UC Berkeley
Department of Electrical Engineering and Computer Science
Department of Statistics

EECS 281A / STAT 241A Statistical Learning Theory

Solutions to Problem Set 2

Fall 2011

Issued: Wednesday, September 28, 2011 **Due:** Monday, October 10, 2011

Reading: For this problem set: Chapters 4, 16.
Total: 40 points.

Problem 2.1

(10 pts) *Conditioning:*

Consider an undirected graph that is a single cycle (a "ring"). Let each of the nodes be discrete-valued, taking on one of K states.

(a) Note that if we condition on one of the nodes then we break the cycle into a chain. We can run the sum-product algorithm on the chain. Devise an algorithm for computing all single-node marginals in the cycle via multiple runs of the sum-product algorithm.

Solution: Since the graphical model for the joint distribution is a ring, we can write the joint probability as a product of edge potentials:

$$p(x) \;\propto\; \psi(x_1, x_2)\psi(x_2, x_3)\cdots\psi(x_{m-1}, x_m)\psi(x_m, x_1).$$

Suppose we condition on $X_1 = \bar{x}_1$. Note that

$$p(x_2, \ldots, x_m \mid \bar{x}_1) \;\propto\; p(\bar{x}_1, x_2, \ldots, x_m) \;\propto\; \psi(\bar{x}_1, x_2)\psi(x_2, x_3)\cdots\psi(x_{m-1}, x_m)\psi(x_m, \bar{x}_1).$$

Hence, we see that the joint distribution over $(x_2, \ldots, x_m)$ of $p(x_2, \ldots, x_m \mid \bar{x}_1)$ can be represented by a chain graph, with edge potentials $\psi(x_i, x_{i+1})$ for $2 \le i \le m-1$, and node potentials $\psi(\bar{x}_1, x_2)$ and $\psi(x_m, \bar{x}_1)$ on nodes 2 and m, respectively. So we can run the sum-product algorithm on the chain to obtain marginal (conditional) probabilities $p(x_i|\bar{x}_1)$, for all $i \neq 1$.

In order to recover the marginal probabilities $p(x_i)$, note that

$$\frac{p(X_1 = x_1|X_2 = x_2)}{p(X_2 = x_2|X_1 = x_1)} = \frac{p(X_1 = x_1)}{p(X_2 = x_2)}.$$

Our algorithm proceeds as follows. Pick two distinct nodes j, k. Run the sum-product algorithm on the chain obtained by conditioning on $X_k = x_k$, for some fixed value x_k; this yields the conditional probabilities $p(X_j = x_j|X_k = x_k)$, for all values x_j. Then for all values x_j of X_j, run the sum-product algorithm conditioned on $X_j = x_j$. By taking quotients, we therefore obtain the ratios $\frac{p(X_j = x_j)}{p(X_k = x_k)}$ for all values x_j. Hence, renormalization yields the marginal $p(x_j)$. Finally, to obtain a single-node marginal $p(x_i)$ for another node i, simply note that

$$p(X_i = x_i, X_j = x_j) = p(X_i = x_i|X_j = x_j)\, p(X_j = x_j).$$

Since we know $p(x_j)$ and we have already computed the conditionals $p(X_i = x_i|X_j = x_j)$ from a previous run of the sum-product algorithm, we can compute all pairwise marginals

$p(X_i = x_i, X_j = x_j)$, then sum up over x_j, to obtain the marginal $p(x_i)$. Since we have run sum-product k times, the overall runtime is $\mathcal{O}\left(mk^3\right)$, where m is the number of nodes and k is the number of values for each x_i.

*Note: Several people claimed in their homework that running sum-product conditioned on $X_1 = \bar{x}_1$ yields the joint distributions $p(x_i, \bar{x}_1)$ for different values of x_i, up to a proportionality constant. Then summing over all values of $\bar{x}_1$ and renormalizing yields the marginal probability $p(x_i)$. The problem with this approach is that running sum-product conditioned on different values of $\bar{x}_1$ does **not** in fact yield the joints $p(x_i, \bar{x}_1)$ up to a constant of proportionality. This stems from the fact that*

$$p(x_2, \dots, x_m \mid \bar{x}_1) = \frac{p(\bar{x}_1, x_2, \dots, x_m)}{p(\bar{x}_1)},$$

so if we run sum-product on the chain obtained by conditioning on $X_1 = \bar{x}_1$, the normalization constant will include a factor of $p(\bar{x}_1)$, which depends on the value of $\bar{x}_1$ being conditioned upon. Therefore, this approach is invalid.

(b) Another approach to computing all single-node marginals in the cycle is to use the junction tree algorithm. What is size of biggest clique that arises during a sensible triangulation of the cycle? Compare the computational complexity of the junction tree algorithm using this triangulation to that of the conditioning-based algorithm in (a).

Solution: The natural triangulation creates $m-2$ triangles in the m-node cycle, with the largest clique size equal to three. We can then create a junction tree with $m-3$ edges among the cliques (see Figure 1). Since the largest clique size is three, the overall runtime is $\mathcal{O}\left(mk^3\right)$, agreeing with the result obtained in (a). Note that the runtime of the junction tree algorithm is exponential in the size of the largest clique (since we must sum up over values of variables in the clique in the message-passing step), and linear in the number of triangles/nodes.

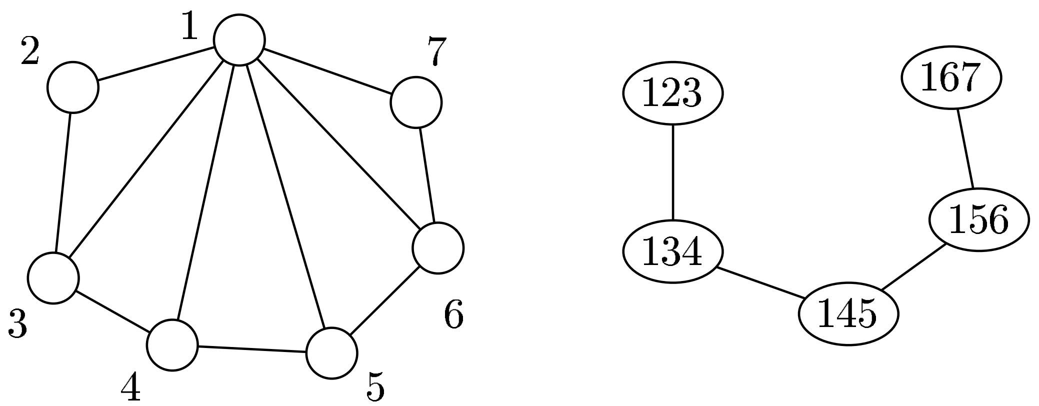


Figure 1: (a) Triangulation of an m-cycle with $m-2$ triangles. (b) Junction tree for the triangulated m-cycle.

Problem 2.2
(10 pts) *Sum-product algorithm from junction tree:*

Consider an undirected tree, $G = (V, E)$, parametrized by pairwise potentials, $\psi_{st}(x_s, x_t)$, for $(s,t) \in E$. Outline the junction tree construction for this model, writing out the propagation rules explicitly, and derive the sum-product algorithm for this model from the junction tree propagation rules.

Solution: We provide two solutions to this problem, one based on induction and the other using a two-phase message passing.

Solution 1 (induction): For clarification, we will use i, j, k, l to represent nodes in the original tree, and use u, v, w to represent nodes in the junction tree.

We first outline the junction tree construction for the model. Specifically, we can construct the junction tree iteratively as follows (note that the maximal cliques in the graph have size 2):

1. Starting from a single clique (i, j), and initialize the set of unprocessed nodes as $V \backslash \{i, j\}$, and the set of processed nodes as $\{i, j\}$;
2. while the set of unprocessed nodes is not empty, do the following:
   (a) select an unprocessed node k that has one of its neighbors (which we denote by k') in the set of processed nodes;
   (b) add a clique (k, k') to the existing clique graph, and connect it to any existing clique that contains k', with the corresponding separator as $\{k'\}$;
   (c) move k from the set of unprocessed nodes to that of processed nodes.

We show that the obtained clique graph is a junction tree by induction. Consider a tree of two nodes, this clearly holds as the clique graph is one single clique. Assuming that this holds for a tree of n nodes ($n \geq 2$). For a tree of $n+1$ nodes, the last processed node would always be a leaf node, say k, so the clique tree construction is equivalent to finding the clique graph of its subtree excluding k, and adding the leaf node using the method described in step (2). By induction, we know that the clique graph of the subtree is a junction tree. When we add the leaf node, no loops can be introduced, so the resulting clique graph is still a tree. Further, denote the newly added clique by (k, k'), from the running intersections property we know that all the cliques in the previous junction tree containing k' are all connected. Adding the clique (k, k') still preserves the running intersections property, thus the resulting new clique tree is a junction tree.

Now we derive the sum-product algorithm for this model by slightly modifying the junction tree propagation rule: instead of updating potentials on the fly as the original algorithm does, we keep the clique potentials unchanged, and instead make the cliques to "relay" messages from its neighbors. Specifically, a clique (i, j) sends a message to its neighbor (j, k) as follows:

$$M_{(i,j)\to(j,k)}(x_j) = \sum_{x_i} \psi_{ij}(x_i, x_j) \prod_{l \in N(i)\backslash\{j\}} M_{(l,i)\to(i,j)}(x_i)$$

After the message passing ends, we freeze the messages and simultaneously update the final potentials as

$$\psi_{ij}^{**}(x_i, x_j) = \psi_{ij}(x_i, x_j) \prod_{l \in N(i)} M_{(l,i)\to(i,j)}(x_i).$$

We now show that such an algorithm is equivalent to the original junction tree propagation by induction. It is easy to verify the equivalency when the junction tree only has two clique nodes. Assume that the equivalency holds for junction trees with n ($n \geq 2$) clique nodes, we will prove that it holds for junction trees with $n+1$ clique nodes. To this end, consider one of the leaf clique node $v = (i, j)$ with neighbor $w = (j, k)$ in the junction tree with $n+1$ cliques. The original propagation states that v propagates potential to w by updating w's potential:

$$\psi_w^*(x_j, x_k) = \psi_w(x_j, x_k) \sum_{x_i} \psi_v(x_i, x_j).$$

Note that $\sum_{x_i} \psi_v(x_i, x_j)$ is exactly the message $M_{v \to w}$ in our algorithm. By induction, we know that our message passing is equivalent to the original propagation on the subtree of n clique nodes, excluding v. Thus, after the message passing (or propagation) on this subtree, the new potential ψ_w^{**} as stated by the original propagation algorithm is[1]

$$\psi_w^{**}(x_j, x_k) = \psi_w^*(x_j, x_k) \prod_{u \in N(w) \setminus \{v\}} M_{u \to w}.$$

Plugging $\psi_w^*(x_j, x_k)$ into this equation, we obtain

$$\psi_w^{**}(x_j, x_k) = \psi_w(x_j, x_k) \prod_{u \in N(w)} M_{u \to w},$$

which is equivalent to the result of our algorithm.

As for v, the original propagation states that

$$\psi_v^{**}(x_i, x_j) = \frac{\sum_{x_k} \psi_w^{**}(x_j, x_k)}{\sum_{x_i} \psi_v(x_i, x_j)} \psi_v(x_i, x_j).$$

Plugging $\psi_w^{**}(x_j, x_k)$ into this equation gives us

$$\psi_v^{**}(x_i, x_j) = \psi_v(x_i, x_j) M_{w \to v}(x_j),$$

which is again equivalent to the result of our algorithm.

In conclusion, our message-passing algorithm gives the same clique potentials as the original junction tree propagation algorithm.

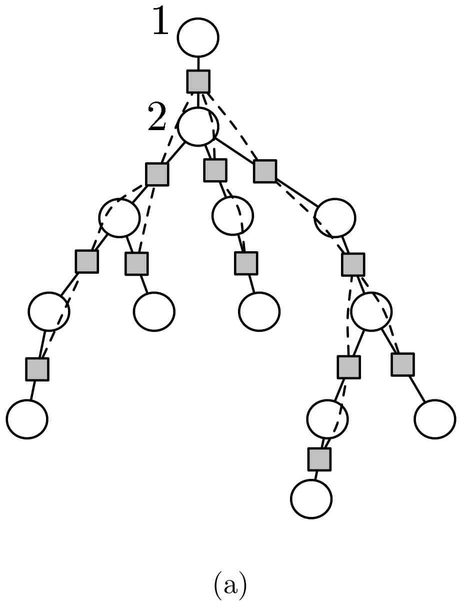
To see how this relates to the sum-product algorithm on the original tree (not the junction tree), notice that the message that $v = (i, j)$ passes to $w = (j, k)$ is equivalent to the message that i passes to j in the original sum-product algorithm. In other words, the junction tree propagation algorithm on the junction tree of an undirected tree is equivalent to the sum-product algorithm on the original tree.

Solution 2 (direct proof): First note that the junction tree algorithm is specified in such a way that if we *only* consider clique potentials on the nodes of the junction tree (ignoring separator potentials), the order of message-passing does not matter. Indeed, suppose (v, w) is an edge of the junction tree, with corresponding clique potentials ψ_v and ψ_w, and separator potential ψ_s. Suppose the first message is passed $v \to w$, and the second message is passed $w \to v$. The first

[1]Note that $M_{u \to w}$ is a function (a table in the context of discrete random variables) of the separator between u and w. For simplicity we omitted the random variables in the equations.

message will update the separator potential to a function ψ_s^* formed by summing out a product of ψ_v and incoming messages, and update the clique potential ψ_w to $\psi_w^* = \psi_w \delta_{v \to w} \psi_s^*$, where $\delta_{v \to w}$ is the message sent from v. When w is ready to pass a message back to v, the message updates the potential ψ_v by first dividing out the value ψ_s^*, and then multiplying together the incoming messages to v, summed over all variables not involved in s. In particular, the factor ψ_s^* cancels out in the update of ψ_v, so the updated values of ψ_v and ψ_w do not depend on whether we first pass a message $v \to w$ or a message $w \to v$ in the junction tree algorithm.

Given an undirected tree, we now specify the operation of the junction tree algorithm as follows (see Figure 2). First take one of the leaf nodes (say node 1, with neighbor node 2) to be the root of the tree. Then place a clique node on each edge of the tree, with potential ψ_{ij} assigned to edge (i, j). Finally, connect clique nodes (i, j) and (j, k) only if nodes i and k are at a different depth in the rooted tree. It is easy to see that this method creates a junction tree. Finally, modify the original tree by eliminating the root node and placing the potential $\sum_{x_1} \psi_{12}(x_1, x_2)$ on its neighbor. We will argue that the sum-product algorithm on this modified tree yields the same updates as the junction tree algorithm on our junction tree.



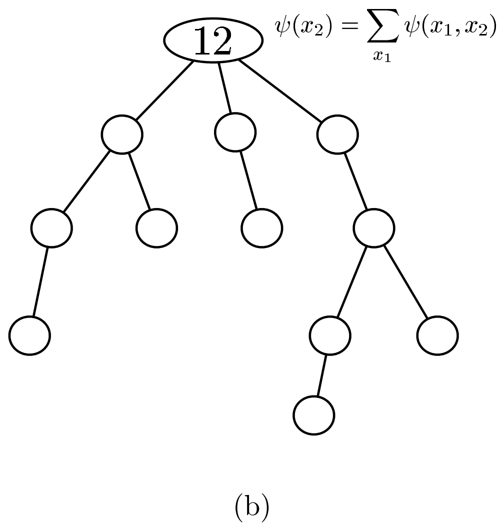


Figure 2: (a) Junction tree formed from an undirected tree. Squares represent clique nodes in the junction tree. (b) Modified tree for the sum-product algorithm. Node 1 has been absorbed into node 2, and the summed-out potential has been placed on node $(1, 2)$.

Consider the clique potentials in the junction tree. By the argument in the first paragraph, the order of message updates does not matter; so let us first send in messages from all the leaves to the root of the junction tree, then distribute messages back to the leaves in a downward pass. If (i, j) is a leaf clique of the junction tree, with i a leaf of the original tree, then the message passed from (i, j) to its neighboring clique (j, k) updates the potential at (j, k) to the function

$$\psi_{jk}^*(x_j, x_k) = \psi_{jk}(x_j, x_k) \sum_{x_i} \psi_{ij}(x_i, x_j)$$

according to the junction tree algorithm. Note that the sum-product algorithm on the original tree would send the message $M_{i\to j}(x_j) = \sum_{x_i} \psi_{ij}(x_i, x_j)$. Hence, passing the message in the junction tree from $(i,j) \to (j,k)$ exactly corresponds to passing the message $i \to j$ in the sum-product algorithm.

Continuing in this manner, it is easy to see that the message-passing updates in the upward pass of the junction tree algorithm are identical to the message-passing updates in an upward pass of the sum-product algorithm, since in each step, we multiply the clique potential by all incoming messages, and sum out the previous variables. We now need to demonstrate a similar correspondence with the messages distributed downward from the root back to the leaves. This, too, is easy to see: Consider a message passed from the root $(1,2)$ of the junction tree to a neighbor $(2,i)$. As we have argued before, the order of message-passing in the junction tree does not matter in our updates to the clique potentials. Hence, we can imagine re-rooting the junction tree at a different leaf clique, in such a way that $(2,i)$ is now closer to the root than $(1,2)$. Then the upward pass of the junction tree sends a message from $(1,2)$ to $(2,i)$, which as we have argued, must agree with the sum-product message sent from $2 \to i$. Similarly, we can argue that all other messages sent in the downward pass of the junction tree correspond to sum-product messages in the tree, by re-rooting the junction tree accordingly.

Finally, consider the separator potentials of the junction tree after all messages have been passed. By the correctness of the junction tree algorithm, the final separator potential $\psi_j^*(x_j)$ between two edges (i,j) and (j,k) is proportional to $p(x_j)$. Furthermore, $\psi_j^*(x_j) = \sum_{x_i} \psi_{ij}^*(x_i, x_j)$. By our earlier arguments (showing the equivalence between message-passing in the junction tree and sum-product message updates), we know that

$$\psi_{ij}^*(x_i, x_j) = \psi_{ij}(x_i, x_j) \prod_{l \in N(i)\setminus\{j\}} M_{l\to i}(x_i) \prod_{l' \in N(j)\setminus\{i\}} M_{l'\to j}(x_j).$$

Hence,

$$\begin{aligned} p(x_j) \;\propto\; \psi_j^*(x_j) &= \sum_{x_i} \psi_{ij}(x_i, x_j) \prod_{l \in N(i)\setminus\{j\}} M_{l\to i}(x_i) \prod_{l' \in N(j)\setminus\{i\}} M_{l'\to j}(x_j) \\ &= M_{i\to j}(x_j) M_{l'\to j}(x_j), \end{aligned}$$

which is exactly the calculation used to obtain single-node marginals in the sum-product algorithm. Thus, we have shown how to deduce the updates for the sum-product algorithm, and proven the correctness of the algorithm via junction tree.

We technically still need to check that node 1 will be assigned the proper marginal, since we combined nodes 1 and 2 before running the sum-product algorithm. But this is easy, since running the sum-product algorithm would give us an edge potential proportional to $p(x_1, x_2)$, and we would then sum over x_2 to obtain the marginal for x_1. This corresponds exactly to the procedure in the junction tree algorithm, where we would take the clique potential $\psi_{12}^*(x_1, x_2)$ after messages are passed, and sum over x_2 to obtain the separator potential for x_1.

Problem 2.3
(10 pts for (a); (b) not graded) *Sum-product algorithm:*

Consider the sum-product algorithm on an undirected tree with potential functions ψ_s and ψ_{st}. Consider any initialization of the messages such that $M_{t\to s}(x_s) > 0$ for all directions $t \to s$ and all states x_s.

(a) Prove by induction that:

(i) the sum-product algorithm, with the flooding schedule, converges in at most diameter of the graph iterations. (Diameter of the graph is the length of the longest path.)

(ii) the message fixed point M^* can be used to compute marginals

$$p(x_s) \;\; \propto \;\; \psi_s(x_s) \prod_{t \in N(s)} M^*_{t \to s}(x_s)$$

for every node of the tree.

Solution:

(i) We induct on the tree diameter D. For $D = 1$, the result is immediate. Consider a graph of diameter D. Let L be the set of all leaf nodes. For each $v \in L$, let w_v be the only element of N_v, which is a singleton set because v is a leaf node. Thus, at the first time step $t = 1$, the (fixed) message v sends to w_v is

$$M^*_{v \to w_v}(x_{w_v}) = \sum_{x_v} \psi_v(x_v) \psi_{v w_v}(x_v, x_{w_v}).$$

At each subsequent time step, the message each leaf sends to its neighbor is constant, since it does not depend on messages from any other nodes. We construct a new undirected graph G' by stripping each of the leaf nodes from the original graph and redefining the compatibility function for each w_v as

$$\psi'_{w_v}(x_{w_v}) = \psi_{w_v}(x_{w_v}) M_{v \to w_v}(x_{w_v}).$$

This new graphical model has diameter at most $D-2$, which follows because the longest path in a tree is between two leaf nodes. Therefore, we can apply the inductive hypothesis to G'. Thus, after at most $D - 2$ time steps, the messages will all converge. Finally, note that replacing the nodes $v \in L$ in G' will not affect any of the fixed-point messages, since the nodes in L always send the same messages to their neighbors. Adding on the first iteration where all leaf nodes send in messages and the last iteration from nodes w_v to the leaf nodes v, we conclude that after D iterations, all messages will converge to a fixed point.

(ii) The easiest way to prove this part is to induct on the number of nodes in the tree. When the tree has one node, the conclusions are trivial. Now let $m \geq 2$, and suppose the conclusions hold for any tree with $m - 1$ nodes. We will show that the results hold for a tree on m nodes, as well.

Renumber the nodes of the tree so that node m is a leaf, with node 1 as its neighbor. In the first iteration, node m sends the message $M^*_{m \to 1}(x_1) = \sum_{x_m} \psi_m(x_m) \psi_{1m}(x_1, x_m)$ to node 1. Note that by marginalizing out x_m and using the factorization over potentials, we have

$$p(x_1, \ldots, x_{m-1}) \; = \; \sum_{x_m} p(x_1, \ldots, x_m) \; \propto \; M^*_{m \to 1}(x_1) \prod_{i=1}^{m-1} \psi_i(x_i) \prod_{(j,k) \in E \setminus \{(1,m)\}} \psi_{jk}(x_j, x_k).$$

In particular, all subsequent messages sent throughout the rest of the graph behave as the sum-product algorithm on the nodes $\{1, \ldots, m-1\}$, with the potential on node 1 replaced by $\psi_1'(x_1) = M^*_{m\to 1}(x_1)\psi_1(x_1)$. By the induction hypothesis, the messages on all edges between nodes $\{1, \ldots, m-1\}$ converges to the proper messages $M^*_{t\to s}(x_s)$, such that the marginal probability of each node is proportional to the product of the node potential and all incoming messages.

It remains to show that the sum-product algorithm computes the correct marginal on node m; i.e., $p(x_m) \propto \psi_m(x_m)M^*_{1\to m}(x_m)$. Since $p(x_1)$ is proportional to the product of $\psi_1(x_1)$ and all incoming messages, and $M_{1\to m}(x_m)$ is a sum of the product of $\psi_1(x_1)\psi_{1m}(x_1, x_m)$ and the incoming messages from nodes in $N(1)\backslash\{m\}$, we have

$$M^*_{1\to m}(x_m) \ \propto \ \sum_{x_1} \frac{p(x_1)}{M^*_{m\to 1}(x_1)}\psi_{1m}(x_1, x_m).$$

Hence, we see that

$$\psi_m(x_m)M^*_{1\to m}(x_m) \ \propto \ \psi_m(x_m)\sum_{x_1} \frac{p(x_1)}{\sum_{x_m'} \psi_m(x_m')\psi_{1m}(x_1, x_m')}\psi_{1m}(x_1, x_m),$$

where we have substituted in the form of $M^*_{m\to 1}(x_1)$. Finally, note that marginalizing out the factorized form of $p(x)$ with potentials (summing over all variables except x_1 and x_m in the numerator, and all variables except x_1 in the denominator), we have

$$p(x_m|x_1) = \frac{p(x_1, x_m)}{p(x_1)} = \frac{\psi_m(x_m)\psi_{1m}(x_1, x_m)}{\sum_{x_m'} \psi_m(x_m')\psi_{1m}(x_1, x_m')}.$$

It follows that

$$\psi_m(x_m)M^*_{1\to m}(x_m) \ \propto \ \sum_{x_1} p(x_1)p(x_m|x_1) = p(x_m),$$

as wanted.

Note: Some people appear to have copied their solution to this problem from a previous year's homework set. Unfortunately, there was a typo in last year's solutions. If we encountered that typo in your homework set, you were given a 0 for this problem.

(b) Implement the sum-product algorithm for an arbitrary tree-structured graph. Please document and hand in your code, including brief descriptions of the data structures that you define and steps involved. Run your algorithm on the tree in Figure 3, using the edge potential functions

$$\psi_{st}(x_s, x_t) = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

for all edges (s, t), and the singleton potential functions

$$\psi_s(x_s) = \begin{bmatrix} 0.7 \\ 0.3 \end{bmatrix} \quad \text{for } s = 1, 3, 5, \text{ and} \quad \psi_s(x_s) = \begin{bmatrix} 0.1 \\ 0.9 \end{bmatrix} \quad \text{otherwise.}$$

Report the values of the single node marginals for all nodes.

Solution: The marginals for each node are

$p(x_1)$	0.591	0.409
$p(x_2)$	0.102	0.898
$p(x_3)$	0.596	0.404
$p(x_4)$	0.065	0.934
$p(x_5)$	0.567	0.433
$p(x_6)$	0.130	0.870

The following is example code in Python:

Listing 1: code.py

```python
#!/usr/bin/env python
from numpy import array, matrix, ones, sum, prod

class Vertex():
    def __init__(self,compat):
        self.compat=compat
    compat = []
    neighbors = [] #list of numbers
    messageout = [] #list of outgoing messages
    messagein = [] #list of incoming messages from neighbors

    def marginal(self):
        a=prod(self.messagein.values(),0)
        a=self.compat*a
        return a/(sum(a))

def compat(i,j):
    #defines the edge potential functions
    if i==j:
        return 1.0
    else:
        return 0.5

def f(x):
    #defines the singleton potential functions
    if x % 2 == 0:
        return array([0.7,0.3])
    else:
        return array([0.1,0.9])

class myV():
    V = [(Vertex(f(x))) for x in range(6)]
    V[0].neighbors=[1,2]; V[1].neighbors=[0,3,4];
    V[2].neighbors=[0,5]; V[3].neighbors=[1];
    V[4].neighbors=[1]; V[5].neighbors=[2];
    for i in range(6):
        for j in V[i].neighbors:
            V[i].messageout = \
                            dict([(x,array([1,1])) \
                                  for x in V[i].neighbors])
            V[i].messagein = \
                           dict([(x,array([1,1])) \
                                 for x in V[i].neighbors])
    def broadcast(self):
        for i in range(6):
            for j in self.V[i].neighbors:
                self.V[i].messageout[j]=array([0,0])
                for my_loop in range(2):
                    self.V[i].messageout[j] = \
                     (self.V[i].messageout[j] +
                     self.V[i].compat[my_loop] *
                     array([compat(my_loop,0),
                            compat(my_loop,1)]) *
                     prod([self.V[i].messagein[x][my_loop]
                           for x in self.V[i].neighbors
                           if x != j]))
    def receive(self):
        for i in range(6):
            for j in self.V[i].neighbors:
                self.V[i].messagein[j] =\
                                     self.V[j].messageout[i]
def main():
    a=myV()
    for k in range(6):
        a.broadcast()
        a.receive()

    for k in range(6):
        print 'Marginal for node', k, 'is', a.V[k].marginal()

if __name__ == '__main__':
    main()
```

Problem 2.4
(not graded) *Computing edge marginals in a tree.*

Consider an undirected tree $T = (V, E)$, where V are the nodes and E are the edges. Recall that we can use the sum-product algorithm to compute all the single node marginals $p(x_i)$.

(a) In this part you are to provide a modification of the sum-product algorithm that will yield edge marginals; i.e., a probability $p(x_i, x_j)$ for $(i, j) \in E$. What is the running time of the

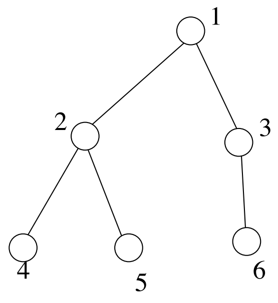


Figure 3: Tree for Problem 2.3(b).

algorithm for computing all edge marginals?

Solution: Only the calculation of the marginals needs to be changed. All other parts, including the message-passing parts and calculating a normalizing constant, remain the same.

Calculate the marginal using

$$p(x_i, x_j) = \frac{1}{Z}\psi_i(x_i)\psi_i(x_j)\psi(x_i, x_j) \prod_{x_k \in N(i)\setminus j} M_{k\to i}(x_i) \prod_{x_k \in N(j)\setminus i} M_{k\to j}(x_j). \qquad (1)$$

We wanted you to think of what the messages in the sum-product algorithm mean, so “Run the junction tree algorithm” was a correct statement but not the solution we wanted. Sum-product potentials/messages are not quite the same as those for a junction tree, so we wanted you to be explicit about the algorithm.

To get a better understanding of the messages, consider the tree rooted at node i. The message $M_{k\to i}(x_i)$ is the “information” from summing over all variables in the subtree rooted at node k. Alternatively, you can view it as the message obtained if you use Eliminate on the nodes in that subtree.

By taking the product of all the messages into i and j excluding the messages between i and j, one “eliminates” all nodes other than i and j. The remaining parts of the marginal probability are the evidence potentials for x_i and x_j and the last edge potential $\psi(x_i, x_j)$.

Note that in this problem, we still want to calculate all edge marginals in one run of the modified sum product algorithm, since we already know how to calculate a single edge marginal by running eliminate.

The running time is the same as the plain sum-product algorithm since the message passing is identical. It is $\mathcal{O}\left(|E|k^2\right) = \mathcal{O}\left(mk^2\right)$, where m is the number of nodes and k is the number of values a variable can take.

(b) Now consider computing arbitrary pairwise marginals in a tree; a probability $p(x_i, x_j)$ for $(i, j) \notin E$. How can such a marginal be computed for a single pair (i, j)? What can you say about the running time of the algorithm?

Solution:

The purpose of this question was to generalize the solution to part (a). We were looking for solutions that made clear how messages computed using the sum-product algorithm could form part of an extended algorithm that produces arbitrary pairwise marginals. The core idea is that the adjacent pair of nodes (i, j) in part (a) is a special case of the set of nodes $P = \{i, v_1, \ldots, v_d, j\}$ that lie on the unique path connecting i and j in part (b). In part (a) we collected all sum-product messages that flow "into" the pair, but excluded any messages flowing between them. For part (b), we will collect all sum-product messages flowing into nodes in P, but exclude any that flow among nodes in P:

$$p(x_P) = p(x_i, x_{v_1}, \ldots, x_{v_d}, x_j) \tag{2}$$

$$= \frac{1}{Z}\left[\prod_{v\in P}\psi_v(x_v)\right]\left[\prod_{\substack{v_1,v_2\in P\\(v_1,v_2)\in E}}\psi_{v_1,v_2}(x_{v_1},x_{v_2})\right]\prod_{v\in P}\left[\prod_{p\in N(v)\setminus P} M_{p\to v}(x_v)\right]. \tag{3}$$

The sum-product messages are a computationally efficient way to marginalize out (or summarize) large branches of the tree which do not directly mediate information between i and j. However, they cannot directly be used to communicate information between i and j. To compute the final pairwise probability the joint needs to be marginalized using the elimination algorithm:

$$p(x_i, x_j) = \sum_{x_{v_1},\ldots,x_{v_d}} p(x_i, x_{v_1}, \ldots, x_{v_d}, x_j). \tag{4}$$

As before, to compute all sum-product messages requires worst-case time $\mathcal{O}\left(mk^2\right)$. Running Eliminate will create elimination cliques of size 3, hence will cost worst-case time $\mathcal{O}\left(mk^3\right)$. The two running times are extremes of two particular instances of the algorithm. For one, if i and j are in fact adjacent, then we reduce to part (a), yielding the same run time as before. On the other hand, if i and j are at opposite ends of a long chain graph, then none of the sum-product messages are strictly necessary and the cost of the elimination algorithm will dominate.

Problem 2.5
(10 pts) *Hammersley-Clifford and Gaussian models:*

Consider a zero-mean Gaussian random vector $(X_1, \ldots, X_N)$ with a strictly positive definite $N \times N$ covariance matrix $\Sigma \succ 0$. For a given undirected graph $G = (V, E)$ with N vertices, suppose that $(X_1, \ldots, X_N)$ obeys all the basic conditional independence properties of the graph G (i.e., one for each vertex cut set).

(a) Show the sparsity pattern of the inverse covariance $\Theta = (\Sigma)^{-1}$ must respect the graph structure (i.e., $\Theta_{ij} = 0$ for all indices i, j such that $(i, j) \notin E$.)

Solution: Fix any pair of indices $(i, j) \notin E$, and let $x_{-\{i,j\}}$ be the subvector of x with components i and j removed. We know that our zero-mean multivariate Gaussian density satisfies

$$\begin{aligned} p(x) &= \frac{1}{Z}\exp(-\frac{1}{2}x^T\Theta x) = \frac{1}{Z}\exp(-\frac{1}{2}\sum_{k,l}x_kx_l\Theta_{kl}) \\ &= \frac{1}{Z}\exp(-\frac{1}{2}x_ix_j\Theta_{ij})\exp(-\frac{1}{2}\sum_{k\neq i}x_kx_j\Theta_{kj})\exp(-\frac{1}{2}\sum_{l\neq j}x_ix_l\Theta_{il}) \\ &= \frac{1}{Z}\exp(-\frac{1}{2}x_ix_j\Theta_{ij})g_i(x_i,x_{-\{i,j\}})g_j(x_j,x_{-\{i,j\}}), \end{aligned} \tag{5}$$

where Z is the normalization constant and g_i, g_j are positive functions.

Since $(i,j) \notin E$ and $(X_1,\ldots,X_N)$ factorizes according to G, we have $X_i \perp X_j \mid X_{-\{i,j\}}$. By Problem 1.3(d) from Homework 1, this implies

$$p(x) = f_i(x_i, x_{-\{i,j\}})f_j(x_j, x_{-\{i,j\}}) \tag{6}$$

for positive functions f_i, f_j.

Combining equations (5) and (6), we have

$$\exp(-\frac{1}{2}x_ix_j\Theta_{ij}) = \frac{Zf_i(x_i,x_{-\{i,j\}})}{g_i(x_i,x_{-\{i,j\}})}\frac{f_j(x_j,x_{-\{i,j\}})}{g_j(x_j,x_{-\{i,j\}})} = h_i(x_i)h_j(x_j),$$

for positive functions h_i, h_j (the second equality follows since the left-hand side does not depend on $x_{-\{i,j\}}$—for instance, consider fixing the values of $x_{-\{i,j\}}$, which will not affect the LHS of the equality above). When $x_i = 0$, we see $h_j(x_j) = \frac{1}{h_i(0)}$ for all x_j, implying that $h_j(x_j)$ is a constant function of x_j. Similarly, $h_i(x_i)$ is constant, and so $\exp(-\frac{1}{2}x_ix_j\Theta_{ij})$ is a constant function of x_i, x_j. Thus $\exp(-\frac{1}{2}x_ix_j\Theta_{ij}) = \exp(-\frac{1}{2}(0)(0)\Theta_{ij}) = 1$, and $\Theta_{ij} = 0$.

(b) Interpret this sparsity relation in terms of cut sets and conditional independence.

Solution: By the previous part, we see that $\Theta_{ij} = 0$ whenever $(i,j) \neq E$. In terms of conditional independence, this means $\Theta_{ij} = 0$ if and only if there is a cut set that separates X_i and X_j. For examples of such cut sets, $N(X_i)$ or $N(X_j)$ or $V\backslash\{X_i, X_j\}$ all serve as cut sets.