STAT 535 Lecture 8

The Junction Tree Algorithm: Remarks, Variants, Sum-Product Algorithm

©Marina Meilă
mmp@stat.washington.edu

JUNCTION TREE ALGORITHM

1.	For every observed variable E_j find clique C_j that contains E_j. Set $\phi_{C_j} \leftarrow \phi_{C_j}\delta_{E_j=e_{0,j}}$	*Enter evidence*
2.	Choose a root clique C. Recursively on the rooted tree with root C, starting with $\tilde{C} = C$ for all C' children of $\tilde{C}$ i. $\tilde{C}$ calls *Collect evidence* in C' ii. ABSORB$(C' \to \tilde{C})$	*Collect evidence*
3.	Normalize ϕ_C and store $Z = \sum_{x_{C\setminus E}} \phi(x_C)$	*Normalize*
4.	Recursively on the rooted tree with root C, starting with $\tilde{C} = C$ for all children C' of $\tilde{C}$ ABSORB$(\tilde{C} \to C')$	*Distribute evidence*

Figure 1: Junction Tree Algorithm

1 Variations in message propagation

Exercise: show that the JT need not be calibrated before step 1, but it must satisfy $\phi_V = P_V$

The propagation of ABSORBTIONS can be asimilated with propagation of information in the tree, in which a clique C sends a **message** to neighboring clique C' consisting of ϕ_S^{new}/ϕ_S. From now on we will often talk about the algorithm in terms of *message passing*.

The order of the messages in COLLECTEVIDENCE or DISTRIBUTEEVIDENCE is not unique. For calibrated message propagation is sufficient that

Rule 1 | before ABSORB$(C \to C')$, C receives messages from all its neighbors other than C' |

Rule 2 Each clique must receive messages from all its neighbors.

Any schedule that complies with these rules is a valid schedule for JT propagation, that will lead to a calibrated JT. If we also want normalization, and if we want to perform it in a single clique only, then we need to be more restrictive in the schedules.

There is a practical "exception" from the second rule above: if two cliques are calibrated before ABSORB, then the absorbtion needs not be performed. Hence, the branches of the JT that contain no evidence do not need to perform COLLECTEVIDENCE. But they still need to be updated in the DISTRIBUTEEVIDENCE phase. **Exercise:** why?

2 New evidence, retraction of evidence, uncertain evidence

The JT framework can be summarized as follows:

prior	enter evidence	propagation, normalization	posterior
$\phi_V = P_V$	$\phi_V \propto P_{V \setminus E \mid E}$		$\phi_V = P_{V \setminus E \mid E}$
calibrated	uncalibrated		calibrated

If now new evidence $E' = e'$ is observed, we can treat the current ϕ_V as prior, and perform another round of entering evidence and propagation. The result will be that the data structure ϕ_V will contain the updated, calibrated posterior of the unobserved variables given $E = e, E' = e'$. And so on.

The result of successive rounds of entering evidence and JT propagation will be correct and cosistent, as long as the evidence is non-contradictory. If contradictory evidence is entered, then after propagation $\phi_V \equiv 0$. For instance, if in the JT of Handout 7, Figure 1, we enter first $A = 0$ then later $A = 1$, all the configurations with $a \neq 0$ will be set to 0 after the first observation, including the ones with $a = 1$. Then, when we observe $A = 1$, we also set to zero the configurations with $a = 0$, which means that the whole clique potential becomes 0, and by propagation all of ϕ_V is zero. This emphasizes the fact that, when we observe new variables, the

assumption is that the system of variables V is in a fixed but (paritally unknown) configuration (e.g A cannot be both 0 and 1 in the same time) of which we successively gather more knowledge.

But it is legitimate to ask another question: what if A were 1 and not 0 as it was observed? This is called **retraction of evidence**. In the algorithm given in Figure 1, we cannot do that, because we have destroyed the original distribution P_V when we entered the first observation (say $A = 0$). In order to be able to retract evidence, the JT algorithm is modified as follows:

- The clique potentials $\phi_C = P_C$ are never changed.
- One creates **evidence messages** with value $\delta_{E=e_0}$ which enter the respective cliques (but they are not multiplied into the clique potentials)
- For each separator, instead of the "permanent" ϕ_S and the "temporary" ϕ_S^{new} one keeps two messages, $\phi_S^{C\to C'}$ and $\phi_S^{C'\to C}$ one for each direction of absorbtion.
- During $\textsc{Absorb}(C \to C')$ the message from C to C' is computed by multiplying all the messages into C and marginalizing

$$\phi_S^{C\to C'} \;=\; \left[\sum_{x_{C\S}} \phi_C \delta_{E\cap C} \prod_{S'\neq S} \phi_{S'}^{\tilde{C}\to C}\right] / \phi_S^{C'\to C} \tag{1}$$

  but the destination ϕ'_C is not updated.
- To compute the posterior marginal, a clique multiplies all messages it receives by its potential (and normalizes).

$$P_{C\setminus E|E=e_0} \;\propto\; \phi_C^{posterior} \;=\; \phi_C \delta_{E\cap C} \prod_{S'} \phi_{S'}^{\tilde{C}\to C} \tag{2}$$

This JT algorithm is initialized with $\phi_C = P_C$ for all cliques C and with $\phi_S^{C\to C'} = 1$ for all separator messages.

3 Relations with other algorithms and extensions

$\textsc{CollectEvidence}(C)$ is equivalent to Variable Elimination (VE) of all variables but those in C in an order given by the JT. The gain of JT algorithm vs VE is that with $\textsc{CollectEvidence}(C)$ followed by $\textsc{DistributeEvidence}(C)$ we perform simulataneously the variable eliminations for all cliques.

There are many similarities with the Forward-Backward (FB) algorithm. The Forward step is CollectEvidence towards the last clique (plus marginalization of the Q_{n-1}). The Backward pass is CollectEvidence toward the first clique (plus marginalization of Q_2). Computing $P(Q_i|y)$ for some $1 < i < n$ is CollectEvidence towards [a clique containing] Q_i (plus marginalizing out the other variable, and normalization).

[Why Running Intersection Property?]

[Why tree?]

The JT algorithm is an example of several important classes of probabilistic algorithms:

- **Sum-Product algorithms**
- **Iterative Proportional Fitting (IPF)**
- **Generalized Distributive Law algorithms**

4 The Shenoy-Shaefer algorithm: Sum-Product Algorithm for junction trees

Data structure and Initialization This algorithm keeps a potential ϕ_C for each clique. For each separator, there are *two "potential" tables*, one for each direction of propagation. They are called **messages** and denoted by $\mu_{C\to C'}, \mu_{C'\to C}$.

The J.T. is rooted at clique C_0 and the clique potentials are intialized to the conditional probability given the parent clique

$$\phi_{C_0} = P_{C_0}(x_{C_0}) \qquad \phi_C(x_C) = P_{C\setminus pa(C)}(x_{C\setminus pa(C)} \,|\, x_{pa(C)}) \tag{3}$$

so that $P_V = \prod_C \phi_C$

Entering evidence The evidence for an observed variable E is entered in a single clique containing E. In this algorithm, rather than multiplying ϕ_C by $\delta_{E=e_0}$, one creates an **evidence message** $\mu_{E=e_0\to C}$ with destination C and which is equal to $\delta_{E=e_0}$.

The message between two cliques is given by

$$\mu_{C\to C'} \;=\; \sum_{x_{C\setminus C'}} \phi_C(x_C) \prod_{C''\sim C, C''\neq C'} \mu_{C''\to C}(x_{C\setminus C''}) \tag{4}$$

In the above, $C \sim C''$ means that the two cliques are neighbors in the tree. In other words, to send a message to a neighbor, C multiplies her potential with the messages from all its neighbors *besides* C', and marginalizes out the variables not in C'.

If the clique C contains evidence, then the evidence messages are multiplied in too (one creates a fictitious C'' for each evidence message that has to be incorporated).

The propagation/message passing schedule Each clique can send a message to a neighbor only after it has received messages (i.e "collected evidence") from all its other neighbors. An example of such a schedule is to start the message passing from the leaves of the J.T and progress recursively to the root. When the root receives messages from all its children, it in turn sends messages back. These messages propagate recursively down the tree from the root's children down to the leaves. When all the leaves have received their messages the algorithm terminates. (No messages are sent to the evidence.)

Final state When the algorithm finishes, the JT data structure contains:

the same clique potentials ϕ_C as before

the evidence messages entered, unchanged

messages on each edge (each messages was calculated and updated exactly once)

For each clique C, the marginal probability P_C is

$$P_C(x_C) \;\propto\; \phi_C(x_C) \prod_{C'\sim C} \mu_{C'\to C}(x_{C\cap C'}) \tag{5}$$

In the above equation, the product includes the evidence messages.

Remarks and comparison with the J.T algorithm In this algorithm there are no divisions, only multiplications. This is nice (in a little way) from the point of view of coding (no danger of dividing 0/0). The reason

for this is that the separator potentials have already been divided out by making $\phi_C = P_{C|pa(C)}$.

The algorithm preserves the prior P_V. This means one needs not save the prior elsewhere when doing inference. It also makes it easy to retract or change the evidence.

The price one pays is in the computation. For each neighbor to which C sends a message, one must multiply all the messages from the other neighbors before marginalizing. So there are a lot (Exercise: how many?) redundant multiplications. The JT algorithm is faster because it does each multiplication exactly once (plus a division).

[**Example**]

5 The Sum-Product algorithm

(see Jordan for more details)

A **factor graph** is a data structure that has

- **variables** $A \in V$
- **factors** $C \subset V$ for arbitrary subsets of V (not necessarily cliques).
- **potentials** $\phi_A(x_A), \phi_C(x_C)$ for every variable and every factor
- **edges** $\{\, A$–C, for every C that contains $A \,\}$
- **messages** along the edges: $\mu_{C\to A}(x_A)$ from factors to variables, and $\nu_{A\to C}(x_A)$ from variables to factors

It is assumed that the potentials and messages are tables forming a data structure. The **joint probability** represented by the data structure is given by $\phi_V = \prod_{A\in V} \phi_A \prod_C \phi_C$.

Evidence $E = e_0$ is entered by multiplying the potentials of the observed variables with the respective δ functions, i.e $\phi_E(x_E) \leftarrow \phi_E(x_E)\delta_{E=e_0}$.

The messages are

$$\mu_{C\to A}(x_A) = \sum_{x_{C\setminus A}} \phi_C(x_C) \prod_{Y\in C,\, Y\neq A} \nu_{Y\to C}(x_Y) \tag{6}$$

$$\nu_{A\to C}(x_A) = \phi_A(x_A) \prod_{C':A\in C', C'\neq C} \mu_{C'\to A}(x_A) \tag{7}$$

In words, to send a message to a variable A, a factor multiplies its potential with all the *other* messages it receives, then marginalizes over all the variables but A. When a variable A sends a message to C, it multiplies its potential by the messages it receives from all the other factors it is in, except for C.

The message passing schedule has to obey

Rule 1 before sending a message to a factor C (or variable A), a variable (factor) must receive from all it neighbor factors (variables) other than receives messages from all its neighbors other than C (A)

Rule 2 Each variable (factor) must receive messages from all its neighbors.

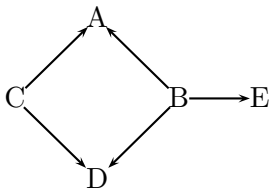
The marginal of a variable is given by

$$P_A(x_A) \propto \phi_A \prod_{C:A\in C} \mu_{C\to A}(x_A) = \nu_{A\to C'}(x_A)\mu_{C'\to A}(x_A) \quad \text{for any } C' \text{ containing } A \tag{8}$$

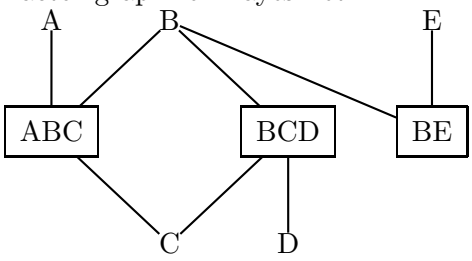
Correctness The Sum-Product algorithm gives the correct marginals only if the factor graph has no cycles.

5.1 Examples of factor graphs

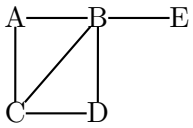
Bayes net



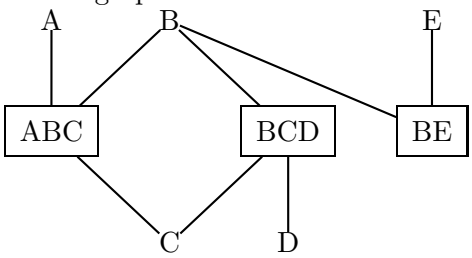
Factor graph for Bayes net



Markov field

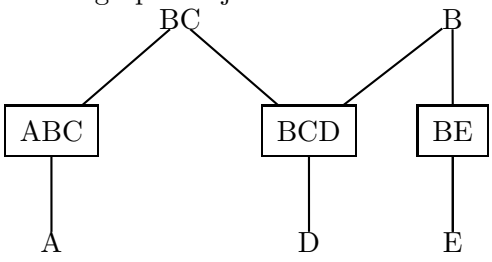


Factor graph for Markov Field

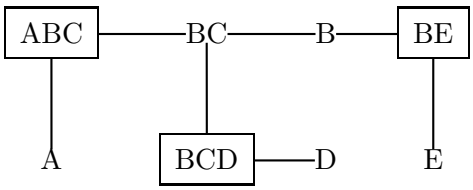


Junction Tree

Factor graph for junction tree



or



Note that in a junction tree, the separators play the role of variables, and the cliques play the role of factors.