CS281A/Stat241A: Statistical Learning Theory

Sum-Product Algorithm

Lecturer: Michael I. Jordan *Scribes: Lara Dolecek and Raul Etkin*

Overview

- *Elimination*: works for both directed and undirected graphs, but only answers queries about one node.
- *Sum-Product algorithm* (Belief Propagation): allows to answer queries about arbitrary sets of nodes but it only works on directed and undirected trees.

Complexity of *Eliminate*

$$m_i(x_{S_i}) = \sum_{x_i} \phi(x_{T_i})$$

This is the only algebraic step, so it determines the complexity of the algorithm. The complexity of the above sum is exponential in $|T_i|$, where $T_i = S_i \bigcup \{i\}$. The complexity of the algorithm is dominated by $\max_i |T_i|$.

UndirectedGraphEliminate

This algorithm computes the sets T_i without explicitly computing the sums.

The sum over X_i involves potentials that depend on j, k, and l.

$$\sum_{x_i} \psi(x_i, x_j)\psi(x_i, x_l)\psi(x_i, x_k) = f(x_j, x_l, x_k)$$

When eliminating node i we connect all neighbors of i to represent this dependence, and we get a clique whose size is $|T_i|$. The potentials on the active list which reference X_i are grouped together.

For the graph in Figure 1, when the node i is about to be eliminated, all its neighbors, namely {j,k,l}, are connected. The result of this operation is shown in Figure 2.

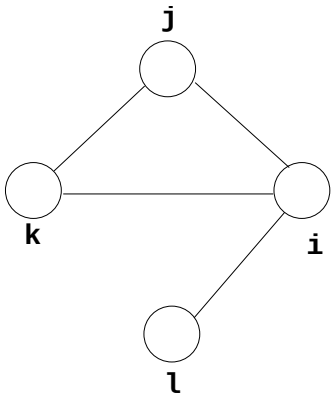

Figure 1: Elimination of node i

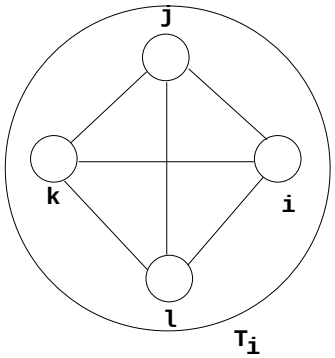

Figure 2: Elimination clique i

Treewidth of a graph

Definition: *Treewidth* $= \min_I$ (max clique size found by UndirectedGraphEliminate for an ordering I) $- 1$

The "-1" in the definition above is simply a convention so that the treewidth of a tree has value 1.

Figures 3 and 4 show how, depending on the choice of the ordering, the resulting cliques can be smaller (Figure 3) or much bigger (Figure 4).

Theorem: The complexity of *Eliminate* is exponential in (*Treewidth* $+1$), both in time and space.

Computing Treewidth is NP-hard.

We can run *Eliminate* for a particular ordering to obtain an upper bound on Treewidth. Note that UGE runs in polynomial time.

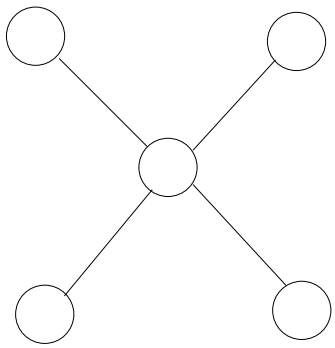

Figure 3: Good elimination ordering

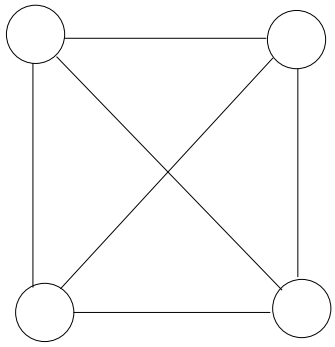

Figure 4: Dumb elimination ordering

A model for representing speech or video is shown in Figure 5. In this case, the treewidth is 2. A similar example is given in Figure 6. For this graph, the treewidth is 3.

Generally, for a lattice of size d, the cost of *Eliminate* is exponential in d.

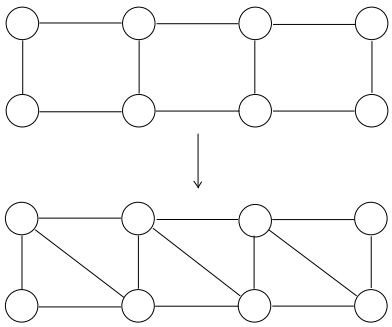

Figure 5: Lattice graph of size 2

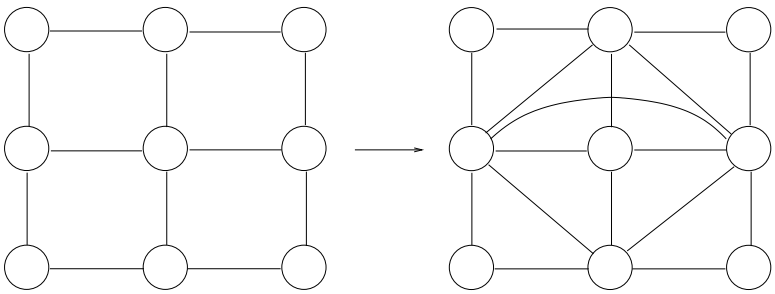

Figure 6: Lattice graph of size 3

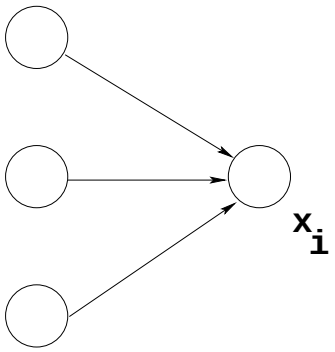

Figure 7: Directed graph

Directed Graphs

For the variable X_i in Figure 7, we write $P(X_i|X_{\pi_i})$ where π_i are the parents of the node i, i.e. the three nodes on the left.

Definition: *Moralization*: For all nodes in the directed graph, connect the parents, and drop the orientation of the arrows in the resulting graph.

The result of the moralization applied to the graph from Figure 7, is shown in Figure 8.

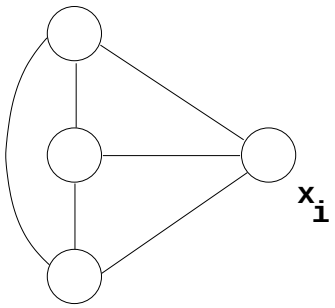

Figure 8: Moralized graph

The moralized graph is the cover of the original graph in the sense that it includes all of the original edges (without the arrows) plus some additional ones.

The potential functions on the moralized graph are of the form $\mathrm{P}(X_i|X_{\pi_i})$.

UGE can be applied to the moralized graph.

Complexity of inference on directed graphs

The complexity is governed by two operations, moralization and UGE on the moralized graph.

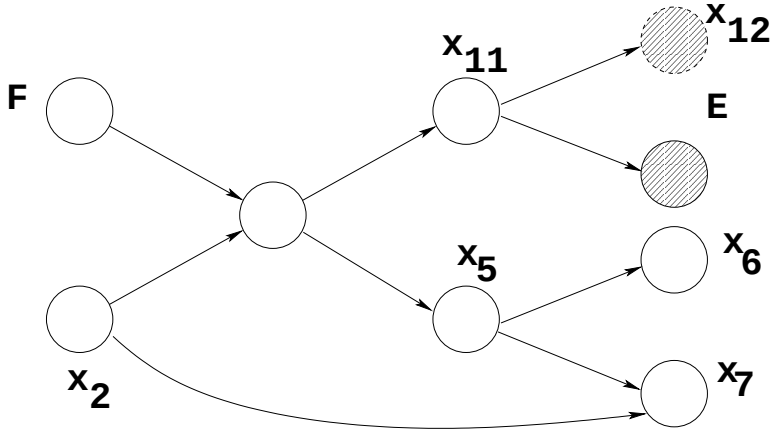

Figure 9: Inference on a directed graph

Consider a directed graph, shown in Figure 9. The evidence nodes are shaded and labeled with E, and the node whose marginal we are interested is labeled F. For this graph, we have the following:

$\sum_{X_6} P(X_6|X_5) = 1$

$\sum_{X_7} P(X_7|X_5, X_2) = 1$

$\sum_{X_{12}} P(X_{12}|X_{11})\delta(X_{12}, \overline{X_{12}}) = P(\overline{X_{12}}|X_{11})$

Sum Product Algorithm

Sum Product (S-P) algorithm is generally used for the inference on trees, but can be also used for the approximate inference on general graphs. The application of the latter is in the Error Control Coding.

The goal of the algorithm is to compute ALL singleton marginals, $\mathrm{P}(X_i|X_E)$ for all i $\in V \backslash E$.

Some applications of this algorithm are computing faults from an observed set of symbols, as shown in Figure 10, or in a Hidden Markov Model (and Kalman filtering), shown in Figure 11.

The elimination algorithm computes the same terms over and over again. For example, in Kalman filtering it would repeat most of the same calculations for computing marginals of two adjacent nodes. The Sum Product algorithm adopts a dynamic programming approach and thus reuses most calculations.

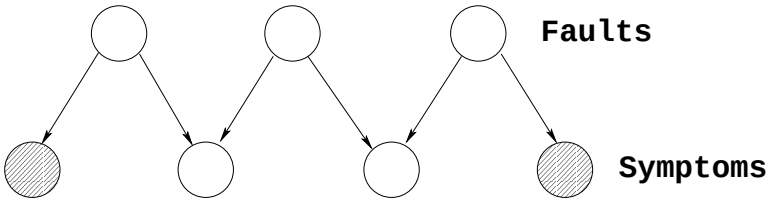

Figure 10: Faults and symptoms

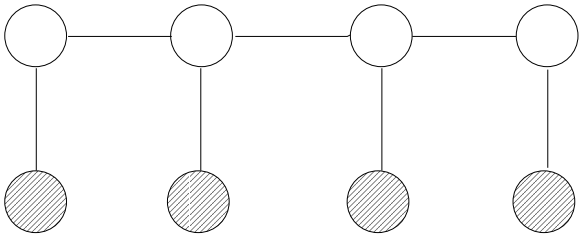

Figure 11: Kalman filtering

Undirected Trees

Let us consider the undirected tree first. One such example is shown in Figure 12.

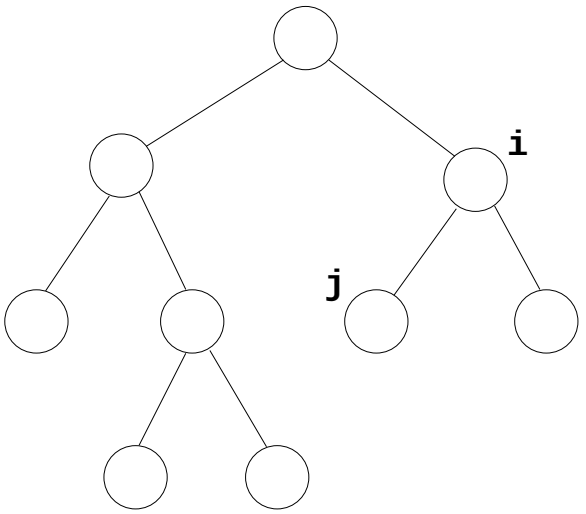

Figure 12: Undirected tree

Cliques can be of size 1, i.e. representing singletons of the type $\{X_i\}$, or of size 2, representing $\{X_i, X_j\}$. Potentials are then $\psi\{X_i\}$ and $\psi\{X_i, X_j\}$. Those over non-maximal cliques are included for convenience.

The joint probability function is:

$P(X) = \frac{1}{Z} \prod_{i \in V} \psi(X_i) \prod_{(i,j) \in E} \psi(X_i, X_j)$.

Including the evidence, the last expression becomes

$P^{EV}(X) = \frac{1}{Z} \prod_{i \in V} \psi(X_i) \prod_{(i,j) \in E} \psi(X_i, X_j) \prod_{i \in EV} \delta(X_i, \overline{X_i})$.

This can also be written as

$P^{EV}(X) = \frac{1}{Z} \prod_{i \in V} \psi^{EV}(X_i) \prod_{(i,j) \in E} \psi(X_i, X_j)$

where $\psi^{EV}(X_i)$ is defined as

$$\psi^{EV}(X_i) = \begin{cases} \psi(X_i)\delta(X_i, \overline{X_i}) & \text{if } i \in EV \\ \psi(X_i) & \text{if } i \notin EV \end{cases}$$

Directed Trees

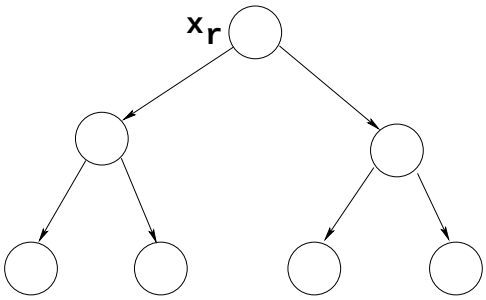

Figure 13: Directed tree

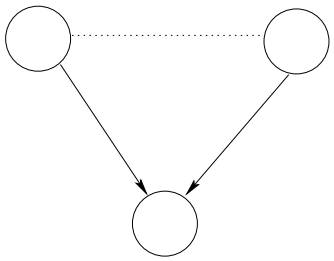

Figure 14: A graph which is not a directed tree

Directed trees are defined as directed graphs whose moralized graph is an undirected tree. An example of a directed tree is shown in Figure 13, and an example of a graph which is not a directed tree is shown in Figure 14. For the latter, a node has two parents, a property not allowed in directed trees.

For directed trees we have

$P(X) = P(X_r) \prod_{i \in V \setminus r} P(X_i | X_j)$

where r denotes the root node. For every other node i, there is a single node j which is its parent.

Belief Propagation Algorithm

Message m_{ji} passed from the node j to the node i is

$m_{ji}(X_i) = \sum_{x_j} \psi(X_i, X_j) \psi^{EV}(X_j) \prod_{k \in N(j) \setminus \{i\}} m_{kj}(X_j)$

This step is shown in Figure 15. Note that the update is between two adjacent nodes, so the algebraic complexity of the message passing is quadratic in the number of values a random variable can take.

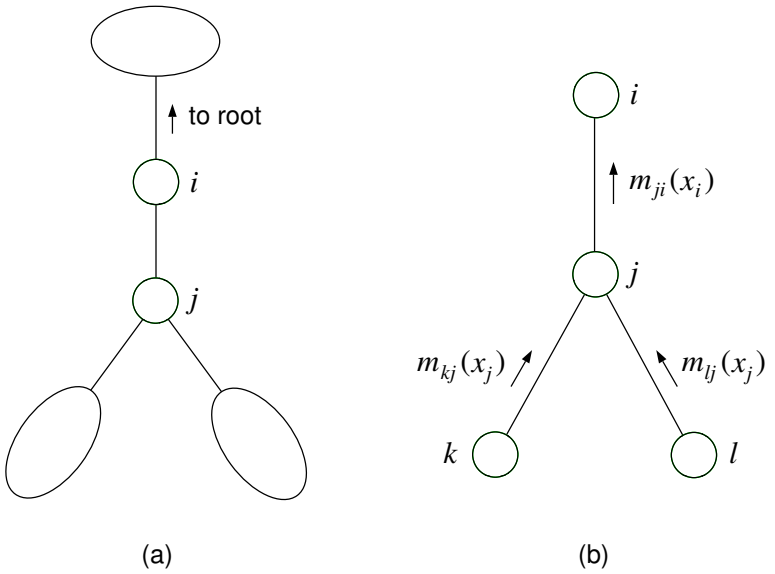

Figure 15: Message update between nodes j and i

Protocol

Send a message from a node to a neighboring node when the former node has received messages from all other neighbors.

The process starts at the leaves, and the message passing can be done in parallel.

Every node will be visited twice and there will be no blocking.

This process is shown in Figure 16, where the enumeration 1 through 5 denotes when is the massage passed between two nodes.

Calculation of Marginals

$P(X_i|X_{EV}) \propto \psi^{EV}(X_i) \prod_{j \in N(i)} m_{ji}(X_i)$

If the graph is not a tree, it might still be possible to use this algorithm because from the point of view of a single node, the graph looks like a tree. In this scenario, all messages are initialized to 1 and then they are passed on from one node to its neighbor. It can, however, happen that the algorithm does not converge or converges to a wrong limit.

Derivation of Sum Product Algorithm

Let us consider *Eliminate*.

Choose an ordering I which respects graph topology, that is, the descendants of a node appear before the

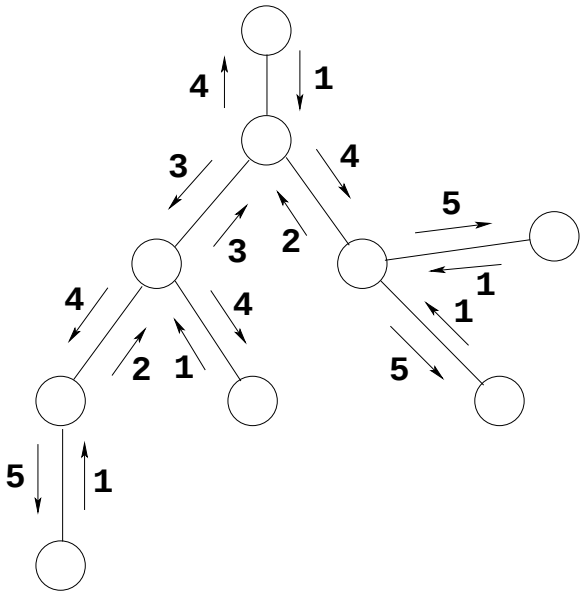

Figure 16: Message passing over the entire graph

node itself in the ordering I.

Consider the active list at the moment we eliminate node X_j. At this point all descendants of j have been eliminated. The list includes the following:

$\psi^{EV}(X_j), \psi(X_i, X_j), m_k(X_j) \forall k \in N(j) \backslash \{i\}$

We add the extra index in the messages in the general algorithm because we can have messages propagating in both directions.

This shows that Sum Product algorithm is just the application of *Eliminate* on a single node of a tree.

This is graphically shown in Figure 17.

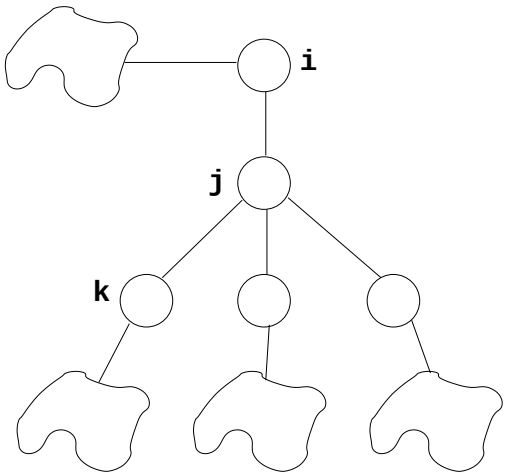

Figure 17: Step in S-P algorithm