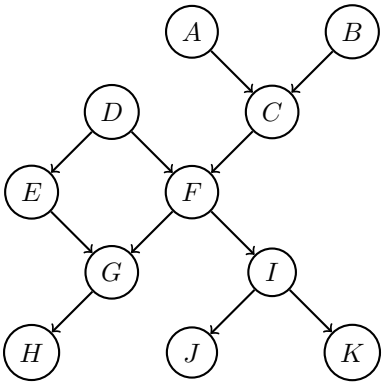
Project 7

Richard Affolter

21 April, 2021

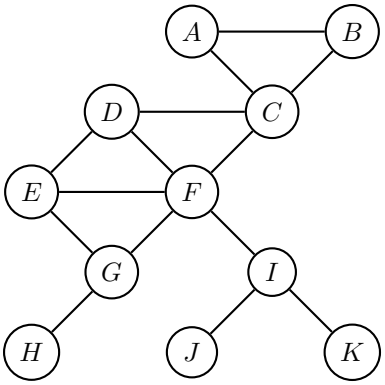
Problem 17: Junction Tree Algorithm

Consider the Bayesian network on the variables $U = \{A, ..., K\}$ given by the following graph:

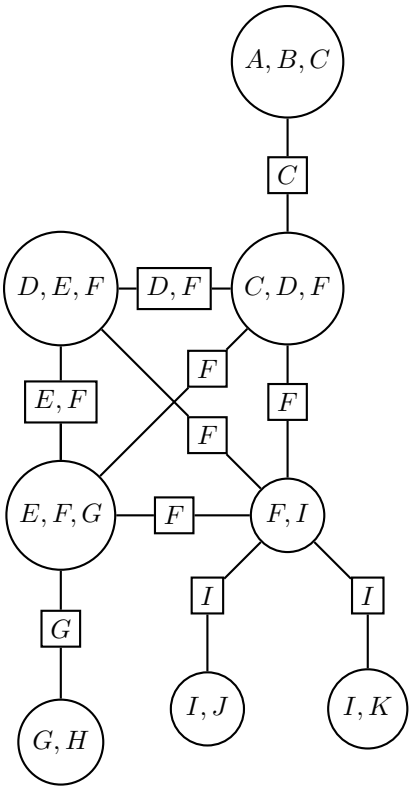


(a) Build the *Junction Tree* of the network.

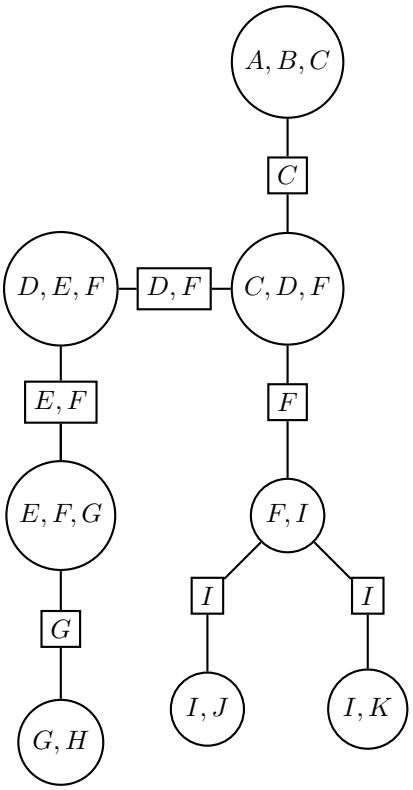
After moralizing we get the following graph:



Since the graph is already triangulated we can directly construct the Junction graph



Finally we can get the Junction tree by constructing the maximum spanning tree of the Junction graph



(b)) Write the joint probability $P(U)$ in terms of the cluster and separator potentials

$$P(U) = \frac{\psi(A,B,C)\psi(C,D,F)\psi(D,E,F)\psi(E,F,G)\psi(F,I)\psi(G,H)\psi(I,J)\psi(I,K)}{\psi(C)\psi(D,F)\psi(E,F)\psi(F)\psi(G)\psi(I)^2}$$
$$= P(A)P(B)P(C|A,B)P(D)P(E|D)P(F|C,D)P(G|E,F)P(I|F)P(H|G)P(J|I)P(K|I)$$

This equality holds, for example, for $\psi(A,B,C) = P(A)P(B)P(C|A,B)$, $\psi(C,D,F) = P(D)P(F|C,D)$, $\psi(D,E,F) = P(E|D)P(F|D)$, $\psi(E,F,G) = P(G|E,F)$, $\psi(F,I) = P(I|F)$, $\psi(G,H) = P(H|G)$, $\psi(I,J) = P(J|I)$, $\psi(I,K) = P(K|I)$, $\psi(D,F) = P(F|D)$ and $\psi(EF) = \psi(C) = \psi(F) = \psi(I) = 1$

Problem 18: Benefits of storing messages

(a) Write the formula for recursively computing the forward and backward messages.

$$\mu_\alpha(x_n) = \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n)\mu_\alpha(x_{n-1})$$

$$\mu_\beta(x_n) = \sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1})\mu_\beta(x_{n+1})$$

(b) What is the complexity of computing the marginal probability $P(X_4 = 1)$ using message passing?

We know that $P(X_4) = \frac{1}{Z}\mu_\alpha(x_4)\mu_\beta(x_4)$. To compute $\mu_\alpha(x_4)$ we need to recursively also calculate $\mu_\alpha(x_3)$ and $\mu_\alpha(x_2)$ which gives a total of 3 forward messages. To compute $\mu_\beta(x_4)$ we need 1 backward message

So in total we have 3 forward + 1 backward = 4 messages.

For each message we have to marginalize the K states of x_{n-1} for each of the K states of x_n resulting in a complexity of K^2. If we assume that each variable is binary we get $K^2 = 2^2 = 4$. To calculate the normalising constant Z we need to sum up all K possible states of $P(X_4)$.

To summarize we need $4K^2 + K$ summation steps, which can be represented as $(N-1)K^2 + K \in \mathcal{O}(NK^2)$

(c) If you store all the messages, what is the complexity of computing all marginal probability distributions X_1, X_2, X_3, X_4, and X_5? How about the general case, with a chain of length N where each node can assume K values? Assume that storing and multiplying is free, so that only summation counts for the complexity

To compute all marginal probability distributions we need to compute all forward and all backward messages. For a chain we have $N-1$ forward and $N-1$ backward messages. The normalising constant can be computed at any node, so we have again K summations. So in total we get $2(N-1)K^2 + K \in \mathcal{O}(NK^2)$. For the specific case of $X_1 - X_5$ we have of course the same solution with $N = 5$.

Problem 19(data analysis): Message passing on a chain

(a) Store clique potentials in an R object

```
# initialize the potentials
potentials = array(dim = c(2, 2, 4), dimnames =
                     list(c("0", "1"), c("0", "1"),
                          c("Psi12", "Psi23", "Psi34", "Psi45")))
```

```r
# conditional probability distributions, p_Xn = p(Xn)==1.

# the first pos. is if X(n-1)==0, second pos if X(n-1)==1
p_X1 = c(1/3, 1/3)
p_X2 = c(4/5, 2/3)
p_X3 = c(5/7, 1/3)
p_X4 = c(3/5, 2/5)
p_X5 = c(1/2, 7/9)

# compute potentials
# Psi12 = p(X1) * p(X2|X1), Psi23 = p(X3|X2), ...
potentials[,,"Psi12"] = rbind(1-p_X1, p_X1) * cbind(1-p_X2,p_X2)
potentials[,,"Psi23"] = cbind(1-p_X3, p_X3)
potentials[,,"Psi34"] = cbind(1-p_X4, p_X4)
potentials[,,"Psi45"] = cbind(1-p_X5, p_X5)

print(potentials)
```

```
## , , Psi12
##
##           0         1
## 0 0.1333333 0.5333333
## 1 0.1111111 0.2222222
##
## , , Psi23
##
##           0         1
## 0 0.2857143 0.7142857
## 1 0.6666667 0.3333333
##
## , , Psi34
##
##     0   1
## 0 0.4 0.6
## 1 0.6 0.4
##
## , , Psi45
##
##           0         1
## 0 0.5000000 0.5000000
## 1 0.2222222 0.7777778
```

(b) Computing forward messages

We want to get the forward message in the following form

$$
\begin{aligned}
\mu_\alpha(x_n) &= (\mu_\alpha(X_n = 0), \mu_\alpha(X_n = 1)) \\
&= \begin{pmatrix} \psi_{n-1,n}(0,0) \cdot \mu_\alpha(X_{n-1} = 0) + \psi_{n-1,n}(1,0) \cdot \mu_\alpha(X_{n-1} = 1) \\ \psi_{n-1,n}(0,1) \cdot \mu_\alpha(X_{n-1} = 0) + \psi_{n-1,n}(1,1) \cdot \mu_\alpha(X_{n-1} = 1) \end{pmatrix}^T \\
&= (\mu_\alpha(X_{n-1} = 0), \mu_\alpha(X_{n-1} = 1)) \begin{pmatrix} \psi_{n-1,n}(0,0) & \psi_{n-1,n}(0,1) \\ \psi_{n-1,n}(1,0) & \psi_{n-1,n}(1,1) \end{pmatrix} \\
&= (\mu_\alpha(X_{n-1} = 0), \mu_\alpha(X_{n-1} = 1)) \, \Psi_{n-1,n}
\end{aligned}
$$

```
forward_messages = matrix(NA, nrow = 5, ncol = 2,
                          dimnames = list(c("mu_alpha_x1","mu_alpha_x2",
                                            "mu_alpha_x3","mu_alpha_x4",
                                            "mu_alpha_x5")))
forward_messages["mu_alpha_x1",]=c(1,1)

compute_forward = function(idx){
  if (idx==1){
    return(forward_messages)
  }
  else{
    tmp_messages = compute_forward(idx-1)
    tmp_messages[idx,] = tmp_messages[idx-1,] %*% potentials[,,idx-1]
    return(tmp_messages)
  }
}

forward_msg = compute_forward(5)
print(forward_msg)
```

```
##                   [,1]      [,2]
## mu_alpha_x1 1.0000000 1.0000000
## mu_alpha_x2 0.2444444 0.7555556
## mu_alpha_x3 0.5735450 0.4264550
## mu_alpha_x4 0.4852910 0.5147090
## mu_alpha_x5 0.3570253 0.6429747
```

(c) Computing backward messages

Similarly to (b) we can get the backwards messages in the form

$$
\begin{aligned}
\mu_\beta(x_n) &= (\mu_\beta(X_n = 0), \mu_\beta(X_n = 1)) \\
&= \begin{pmatrix} \psi_{n,n+1}(0,0) \cdot \mu_\beta(X_{n+1} = 0) + \psi_{n,n+1}(0,1) \cdot \mu_\beta(X_{n+1} = 1) \\ \psi_{n,n+1}(1,0) \cdot \mu_\beta(X_{n+1} = 0) + \psi_{n,n+1}(1,1) \cdot \mu_\beta(X_{n+1} = 1) \end{pmatrix}^T \\
&= \begin{pmatrix} \psi_{n,n+1}(0,0) & \psi_{n,n+1}(0,1) \\ \psi_{n,n+1}(1,0) & \psi_{n,n+1}(1,1) \end{pmatrix} \begin{pmatrix} \mu_\beta(X_{n+1} = 0) \\ \mu_\beta(X_{n+1} = 1) \end{pmatrix} \\
&= \Psi_{n,n} \, (\mu_\beta(X_{n+1} = 0), \mu_\beta(X_{n+1} = 1))^T
\end{aligned}
$$

```
backward_messages = matrix(NA, nrow = 5, ncol = 2,
                          dimnames = list(c("mu_beta_x1","mu_beta_x2",
                                            "mu_beta_x3","mu_beta_x4",
                                            "mu_beta_x5")))
```

```
backward_messages["mu_beta_x5",]=c(1,1)

compute_backward = function(idx){
  if (idx==5){
    return(backward_messages)
  }
  else{
    tmp_messages = compute_backward(idx+1)
    tmp_messages[idx,] = potentials[,,idx] %*% tmp_messages[idx+1,]
    return(tmp_messages)
  }
}

backward_msg = compute_backward(1)
print(backward_msg)
```

```
##                 [,1]      [,2]
## mu_beta_x1 0.6666667 0.3333333
## mu_beta_x2 1.0000000 1.0000000
## mu_beta_x3 1.0000000 1.0000000
## mu_beta_x4 1.0000000 1.0000000
## mu_beta_x5 1.0000000 1.0000000
```

(d) Compute the marginal probability distribution for each node

```
marginals = forward_msg * backward_msg
rownames(marginals) = c("p_X1","p_X2","p_X3","p_X4","p_X5")
colnames(marginals) = c("Xn=0", "Xn=1")
Z = rowSums(marginals)
print(marginals)
```

```
##           Xn=0      Xn=1
## p_X1 0.6666667 0.3333333
## p_X2 0.2444444 0.7555556
## p_X3 0.5735450 0.4264550
## p_X4 0.4852910 0.5147090
## p_X5 0.3570253 0.6429747
```

We can easily verify that for each node the rowsum is 1 and since the rowsum of the results is equal to $(\sum_{x_n} \mu_\alpha(x_n)\mu_\beta(x_n) = Z)$, the normalizing constant is 1.