

# CSE 6740 Lecture 26

## How Do I Evaluate Deeply-Nested Sums? (Graphical Model Inference)

Alexander Gray

agray@cc.gatech.edu

Georgia Institute of Technology

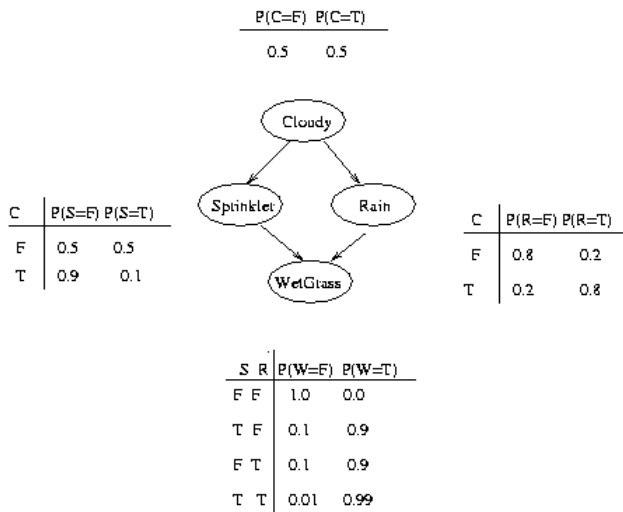
- ① Graphical Model Computations
- ② Exact Inference Algorithms
- ③ Approximate Inference Algorithms

# Graphical Model Computations

What we want to compute with graphical models.

# Sprinkler Example

Recall the sprinkler example:



# Conditional Probability Inference

“Inference” is a general statistical term. In the context of graphical models it means finding out the probability distribution for a variable of interest given that some of the other variables have fixed values.

For example, suppose we observe the fact that the grass is wet. There are two possible causes for this: either it is raining, or the sprinkler is on. Which is more likely?

# Conditional Probability Inference

We can use Bayes' rule to compute the posterior probability of each explanation.

$$\begin{aligned}\mathbb{P}(S = 1|W = 1) &= \frac{\mathbb{P}(S = 1, W = 1)}{\mathbb{P}(W = 1)} && (1) \\ &= \frac{\sum_{c,r} \mathbb{P}(C = c, S = 1, R = r, W = 1)}{\mathbb{P}(W = 1)} = 0.43\end{aligned}$$

$$\begin{aligned}\mathbb{P}(R = 1|W = 1) &= \frac{\mathbb{P}(R = 1, W = 1)}{\mathbb{P}(W = 1)} && (3) \\ &= \frac{\sum_{c,s} \mathbb{P}(C = c, S = s, R = 1, W = 1)}{\mathbb{P}(W = 1)} = 0.41\end{aligned}$$

# Conditional Probability Inference

where

$$\mathbb{P}(W = 1) = \sum_{c,r,s} \mathbb{P}(C = c, S = s, R = r, W = 1) = 0.65 \quad (5)$$

is the normalizing constant.

So we see that it is more likely that the grass is wet because it is raining.

# Graphical Model Computations

Let  $(E, Q)$  be a partitioning of the variables (“evidence” and “query” variables). Two main quantities we desire are:

- *Marginal probabilities:*

$$\mathbb{P}(Q = q) = \sum_e \mathbb{P}(Q = q, E = e) \quad (6)$$

or  $\mathbb{P}(E = e) = \sum_q \mathbb{P}(Q = q, E = e)$ .

- *Maximum a posteriori (MAP) probabilities:*

$$\mathbb{P}^*(Q = q) = \max_e \mathbb{P}(Q = q, E = e). \quad (7)$$

# Graphical Model Computations

From these basic quantities we can obtain other quantities such as *conditional probabilities*:

$$\mathbb{P}(Q = q|E = e) = \frac{\mathbb{P}(Q = q, E = e)}{\mathbb{P}(E = e)} = \frac{\mathbb{P}(Q = q, E = e)}{\sum_q \mathbb{P}(Q = q, E = e)}. \quad (8)$$

In general multiple marginalizations (summations) must be performed. For example if  $(E, Q, H)$  (“evidence”, “query”, and “hidden”) is a partitioning of the variables,

$$\mathbb{P}(Q = q|E = e) = \frac{\mathbb{P}(Q = q, E = e)}{\sum_q \mathbb{P}(Q = q, E = e)} \quad (9)$$

$$= \frac{\sum_h \mathbb{P}(Q = q, E = e, H = h)}{\sum_q \sum_h \mathbb{P}(Q = q, E = e, H = h)}. \quad (10)$$

# Graphical Model Computations

For a directed graph, we can represent the joint probability

$$\mathbb{P}(X_1, \dots, X_D) = \prod_i \mathbb{P}(X_i | \Pi(X_i)) \quad (11)$$

where  $\Pi(X)$  denotes the set of parents of  $X$  and  $\mathbb{P}(X_i | \Pi(X_i))$  is the local conditional probability associated with node  $X_i$ .

# Graphical Model Computations

For an undirected graph, the basic subsets are cliques of the graph – subsets of nodes which are completely connected. For a given clique, if  $\psi_C(X_C)$  denotes a general potential function (a function which assigns a positive real number to each configuration  $X_C$ ), then

$$\mathbb{P}(X_1, \dots, X_D) = \frac{1}{Z} \prod_C \psi_C(X_C) \quad (12)$$

where  $Z$  is a normalizing factor.

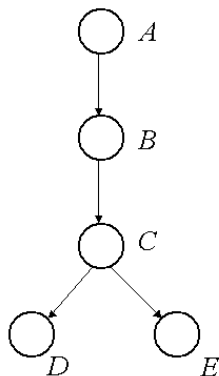
# Graphical Model Computations

It would be nice if we could treat both types of graphs in the same way. Note that the directed graph equation is a special case of the undirected graph equation, except that the parent sets are not necessarily cliques. If we modified the original graph a bit, we could make it a special case on the modified graph.

The *moral graph* is an undirected graph obtained by connecting all the parents of each node in the original graph and removing the directions.

# Elimination

The core algorithmic idea is that of *elimination*. Let's do an example:



# Elimination

We'll develop an algorithm for computing a single marginal probability  $\mathbb{P}(E = e)$ , for which we'll use the shorthand  $P(e)$ . Starting with the joint probability we'll have to, in some fashion, sum over every variable other than  $E$ :

$$P(e) = \sum_a \sum_b \sum_c \sum_d P(a, b, c, d, e). \quad (13)$$

We must pick an order in which to sum - we'll pick the order  $(a, b, d, c)$ . We'll also write the joint in the compact form afforded by knowledge of the conditional independencies specified by the graph:

$$P(e) = \sum_c \sum_d \sum_b \sum_a P(a)P(b|a)P(c|b)p(d|c)p(e|c). \quad (14)$$

# Elimination

Now let's rearrange this summation so that we push in sums as far rightward as possible, based on the variables they depend on:

$$P(e) = \sum_c p(e|c) \sum_d p(d|c) \sum_b p(c|b) \sum_a p(a)p(b|a). \quad (15)$$

Each time we finish a summation over a variable, we end up with an expression, which we can think of as a “message” which we pass to the next summation:

$$P(e) = \sum_c p(e|c) \sum_d p(d|c) \sum_b p(c|b) m_{ab}(b). \quad (16)$$

# Elimination

Continuing on in this fashion,

$$p(e) = \sum_c p(e|c) \sum_d p(d|c) \sum_b p(c|b) m_{ab}(b) \quad (17)$$

$$= \sum_c p(e|c) \sum_d p(d|c) m_{bc}(c) \quad (18)$$

$$= \sum_c p(e|c) m_{bc}(c) \sum_d p(d|c) \quad (19)$$

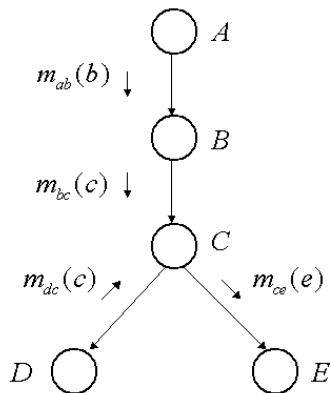
$$= \sum_c p(e|c) m_{bc}(c) m_{dc}(c) \quad (20)$$

$$= m_{ce}(e). \quad (21)$$

The final expression is a function of  $e$  only and is the desired marginal probability.

# Elimination

Here are the messages created during this run of the elimination algorithm:



# Sum-Product Algorithm

In general we want to compute marginal probabilities for more than one query variable, for example, both  $p(e)$  and  $p(d)$ . We could run the elimination algorithm separately for each variable, but many intermediate computations would be duplicated.

The *sum-product algorithm* can be thought of as the multi-query version of elimination, which reuses intermediate terms efficiently. It is also called *belief propagation*. We're going to now assume the graph is a tree.

# Sum-Product Algorithm

The intermediate terms are the “messages”. Rather than viewing inference as an elimination process, based on a global ordering, because we are now interested in multiple query variables, we instead view inference from each variable’s local point of view, *i.e.* the messages it gets in and sends out.

The key operation of summing a product can be written as

$$m_{xy}(y) = \sum_x \psi_{xy}(x, y) \prod_{z \in N(x) \setminus y} m_{zx}(x) \quad (22)$$

where  $N(x)$  is the set of neighbors of  $X$  and  $N(x) \setminus y$  means exclusion of  $Y$ . So summing over  $x$  creates a message  $m_{xy}(y)$  which is sent to node  $Y$ .

# Sum-Product Algorithm

A node can send a message to a neighboring node once it has received messages from all of its other neighbors. The marginal probability at a node is given by the product of all incoming messages:

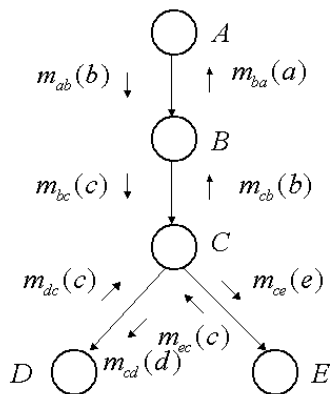
$$p(x) \propto \prod_{z \in N(x) \setminus y} m_{zx}(x) \quad (23)$$

Note that the algorithm avoids double-counting: the message  $m_{xy}(y)$  from  $x$  to  $y$  is not included in the product that node  $y$  forms in computing a message to send back to node  $x$ .

Note that the sum-product algorithm is only correct for trees, not general graphs.

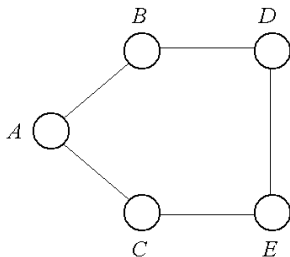
# Sum-Product Algorithm

Here are all the messages sent during the sum-product algorithm for our example:



# Junction-tree Algorithm

Now let's develop a generalization of the sum-product algorithm, the *junction-tree algorithm*, which is correct for arbitrary graphs. Consider an example which is not a tree:



# Junction-tree Algorithm

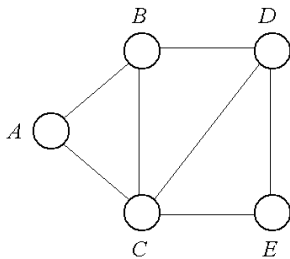
Let's go back to the elimination algorithm, which is correct for arbitrary graphs. Suppose we want the conditional probability  $p(a)$ , and we use the elimination ordering  $(e, d, c, b)$ . At the first step, in which we sum over  $e$ , we remove the potentials  $\psi_{ce}(c, e)$  and  $\psi_{de}(d, e)$  from the active list and form the sum

$$m_{cb}(c, d) = \sum_e \psi_{ce}(c, e) \psi_{de}(d, e). \quad (24)$$

This intermediate term is a function of  $c$  and  $d$ , effectively linking  $C$  and  $D$ , variables which were not linked in the original graph. A similar thing will happen in the next step, linking  $B$  and  $C$ .

# Junction-tree Algorithm

A graphical record of the dependencies induced during the run of the elimination algorithm is shown here:



# Junction-tree Algorithm

We can also think of this process as creating a graph as follows: whenever we eliminate a node, we remove it and link its remaining neighbors. For example when  $E$  is removed, nodes  $C$  and  $D$  are linked.

Another way to do the same thing is to imagine that we had the final graph to start with, and we perform elimination using cliques of that graph as if they were nodes. The cliques are  $C_1 = \{A, B, C\}$ ,  $C_2 = \{B, C, D\}$  and  $C_3 = \{C, D, E\}$  and we have the potentials

$$\psi_{C_1}(a, b, c) = \psi_{ab}(a, b)\psi_{ac}(a, c) \quad (25)$$

$$\psi_{C_2}(b, c, d) = \psi_{bd}(b, d) \quad (26)$$

$$\psi_{C_3}(c, d, e) = \psi_{ce}(c, e)\psi_{de}(d, e), \quad (27)$$

yielding the same product of potentials as before.

# Junction-tree Algorithm

This leads to the following procedure for a general directed graph:

- ① Transform the original graph into a *junction-tree*:
  - ① Moralize the graph
  - ② Triangulate the graph
  - ③ Compute a maximal spanning tree of the cliques (with weights given by the cardinalities of the intersections between cliques)
- ② Perform sum-product on the junction-tree

# Junction-tree Algorithm

All of these algorithms can be modified for the case of computing the most likely configuration of variables (the MAP problem) by simply replacing the sum operator with the max operator.

The computational cost of the junction-tree algorithm is exponential in the maximum number of variables in a clique. So lots of dependencies makes this slow.

Next we'll develop approximate procedures for handling this case.

# Approximate Inference Algorithms

Basically: Variational methods.

# Graphical Model Computations

We want to compute quantities such as *conditional probabilities*. If  $(E, H)$  (“evidence” and “hidden”) is a partitioning of the variables, we want

$$\mathbb{P}(H = h|E = e) = \frac{\mathbb{P}(H = h, E = e)}{\mathbb{P}(E = e)} = \frac{\mathbb{P}(H = h, E = e)}{\sum_h \mathbb{P}(H = h, E = e)} \quad (28)$$

which we'll write in a shorthand:

$$\mathbb{P}(H|E) = \frac{\mathbb{P}(H, E)}{\mathbb{P}(E)} = \frac{\mathbb{P}(H, E)}{\sum_H \mathbb{P}(H, E)} \quad (29)$$

where  $H$  and  $E$  can represent subsets of variables.

# Bounds in Graphical Models

Suppose we can bound each variable's local CPD (conditional probability distribution). Then we can combine these bounds to get bounds on global quantities such as the joint probability distribution:

$$\mathbb{P}(X_1, \dots, X_D) = \prod_i \mathbb{P}(X_i | \Pi(X_i)) \quad (30)$$

$$\leq \prod_i P^u(X_i | \Pi(X_i), \lambda_i^u) \quad (31)$$

where  $\Pi(X_i)$  denotes the set of parents of  $X_i$  and the  $\lambda$  variables are parameters of the bounding functions.

# Bounds in Graphical Models

We'll use bounds that hold for any setting of these parameters. Thus they must also hold for any subset of variables whenever some other subset is held fixed:

$$\mathbb{P}(E) = \sum_H P(H, E) \quad (32)$$

$$= \sum_H \prod_i P(X_i | \Pi(X_i)) \quad (33)$$

$$\leq \sum_H \prod_i P^u(X_i | \Pi(X_i, \lambda_i^u)). \quad (34)$$

The conditional distribution  $P(H|E) = P(H, E)/P(E)$  is the ratio of two (marginal) probabilities. Bounds on it can thus be obtained using bounds on both the numerator and denominator.

# Obtaining Bounds: Convex Duality

Convex duality gives us a way to obtain upper bounds for a concave function (or lower bounds for a convex function). A concave function  $f(x)$  can be represented as a *dual* or *conjugate* function as follows:

$$f(x) = \min_{\lambda} \{\lambda^T x - g(\lambda)\} \quad (35)$$

and vice versa:

$$g(\lambda) = \min_x \{\lambda^T x - f(x)\}. \quad (36)$$

# Obtaining Bounds: Convex Duality

For example if  $f(x) = \log x$ , we have

$$g(\lambda) = \min_x \{\lambda^T x - \log x\}. \quad (37)$$

Taking the derivative with respect to  $x$  and setting to zero yields  $x = \lambda^{-1}$ . Substituting back in yields

$$g(\lambda) = \log \lambda + 1 \quad (38)$$

giving us the *variational* transformation

$$\log(x) = \min_{\lambda} \{\lambda^T x - \log \lambda - 1\}. \quad (39)$$

# Obtaining Bounds: Convex Duality

In general we have

$$\log(x) \leq \lambda^T x - \log \lambda - 1 \quad (40)$$

for a given  $x$  and all values of  $\lambda$ . Thus the dual function gives us a family of upper bounds. The minimum over these bounds is the exact value of the logarithm, for a given  $x$ .

# QMR-DT Example

QMR-DT is a hand-constructed graphical model based on expert medical knowledge of the probabilities of various diseases given various symptoms. In this model, all variables are binary and the local conditional probabilities for each symptom not being present have the form

$$\mathbb{P}(S_i = 0|D) = (1 - \alpha_{i0}) \prod_{j \in \Pi(S_i)} (1 - \alpha_{ij})^{D_j} \quad (41)$$

where  $\alpha_{ij}$  is the probability assigned by an expert that the  $i^{\text{th}}$  symptom is present if only the  $j^{\text{th}}$  disease is present.

This can be rewritten as

$$\mathbb{P}(S_i = 0|D) = e^{\sum_{j \in \Pi(S_i)} \beta_{ij} D_j - \beta_{i0}} \quad (42)$$

where  $\beta_{ij} \equiv -\log(1 - \alpha_{ij})$ . The probability of a symptom being present thus has the form

$$\mathbb{P}(S_i = 1|D) = 1 - e^{\sum_{j \in \Pi(S_i)} \beta_{ij} D_j - \beta_{i0}}. \quad (43)$$

If we consider the joint probability over diseases and symptoms,

$$\mathbb{P}(S, D) = P(S|D)P(D) \quad (44)$$

$$= \left( \prod_i \mathbb{P}(S_i|D) \right) \left( \prod_j \mathbb{P}(D_j) \right) \quad (45)$$

where the  $\mathbb{P}(D_j)$  prior probabilities are obtained by experts, we will see cross-product terms of the form  $D_j D_k$  for  $j \neq k$ .

# QMR-DT Example

The function  $1 - e^{-x}$  is *log concave*, i.e. its logarithm is a concave function. By convex duality we find its conjugate function

$$g(\lambda) = -\lambda \log \lambda + (\lambda + 1) \log(\lambda + 1) \quad (46)$$

so that

$$1 - e^{-x} \leq e^{\lambda x - g(\lambda)}. \quad (47)$$

Since the bound is convex, optimizing for  $\lambda$  can be performed globally using unconstrained methods.

# QMR-DT Example

Plugging in, we obtain

$$\mathbb{P}(S_i = 1|D) \leq 1 - e^{\lambda_i \left( \sum_{j \in \Pi(S_i)} \beta_{ij} D_j - \beta_{i0} \right) - g(\lambda_i)} \quad (48)$$

$$= e^{\lambda_i \beta_{i0} - g(\lambda_i)} \prod_{j \in \Pi(S_i)} \left( e^{\lambda_i \beta_{ij}} \right)^{D_j}. \quad (49)$$

If we now form the joint probability by taking products over  $\mathbb{P}(S_i|D)$ , we do not obtain cross-product terms, *i.e.* we have decoupled all  $D_j \neq D_k$ .

Effectively the variational bound corresponds to using an approximating graph which has some arcs removed. We can now do exact inference on this simplified graph.

# QMR-DT Example

Which nodes to transform? In the original work on this model, first all the nodes were transformed. Then a heuristic chooses the ordering of nodes to reinstate, based on the effect of reinstating each node individually starting from the completely transformed state: After each node is reinstated, see if the resulting graph is still amenable to exact methods; if so, reinstate it and repeat. Otherwise, stop and use an exact method on the resulting graph.

In this work only upper bounds were considered tight enough. The bounds were used as a surrogate for the actual conditional probabilities, rather than as bounds. Because lower bounds are not used, this methodology does not give full error bounds.

# Sequential vs. Block Approach

That was an example of a *sequential* approach, in which nodes are added or removed in sequence. Another kind of approach is the *block* approach, in which an approximating graph is simply chosen ahead of time by the user, generally a sub-graph which is amenable to exact inference.

An example of this is for factorial HMM's, a model consisting of linked HMM's. The approximating graph in this example would simply be the unlinked HMM's.

# Approximating Graph

Suppose  $P(X) = P(X_1, \dots, X_D)$  is the joint distribution of the original graphical model,  $H$  and  $E$  are disjoint subsets of the nodes, and we want to bound or approximate  $P(H|E)$ . By choosing an approximating graph structure we introduce an approximating family of conditional probability distributions  $Q_\lambda(H|E)$ .

# Approximating Graph

We find the best distribution in this family by minimizing the Kullback-Leibler divergence  $D(Q||P)$  over  $\lambda$ :

$$\lambda^* = \arg \min_{\lambda} D(Q_{\lambda}(H|E)||P(H|E)) \quad (50)$$

where for any distributions  $P(X)$  and  $Q(X)$

$$D(Q||P) = \sum_X Q(X) \log \frac{Q(X)}{P(X)}. \quad (51)$$

$Q_{\lambda^*}(H|E)$  is the best approximation of  $P(H|E)$  in the family of  $Q_{\lambda}(H|E)$  in the sense of KL divergence.

# Approximating Graph

One justification for using the KL divergence as a measure of approximation quality is that it yields the best lower bound on the likelihood  $P(E)$  in the family  $Q_\lambda(H|E)$ . We bound it using Jensen's inequality as follows:

$$\log P(E) = \log \sum_H P(H, E) \quad (52)$$

$$= \log \sum_H Q(H, E) \frac{P(H|E)}{Q(H|E)} \quad (53)$$

$$\geq \sum_H Q(H, E) \log \left( \frac{P(H|E)}{Q(H|E)} \right). \quad (54)$$

The difference between the left and right hand sides of this equation is the KL divergence  $D(Q||P)$ .

# Variational Methods

Note that evaluation of conditional probabilities occurs within other methods, like EM and MCMC.

When are variational methods good? If  $\mathbb{P}(X_i|\Pi(X_i))$  is nearly constant as we range across  $\Pi(X_i)$ , or if the bound is relatively insensitive to  $\lambda$  across the values of  $X_i$ , we can expect the bounds to be tight. However, this all depends on the parameters of the CPD's, not simply the graph structure. Current ways of choosing the nodes to approximate variationally only use knowledge of the graph structure, which is suboptimal. This is basically an open problem.

# Main Things You Should Know

- What graphical model inference is, and what we're computing
- How elimination and message-passing works
- What the belief propagation, sum-product, and junction-tree algorithms are
- What variational methods do