Lecture 15:

Belief Propagation on Trees

Sam Roweis

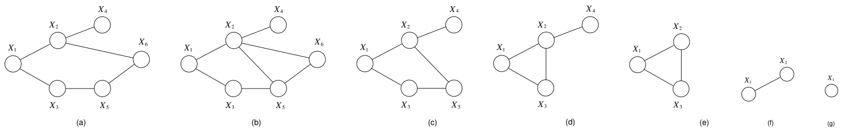
March 3, 2003

Multiple Queries

- The Elimination algorithm we described was query based: given the single node marginal to compute (the last item in the ordering), it efficiently summed out or conditioned on all other variables.
- But what if we want to do multiple inferences?
  For example, during learning, constraint satisfaction, planning.

Reminder: Elimination for Inference

- We want to be able to condition on some "evidence" $\mathbf{x}_E$ (observed nodes) and compute the posterior probabilities of some "query" nodes $\mathbf{x}_F$ while marginalizing out "nuisance" nodes $\mathbf{x}_R$.
- For a single node posterior (i.e. $\mathbf{x}_F$ is a single node), there was a simple, efficient algorithm based on eliminating nodes.
- Notation: $\bar{\mathbf{x}}_i$ is the value of evidence node x_i.
- The algorithm, called *elimination*, required a *node ordering* to be given, which told it which order to do the summations in.

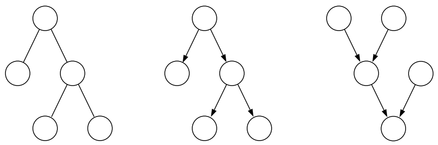


Efficient Multi-Elimination

- We could run Elimination once for each marginal, but this would be *extremely* inefficient since most of the calculations would be duplicated.
- We want an algorithm that reuses work efficiently to compute all marginals (or pairwise marginals) given evidence
- We do not want to duplicate work unecessarily.
  We want to reuse calculations as best as possible.
- This needs:
  1) A plan for which intermediate factors to compute in what order.
  2) Some storage for these intermediate factors.

Tree-Structured Graphical Models

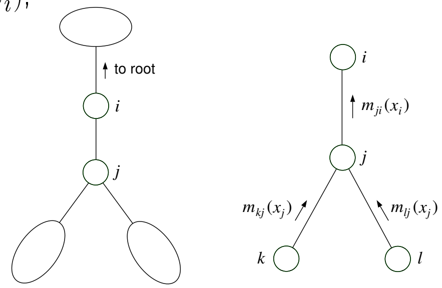
- For now, we will focus on tree-structured graphical models.
- Trees are an important class; e.g. Hidden Markov Models and continuous State Space Models are trees.
- Exact inference on trees is the basis for the *junction tree algorithm* which solves the general exact inference problem for directed acyclic graphs and for many *approximate* algorithms which can work on intractable or cyclic graphs.
- Directed and undirected trees make exactly same conditional independence assumptions, so we cover them together.



(a) (b) (c)

Elimination on Trees

- Now consider eliminating node j which is followed by i in the order.
- Which nodes appear in the potential created after summing over j?
  - nothing in the subtree below j (already eliminated)
  - nothing from other subtrees, since the graph is a tree
  - only i, from ψ_{ij} which relates i and j
- Call the factor that is created $m_{ji}(x_i)$, and think of it as a *message* that j passes to i when j is eliminated.
- This message is created by summing over j the product of all earlier messages $m_{kj}(x_j)$ sent to j as well as $\psi_j^E(x_j)$ if j is an evidence node.



(a) (b)

Elimination on Trees

- Recall basic structure of Eliminate:
  1. Convert directed graph to undirected by moralization.
  2. Chose elimination ordering with query node last.
  3. Place all potentials on active list.
  4. Eliminate nodes by removing all relevant potentials, taking product, summing out node and placing resulting factor back onto potential list.
- What happens when the original graph is a tree?
  1. No moralization is necessary.
  2. There is a natural elimination ordering with query node as root.
  3. All subtrees with no evidence nodes can be ignored (since they will leave a potential of unity once they are eliminated).

Eliminate = Message Passing

- On a tree, Eliminate can be thought of as passing messages up to the query node at the root from the other nodes at the leaves or interior. Since we ignore subtrees with no evidence, observed (evidence) nodes at always at the leaves.
- The message $m_{ji}(x_i)$ is created when we sum over x_j

$$m_{ji}(x_i) = \sum_{x_j} \left(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in c(j)} m_{kj}(x_j) \right)$$

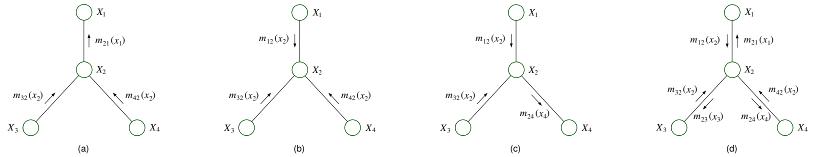
- At the final node x_f, we obtain the answer:

$$p(x_f|\bar{\mathbf{x}}_E) \propto \psi^E(x_f) \prod_{k \in c(f)} m_{kf}(x_f)$$

- If j is an evidence node, $\psi^E(x_j) = \delta(x_j, \bar{x}_j)$, else $\psi^E(x_j) = 1$.
- If j is a leaf node in the ordering, $c(j)$ is empty, otherwise $c(j)$ are the children of j in the ordering.

Messages are Reused in MultiElimination

- Consider querying x_1, x_2, x_3 and x_4 in the graph below.
- The messages needed for x_1, x_2, x_4 individually are shown (a-c).
- Also shown in (d) is the set of messages needed to compute all possible marginals over single query nodes.



- Key insight: even though the naive approach (rerun Elimination) needs to compute N^2 messages to find marginals for all N query nodes, there are only $2N$ possible messages.
- We can compute all possible messages in only double the amount of work it takes to do one query.
- Then we take the product of relevant messages to get marginals.

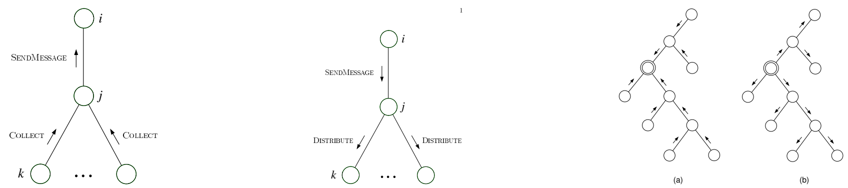
Belief Propagation (Sum-Product) Algorithm

- Choose a root node arbitrarily.
- If j is an evidence node, $\psi^E(x_j) = \delta(x_j, \bar{x}_j)$, else $\psi^E(x_j) = 1$.
- Pass messages from leaves up to root and then back down using:

$$m_{ji}(x_i) = \sum_{x_j} \left(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in c(j)} m_{kj}(x_j) \right)$$

- Given messages, compute marginals using:

$$p(x_i|\bar{\mathbf{x}}_E) \propto \psi^E(x_i) \prod_{k \in c(i)} m_{ki}(x_i)$$

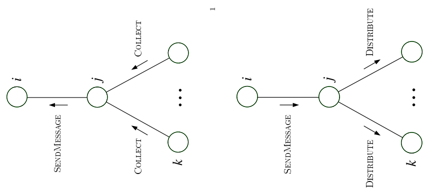


Computing All Possible Messages

- How can we compute all possible messages efficiently?
- Idea: respect the following Message-Passing-Protocol: *A node can send a message to a neighbour only when it has received messages from all its other neighbours.*
- Protocol is realizable: designate one node arbitrarily as the root. Collect messages inward to root then distribute back out to leaves.
- Once we have the messages, we can compute marginals using:

$$p(x_i|\bar{\mathbf{x}}_E) \propto \psi^E(x_i) \prod_{k \in c(i)} m_{ki}(x_i)$$

- Remember that the directed tree on which we pass messages might not be same directed tree we started with.
- We can also consider "synchronous" or "asynchronous" message passing nodes that respect the protocol but don't use the Collect-Distribute schedule above. (Must prove this terminates.)




```
Sum-Product(T, E)
    Evidence(E)
    f = ChooseRoot(V)
    for e ∈ N(f)
        Collect(f, e)
    for e ∈ N(f)
        Distribute(f, e)
    for i ∈ V
        ComputeMarginal(i)

Evidence(E)
    for i ∈ E
        ψ^E(x_i) = ψ(x_i)δ(x_i, x̄_i)
    for i ∉ E
        ψ^E(x_i) = ψ(x_i)

Collect(i, j)
    for k ∈ N(j)\i
        Collect(j, k)
    SendMessage(j, i)

Distribute(i, j)
    SendMessage(i, j)
    for k ∈ N(j)\i
        Distribute(j, k)

SendMessage(j, i)
    m_ji(x_i) = Σ_{x_j} (ψ^E(x_j)ψ(x_i, x_j) Π_{k∈N(j)\i} m_kj(x_j))

ComputeMarginal(i)
    p(x_i) ∝ ψ^E(x_i) Π_{j∈N(i)} m_ji(x_i)
```

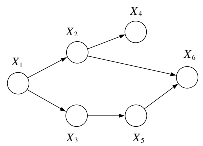
A sequential implementation of the Sum-Product algorithm for a tree $\mathcal{T}(\mathcal{V}, \mathcal{E})$. The algorithm works for any choice of root node, and thus we have left ChooseRoot unspecified. A call to Collect causes messages to flow inward from the leaves to the root. A subsequent call to Distribute causes messages to flow outward from the root to the leaves. After these calls have returned, the singleton marginals can be computed locally at each node.

Maximizing instead of Summing

- Elimination and Belief Propagation both summed over all possible values of the marginal (non-query, non-evidence) nodes to get a marginal probability.
- What if we wanted to *maximize* over the non-query, non-evidence nodes to find the probabilty of the single best setting consistent with any query and evidence?

$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{\mathbf{x}_1} \max_{\mathbf{x}_2} \max_{\mathbf{x}_3} \max_{\mathbf{x}_4} \max_{\mathbf{x}_5} p(\mathbf{x}_1) p(\mathbf{x}_2|\mathbf{x}_1) p(\mathbf{x}_3|\mathbf{x}_1) p(\mathbf{x}_4|\mathbf{x}_2) p(\mathbf{x}_5|\mathbf{x}_3) p(\mathbf{x}_6|\mathbf{x}_2, \mathbf{x}_5)$$
$$= \max_{\mathbf{x}_1} p(\mathbf{x}_1) \max_{\mathbf{x}_2} p(\mathbf{x}_2|\mathbf{x}_1) \max_{\mathbf{x}_3} p(\mathbf{x}_3|\mathbf{x}_1) \max_{\mathbf{x}_4} p(\mathbf{x}_4|\mathbf{x}_2) \max_{\mathbf{x}_5} p(\mathbf{x}_5|\mathbf{x}_3) p(\mathbf{x}_6|\mathbf{x}_2, \mathbf{x}_5)$$

- This is known as the *maximum a-posteriori* or MAP configuration.
- It turns out that (on trees), we can use an algorithm exactly like belief-propagation to solve this problem.



Max-Product Algorithm

- Choose a root node arbitrarily.
- If j is an evidence node, $\psi^E(x_j) = \delta(x_j, \bar{x}_j)$, else $\psi^E(x_j) = 1$.
- Pass messages from leaves up to root using:

$$m_{ji}^{max}(x_i) = \max_{x_j} \left(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in c(j)} m_{kj}^{max}(x_j) \right)$$

- Remember which choice of $x_j = x_j^*$ yielded maximum.
- Given messages, compute max value using any node i:

$$\max_{\mathbf{x}} p^E(\mathbf{x}|E) = \max_{x_i} \left(\psi^E(x_i) \prod_{k \in c(i)} m_{ki}(x_i) \right)$$

- Retrace steps from root back to leaves recalling best x_j^* to get the maximizing argument (configuration) $\mathbf{x}^*$.

Sum-Product, Max-Product and Semirings

- Why can we use the same trick for MAP as for marginals?
- Because multiplication distributes over $\max$ as well as sum:

$$\max(ab, ac) = a \ \max(b, c)$$

- Formally, both the "sum-product" and "max-product" pair are *commutative semirings*.
- It turns out that the "max-sum" pair is also a semiring:

$$\max(a + b, a + c) = a + \max(b, c)$$

which means we can do MAP computations in the log domain:

$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{\mathbf{x}} \prod_i p(x_i|x_{\pi_i}) = \max_{\mathbf{x}} \log p(\mathbf{x}) = \max_{\mathbf{x}} \sum_i \log p(x_i|x_{\pi_i})$$