{y_1}$, and $\mathbf{y_2}$. Marginalizing the potential $\phi(\mathbf{i}, \mathbf{y_1}, \mathbf{y_2})$ over $\mathbf{y_1}$ is to do the integration $\int \phi(\mathbf{i}, \mathbf{y_1}, \mathbf{y_2})\, d\mathbf{y_1}$. Let

$$\mathbf{y} = \begin{pmatrix} \mathbf{y_1} \\ \mathbf{y_2} \end{pmatrix},$$

and for each realization of $\mathbf{i}$, the potential $\phi(\mathbf{i}, \mathbf{y_1}, \mathbf{y_2})$ has canonical characteristics $g(\mathbf{i}), \mathbf{h}(\mathbf{i}), \mathbf{K}(\mathbf{i})$. Correspondingly to $\mathbf{y_1}, \mathbf{y_2}$, we further assume that,

$$\mathbf{h}(\mathbf{i}) = \begin{pmatrix} \mathbf{h}_1(\mathbf{i}) \\ \mathbf{h}_2(\mathbf{i}) \end{pmatrix}, \quad \mathbf{K}(\mathbf{i}) = \begin{pmatrix} \mathbf{K}_{11}(\mathbf{i}) & \mathbf{K}_{12}(\mathbf{i}) \\ \mathbf{K}_{21}(\mathbf{i}) & \mathbf{K}_{22}(\mathbf{i}) \end{pmatrix},$$

where $\mathbf{y_1}$ has dimension $n_1$ and $\mathbf{y_2}$ has dimension $n_2$. If and only if $\mathbf{K}_{11}$ is positive definite, the integral $\int \phi(\mathbf{i}, \mathbf{y_1}, \mathbf{y_2})\ d\mathbf{y_1}$ is finite and has the result as a potential with canonical characteristics $\tilde{g}(\mathbf{i}), \tilde{\mathbf{h}}(\mathbf{i}), \tilde{\mathbf{K}}(\mathbf{i})$, computed by the following equations:

$$\begin{aligned} \tilde{g}(\mathbf{i}) &= g(\mathbf{i}) + 0.5 * \{n_1 * log(2\pi) - \\ &\qquad log[det(\mathbf{K}_{11}(\mathbf{i}))] + \mathbf{h}_1(\mathbf{i})^T \mathbf{K}_{11}(\mathbf{i})^{-1} \mathbf{h}_1(\mathbf{i})\} \qquad (16) \\ \tilde{\mathbf{h}}(\mathbf{i}) &= \mathbf{h}_2(\mathbf{i}) - \mathbf{K}_{21}(\mathbf{i}) \mathbf{K}_{11}(\mathbf{i})^{-1} \mathbf{h}_1(\mathbf{i}) \qquad (17) \\ \tilde{\mathbf{K}}(\mathbf{i}) &= \mathbf{K}_{22}(\mathbf{i}) - \mathbf{K}_{21}(\mathbf{i}) \mathbf{K}_{11}(\mathbf{i})^{-1} \mathbf{K}_{12}(\mathbf{i}) \qquad (18) \end{aligned}$$

For derivation of these equations above, please refer to [7]. Note that it is the key to find the right block of matrix and vector for implementing the computations.

As a special case, sometimes we have to marginalize over all continuous variables in the domain, and have only discrete variables left, which means that we have one hybrid clique passing message to a discrete clique or another hybrid clique but have only discrete variables in common. In the example network Poly8CLG, we do have such a case when we pass message from clique $(B, Y, W)$ to clique $(B, C)$ — marginalizing over continuous variables $Y, W$ to have the potential projected onto discrete variable $B$. In this case, we only need to calculate $\tilde{g}(\mathbf{i})$ by still using Equation (16) with $\mathbf{h}_1(\mathbf{i})$ and $\mathbf{K}_{11}(\mathbf{i})$ actually being the original $\mathbf{h}(\mathbf{i})$ and $\mathbf{K}(\mathbf{i})$ respectively; and of course, $n_1$ in the equation should be the dimension of the whole continous set in original potential; both $\tilde{\mathbf{h}}(\mathbf{i})$ and $\tilde{\mathbf{K}}(\mathbf{i})$ are zeros.

**Case 2 - marginalizing over discrete variable only:**

Similarly, let us assume that the potential has two sets of discrete variables $\mathbf{i}, \mathbf{j}$ and continuous set $\mathbf{y}$, denoted as $\phi(\mathbf{i}, \mathbf{j}, \mathbf{y})$ with canonical characteristics $g(\mathbf{i}, \mathbf{j}), \mathbf{h}(\mathbf{i}, \mathbf{j}), \mathbf{K}(\mathbf{i}, \mathbf{j})$. Marginalizing the potential over $\mathbf{j}$ is equivalent to summing out $\mathbf{j}$:

$$\tilde{\phi}(\mathbf{i}, \mathbf{y}) = \sum_{\mathbf{j}} \phi(\mathbf{i}, \mathbf{j}, \mathbf{y}),$$

where $\tilde{\phi}(\mathbf{i}, \mathbf{y})$ is the new marginalized potential. However, simple addition of CG potentials does not necessarily result in a CG potential. If we marginalize a pure discrete potential, there is then no computation for $\mathbf{h}, \mathbf{K}$ because discrete potential has $\mathbf{h}, \mathbf{K}$ as zeros. So we can do simple addition to compute $g$ in this case as we do for pure discrete Bayesian networks. But if the potential has both discrete and continuous variables like $\phi(\mathbf{i}, \mathbf{j}, \mathbf{y})$, and we marginalize over part of discrete variables $\mathbf{j}$, the computation of summation of potentials is more complicated since in general, $\mathbf{h}, \mathbf{K}$ depend on both $\mathbf{i}, \mathbf{j}$. A better way to handle this case is to convert the potential from canonical form into moment form, so the procedure will be easier to describe and the computation will carry out easier too.

Let us first introduce the moment characteristics $p, \mathbf{u}, \mathbf{\Sigma}$, for CG potential: (1) $p$ represents the probability of the discrete event, i.e., the joint probability of discrete variables in the domain; (2) $\mathbf{u}$ represent the mean vector of CG given discrete variables in the domain; (3) $\mathbf{\Sigma}$ represents the covariance matrix of CG given discrete variables in the domain. For potential $\phi(\mathbf{i}, \mathbf{j}, \mathbf{y})$, its moment characteristics $p(\mathbf{i}, \mathbf{j}), \mathbf{u}(\mathbf{i}, \mathbf{j})$, and $\mathbf{\Sigma}(\mathbf{i}, \mathbf{j})$ are the following:

$$p(\mathbf{i}, \mathbf{j}) = P(\mathbf{I} = \mathbf{i}, \mathbf{J} = \mathbf{j}), \qquad \mathbf{u}(\mathbf{i}, \mathbf{j}) = E(\mathbf{y} | \mathbf{I} = \mathbf{i}, \mathbf{J} = \mathbf{j}), \qquad \mathbf{\Sigma}(\mathbf{i}, \mathbf{j}) = V(\mathbf{Y} | \mathbf{I} = \mathbf{i}, \mathbf{J} = \mathbf{j}).$$

Let the marginal potential $\tilde{\phi}(\mathbf{i}, \mathbf{y})$ has moment characteristics $\tilde{p}(\mathbf{i})$, $\tilde{\mathbf{u}}(\mathbf{i})$, and $\tilde{\mathbf{\Sigma}}(\mathbf{i})$. Now, it is ready to list the equations to compute moment characteristics for marginalizing potential

$\phi(\mathbf{i},\mathbf{j},\mathbf{y})$ over $\mathbf{j}$ to get $\tilde{\phi}(\mathbf{i},\mathbf{y})$:

$$\tilde{p}(\mathbf{i}) = \sum_{\mathbf{j}} p(\mathbf{i},\mathbf{j}) \tag{19}$$

$$\tilde{\mathbf{u}}(\mathbf{i}) = \sum_{\mathbf{j}} \mathbf{u}(\mathbf{i},\mathbf{j})p(\mathbf{i},\mathbf{j})/\tilde{p}(\mathbf{i}) \tag{20}$$

$$\begin{aligned}\tilde{\mathbf{\Sigma}}(\mathbf{i}) = & \sum_{\mathbf{j}} \mathbf{\Sigma}(\mathbf{i},\mathbf{j})p(\mathbf{i},\mathbf{j})/\tilde{p}(\mathbf{i}) + \\ & \sum_{\mathbf{j}} [\mathbf{u}(\mathbf{i},\mathbf{j}) - \tilde{\mathbf{u}}(\mathbf{i})]^T[\mathbf{u}(\mathbf{i},\mathbf{j}) - \tilde{\mathbf{u}}(\mathbf{i})]p(\mathbf{i},\mathbf{j})/\tilde{p}(\mathbf{i})\end{aligned} \tag{21}$$

Recall that initially we represent CG potential in canonical form, and most of computations described so far are carried out in canonical form except the above marginalization over discrete variables using moment form. In fact, these two representation forms for CG potential can be converted each other. Depending on different situations, we choose the one more convenient for computation and efficiency. For sure at the last stage of inference algorithm, it is always to provide the moment form representation for Gaussian posterior distribution.

To make the report complete and self-sufficient, I also list the equations to convert between canonical form and moment form for CG potential as follows. Let potential $\phi(\mathbf{i},\mathbf{x})$ has canonical characteristics $g(\mathbf{i}),\mathbf{h}(\mathbf{i}),\mathbf{K}(\mathbf{i})$ and moment characteristics $p(\mathbf{i}),\mathbf{u}(\mathbf{i}),\mathbf{\Sigma}(\mathbf{i})$, then,

$$\mathbf{\Sigma}(\mathbf{i}) = \mathbf{K}^{-1} \tag{22}$$

$$\mathbf{u}(\mathbf{i}) = \mathbf{\Sigma}(\mathbf{i}) * \mathbf{h}(\mathbf{i}) \tag{23}$$

$$p(\mathbf{i}) = exp\{g(\mathbf{i}) - 0.5 * log[det(\mathbf{K})] - n * log(2\pi) - \mathbf{u}^T\mathbf{K}\mathbf{u}\} \tag{24}$$

where $n$ is the dimension of $\mathbf{x}$, and

$$\mathbf{K}(\mathbf{i}) = \mathbf{\Sigma}^{-1} \tag{25}$$

$$\mathbf{h}(\mathbf{i}) = \mathbf{K}(\mathbf{i}) * \mathbf{u}(\mathbf{i}) \tag{26}$$

$$g(\mathbf{i}) = log(p(\mathbf{i}) + 0.5 * log[det(\mathbf{K})] - n * log(2\pi) - \mathbf{u}^T\mathbf{K}\mathbf{u} \tag{27}$$

These equations can be derived by Gaussian formulas and potential forms. For special cases such as $\mathbf{h} = \mathbf{0}, \mathbf{K} = \mathbf{0}$, the conversion between canonical and moment forms is straightforward and simpler.

We summarize the procedure of marginalizing over discrete variables as: (1)since the initial potential is in canonical form, we first convert $\phi(\mathbf{i},\mathbf{j},\mathbf{y})$ into moment form; (2) use equations (19), (20), and (21) to compute the moment characteristics for the new marginal potential $\tilde{\phi}(\mathbf{i}, \mathbf{y})$; (3)convert moment characteristics of $\tilde{\phi}(\mathbf{i}, \mathbf{y})$ back into canonical form.

**Case 3 - marginalizing over both continuous and discrete variables:**

In case that we need to marginalize both continuous and discrete variables, we first marginalize over the continuous variables, and then over the discrete ones.

## 3.3. Message passing in clique tree

After finding the strong root clique, as mentioned before, we build the directed clique tree, mainly for showing the direction of message flows in the tree. The first stream of message flow is always from leaves to the root, called upstream message propagation, also known as collecting evidence from the point of view of the strong root clique. After the strong root clique obtains all information from all of other cliques, it distributs its updated potential to all of other cliques. This second message flow process is called downstream message propagation, also known as distributing evidence. When downstream flow is done, all potentials of cliques are updated, containing the correct joint distribution of all nodes in the clique. Furthermore, marginalizing onto particular variable provides the posterior distribution of this particular node given evidence. For discrete vararible, this gives the correct updated probabilities of states; for continuous vararible, the mean and variance of its posterior distribution are obtained.

Let us assume that two cliques $\mathbf{C}_i, \mathbf{C}_j$ are neighbors in a clique tree, and separator $\mathbf{S}_{ij}$ is associated to the edge between $\mathbf{C}_i, \mathbf{C}_j$. Potentials for $\mathbf{C}_i, \mathbf{C}_j$ and $\mathbf{S}_{ij}$ are $\phi(\mathbf{C}_i), \phi(\mathbf{C}_j)$ and $\phi(\mathbf{S}_{ij})$ respectively. Sending message from $\mathbf{C}_i$ to $\mathbf{C}_j$ along the separator $\mathbf{S}_{ij}$ follows the message passing protocol presented in Table 3. The sending process is also known as absorption, namely, clique $\mathbf{C}_j$ absorbs information from clique $\mathbf{C}_i$ via their separator $\mathbf{S}_{ij}$.

### 3.3.1. Upstream message flow

As described in section 2.4, we build a directed clique tree after finding a strong root. The only purpose of constructing this directed clique tree is for propagating message in an appropriate manner. Upstream message flow is the process of collecting information from leaves to root. In particular, (1) all leave cliques in the directed clique tree send messages

**Table 3.** Message passing protocol

1. Let $\phi(\mathbf{S}_{ij})' = \sum_{\mathbf{C}_i \setminus \mathbf{S}_{ij}} \phi(\mathbf{C}_i)$, — marginalizing the potential $\phi(\mathbf{C}_i)$ onto the domain of separator $\phi(\mathbf{S}_{ij})$, i.e., projecting it to the domain of separator.
2. Let $\mathcal{L}(\mathbf{S}_{ij}) = \frac{\phi(\mathbf{S}_{ij})'}{\phi(\mathbf{S}_{ij})}$, — dividing the new potential of separator $\phi(\mathbf{S}_{ij})$ by its old one. The ratio $\mathcal{L}(\mathbf{S}_{ij})$ is served as information ratio, also called "likelihood ratio", to update information by filtering out the redundant part.
3. Let $\phi(\mathbf{S}_{ij}) = \phi(\mathbf{S}_{ij})'$, — storing the new potential of the separator for next round message passing.
4. Let $\phi(\mathbf{C}_j) = \phi(\mathbf{C}_j) * \mathcal{L}(\mathbf{S}_{ij})$, — multiplying information ratio from the separator to update potential of $\phi(\mathbf{C}_j)$ .

to their unique parent; (2) intermediate cliques sends message to their unique parent after receiving message from all of their children; (3) the strong root receives message from all of its children. During upstream message flow process, all marginalizations are strong because they are always over either continuous variables or over discrete variables in pure discrete domain.

### 3.3.2. Downstream message flow

Downstream message flow has the opposite direction and always proceeds after upstream message flow. There will be weak marginalizations in downstream message passing, however, the results are still exact in representing the true frist two moments of Gaussian because the cliques we send message from already received all information from their children in upstream message pass, therefore their potentials represent the joint probability function.

## 3.4. Calibration of clique tree

Calibrating a clique tree is to obtain the correct marginal distribution for every clique based on CPDs defined in the BN and evidence observed. It is possible to calibrate a clique tree only if it is stronly rooted. Now it is ready to summarize the calibration process of a clique tree, also known as compiling, in Table 4. Note that when new observations come in, we

**Table 4.** Calibrating a stronly rooted clique tree.

1. Assign node to its corresponding clique.
2. Initialize node potential. If a node is observed, enter evidence for its potential.
3. Initialize clique potential by multiplying potentials of assigning nodes.
4. Initialize potentials for separators.
5. Run upstream message passing.
6. Run downstream message passing.

need to recalibrate the clique tree to update all marginal posterior distributions.

### 3.5. Computing marginal distribution for hidden variables

After calibrating a clique tree, we have correct joint distribution over every clique. It is straightforward to compute the marginal posterior distribution for any particular hidden variable by marginalizing over all of other variables in the clique domain.

## 4. SUMMARY AND DISCUSSION

This report presents a detailed implementation of Clique Tree algorithm, which is probably the most popular exact inference algorithm in the literature. We cover all of details we can think about from theoretical explanations to implementation level pseudo codes for thorough understanding of the algorithm, and hopefully, trying to make a developer easy to program the algorithm.

Clique Tree algorithm can provide exact posterior distributions for pure discrete BNs. For CLG, it provides exact solutions in terms of the correct first two moments for continuous variables. Compared to pure BNs, Clique Tree algorithm is a bit complicated for CLG and has several extra processes to ensure the correct answers. The complexity of Clique Tree algorithm is exponential to the size of discrete parents in CLG, which is same as the hybrid message passing algorithm we presented in 2007 [13], by separating hybrid BNs into pure discrete and pure continuous segments.

Clique Tree algorithm has to transfer the original BN into a tree before computations. Graph transformations then takes extra time other than computations of message passing based on protocal shown in Table 3. Finding an optimal elimination order in triangulation process is NP-hard. This is not surprise because inference for BN is generally NP-hard [2] [3]. For CLG, it is very critical to ensure a strongly triangulated graph before identifying the right cliques, and further building the corresponding clique tree. It is guaranteed that at least one strong root exists in the clique tree constructed from a strongly triangulated graph. Then finding the right strong root in the clique tree for CLG is so important that otherwise the results will be incoreect.

Regarding to the algorithm complexity, let us look at the clique size. In CLG, we call discrete parents, defined in Section 2.2, as interface nodes because they are the interface between discrete and continuous variables in the network. To satisfy the strong triangulation requirement, it has to connect pairwise interface nodes if they are not connected yet in the original graph. This results in a clique containing all of interface nodes and at least one continuous node — let us call this clique Bridge (from the meaning that it may bottleneck the algorithm). The inevitable existence of Bridge can be simply induced by the strong triangulation process according to strong elimination order. Indeed, it is also mentioned and proved in [8]. Bridge has the size as the product of state spaces of all interface nodes. The potentail of Bridge could be huge and marginalization of this potential will be exponential to its size.

## REFERENCES

1. Eugene Charniak. *Bayesian Networks without Tears: making Bayesian networks more accessible to the probabilistically unsophisticated*, AI Magazine, v.12 no.4, p.50-63, 1991.
2. G. F. Cooper. *The computational complexity of probabilistic inference using Bayesian belief networks.* Artificial Intelligence, vol. 42, pp. 393-405, 1990.
3. P. Dagum and M. Luby. *Approximating probabilistic inference in Bayesian belief networks is NP–hard.*Artificial Intelligence, vol. 60, pp. 141-153, 1993.
4. H. Guo and W. Hsu, *A Survey of Algorithms for Real-time Bayesian Network Inference*,

AAAI/KDD/UAI–2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems, Edmonton, Alberta, Canada, 2002.

5. F.V. Jensen. *An Introduction to Bayesian Networks*. Springer-Verlag, New York, 1996.
6. Finn V. Jensen, Frank Jensen. *Optimal Junction Trees*. UAI, Morgan Kaufmann,San Francisco, CA, 1994.
7. Steffen L. Lauritzen. *Propagation of probabilities, means, and variances in mixed graphical association models*. JASA, 87(420):1089–1108, 1992.
8. Uri N. Lerner. *Hybrid Bayesian Networks for Reasoning about Complex Systems*. Ph.D. Thesis, Stanford University, October, 2002.
9. S. L. Lauritzen and F. Jensen. *Stable local computations with conditional Gaussian distributions. Statistics and Computing*, vol.11, no.2, pp191–203, April 2001.
10. S.L. Lauritzen and D.J. Spiegelhalter, *Local Computations with Probabilities on Graphical Structures and Their Applications to Expert Systems*, Proceedings of the Royal Statistical Society, Series B., 50, 157-224, 1988.
11. R. E. Neapolitan. *Probabilistic Reasoning in Expert Systems*. John Wiley & Sons, NY, 1990.
12. J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kauffman, San Mateo, 1988.
13. Wei Sun and K.C. Chang, *Hybrid Message Passing for Mixed Bayesian Networks*. In Proceedings of the 10th International Conference on Information Fusion, Quebec, Canada, July 2007.
14. Sun, Wei. *Efficient Inference for Hybrid Bayesian Networks*. Ph.D. Dissertation, George Mason University, Fairfax, VA 2007.