19. Basics of message passing

19.1. Loopy message passing

Figure 19.1 describes one branch of a loopy cluster graph. It serves as a basic configuration for describing the various message passing algorithms. For ease of explanation we assume discrete random variables (i.e. factor marginalisation is done via summation), but all results also hold for continuous random variables (where marginalisation is done via integration).

Φ_a and Φ_b are the internal factor functions of nodes a and b respectively. (When more than one factor function are allocated to a cluster, it is their product). As such it is a function of the random variables involved in those functions, we suppressed explicit mention of this.

$S_{a,b}$ is the sepset linking nodes a and b, i.e. the set of all variables that those two clusters will exchange information about. It does so by passing message $\mu_{a,b}$ from node a to b and vice versa for $\mu_{b,a}$. We use the notation $\backslash a$ to indicate the set of all nodes excluding a (i.e. its complement). Correspondingly:

$$\mu_{\backslash b,a} = \prod_{i \in \backslash b} \mu_{i,a}$$

is the product of all messages entering node a, except for the message $\mu_{b,a}$.

The product of all incoming messages with the cluster internal factors forms the *cluster belief*:

$$\Psi_a = \Phi_a \mu_{b,a} \mu_{\backslash b,a} \tag{19.1}$$

for cluster a. Similarly the *sepset belief* is formed by the product of the two opposing messages passing through the sepset:

$$\psi_{a,b} = \mu_{a,b} \mu_{b,a} \tag{19.2}$$

for sepset $S_{a,b}$.

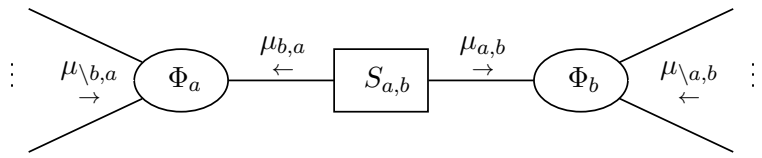

Figure 19.1.: Configuration used for explaining message passing algorithms. See the text for definitions etc..

19.1.1. Loopy belief *propagation* (a.k.a. the Shafer-Shenoy algorithm)

This is the most basic belief propagation algorithm upon which the others build. The details are in Algorithm 1.

Algorithm 1 Loopy belief propagation (LBP)

Initialization:

Form initial messages $\mu_{a,b}$ from all source clusters a to their respective destination clusters b. The standard way to do this is to use vacuous / non-informative messages $\mu_{a,b} = 1$. Instead we prefer to marginalize each neighbouring cluster to only retain the variables specified in the sepset connecting the two clusters i.e.

$$\mu_{a,b} = \sum_{\backslash S_{a,b}} \Phi_a. \tag{19.3}$$

With a tree structure for our graph these two initializations would have turned out to be equivalent. However, in a loopy system the order of message passing is important. Our preferred initialization allows us to measure how far a message deviates from its vacuous version – using this we can schedule messages in order of importance.

Iteration:

1. From each source node a to each destination node b connected to it, take the product of all other messages incoming to a, (i.e. except $\mu_{b,a}$) with the internal cluster factor and then marginalise to only retain the variables specified in the sepset $S_{a,b}$:

$$\mu^{'}_{a,b} = \sum_{\backslash S_{a,b}} \mu_{\backslash b,a} \Phi_a. \tag{19.4}$$

2. Monitor convergence by checking the distance $d(\mu_{a,b}, \mu^{'}_{a,b})$ between the old and new messages. The KL-divergence is a good choice for distance. If a message converged to within a set margin, its destination cluster needs not be propagated further.
3. $\mu_{a,b} = \mu^{'}_{a,b}$.

Termination:

The above iterations continue until either all messages have converged, or a maximum number of iterations were completed.

19.1.2. From belief *propagation* to belief *update* (LBU2)

The trick here is to notice that:

$$\begin{aligned}
\mu^{'}_{a,b} &= \sum_{\backslash S_{a,b}} \mu_{\backslash b,a} \Phi_a \\
&= \sum_{\backslash S_{a,b}} \mu_{\backslash b,a} \Phi_a \mu_{b,a} / \mu_{b,a} \\
&= \left(\sum_{\backslash S_{a,b}} \Psi_a \right) / \mu_{b,a} \qquad \text{(from Eq. 19.1 and using } \backslash S_{a,b} \cap S_{a,b} = \emptyset\text{)}.
\end{aligned}$$

Note in the last step that, since $\mu_{b,a}$ shares no variables with $\S_{a,b}$, we are free to take it out of the summation. In Algorithm 2 we use this to express our message passing i.t.o. cluster beliefs. It holds some benefits:

- Graphs with high connectivity can become considerably faster.
- The full cluster belief can be much more informative. For instance, in a discrete model some variable settings can turn out to be impossible. Or in a continuous setting where the marginalization needs to be approximated, the full cluster belief can focus that approximation to the most relevant variable values. This enhances convergence of the system.

Algorithm 2 Basic loopy belief update – (LBU2)

Initialization:

1. Form initial messages as in Algorithm 1.
2. In addition each cluster a now forms an initial cluster belief Ψ_a as the product of its internal factor(s) with all its incoming messages.

Iteration:

1. To send a message from cluster a to cluster b, we marginalise all variables not contained in the sepset between them out, and then we divide by the earlier message from the opposite direction, i.e.

$$\mu_{a,b}^{'} = \left(\sum_{\setminus S_{a,b}} \Psi_a \right) / \mu_{b,a}.$$

2. At the destination cluster we need to first remove the earlier (previous iteration) version of this message by dividing it out. Then the destination cluster gets updated by multiplication with the new message, i.e.

$$\Psi_b^{'} = \Psi_b \left(\mu_{a,b}^{'} / \mu_{a,b} \right). \tag{19.5}$$

3. Monitor convergence by checking the distance $d(\mu_{a,b}, \mu_{a,b}^{'})$ between the old and new messages. The KL-divergence is a good choice for distance. If a message converged to within a set margin, its destination cluster needs not be propagated further.
4. $\mu_{a,b} = \mu_{a,b}^{'}$.

Termination: Same as in Algorithm 1.

On the downside, two divisions are required for each message being passed. And in some cases those divisions can also introduce numerical sensitivity.

19.1.3. Loopy belief *update* (a.k.a. the Lauritzen Spiegelhalter algorithm)

Now consider the following situation: Initial messages are created as in Eq 19.3, after which the initial cluster and sepset beliefs are determined from Eqs. 19.1 and 19.2. We now want to update the sepset belief to also reflect the information in messages $\mu_{\setminus b,a}$. From definition we know this to be:

$$\begin{aligned} \psi_{a,b}^{'} &= \mu_{a,b}^{'} \mu_{b,a} \\ &= \left(\sum_{\setminus S_{a,b}} \mu_{\setminus b,a} \Phi_a \right) \mu_{b,a} && \text{(from Eq. 19.4)} \\ &= \sum_{\setminus S_{a,b}} \Psi_a, && \text{(from Eq. 19.1)} \end{aligned} \tag{19.6}$$

i.e. marginalising the cluster belief directly gives us the updated sepset belief! Now consider:

$$\begin{aligned}\Psi_b \frac{\psi^{'}_{a,b}}{\psi_{a,b}} &= \Psi_b \frac{\mu^{'}_{a,b}\mu_{b,a}}{\mu_{a,b}\mu_{b,a}} \\ &= \Psi^{'}_b, \qquad \text{(from Eq. 19.5)} \end{aligned} \tag{19.7}$$

i.e. we can update a cluster belief by dividing the old sepset belief out and multiplying the new one in. Note that in contrast to Algorithm 2 we now only do one factor division.

Algorithm 3 Loopy belief update – (LBU)

Initialization:

1. Form initial messages as in Algorithm 1.
2. Form cluster beliefs as in Algorithm 2.
3. Furthermore, each sepset $S_{a,b}$ now forms an initial sepset belief $\psi_{a,b}$ as the product of the two opposing messages passing through it.

Iteration:

1. To send a message from cluster a to cluster b, update the sepset belief:

$$\psi^{'}_{a,b} = \sum_{\backslash S_{a,b}} \Psi_a.$$

2. Use this updated sepset belief to update the target cluster belief:

$$\Psi^{'}_b = \Psi_b \frac{\psi^{'}_{a,b}}{\psi_{a,b}}$$

3. Monitor convergence by checking the distance $d(\psi_{a,b}, \psi^{'}_{a,b})$ between the old and new sepset beliefs. The KL-divergence is a good choice for distance. If it converged to within a set margin, its destination cluster needs not be propagated further.
4. $\psi_{a,b} = \psi^{'}_{a,b}$.

Termination: The above iterations continue until either all sepset beliefs have converged, or a maximum number of iterations were completed.

19.2. Residual error propagation

To start off, inference is done over some graph of probabilistic factors. In our case we use an LTRIP clustergraph (see `clustergraph.hpp` for details) – the procedure for configuring it is nicely explained in [?].

The inference code is contained in `lbu_cg.{hpp,cc}`. For a nitty-gritty level of understanding one must be aware of the important datastructures and their use. The `loopyBU_CG` inference procedure primarily relies on the following three parameters:

- `std::vector< rcptr<Factor> > ClusterGraph::factorPtrs`: These are the `Factor`s that the inference is going to operate on. See `clustergraph.hpp` for more detail.
- `std::map< Idx2, rcptr<Factor> > ssBeliefs`: This contains the marginal beliefs over the sepsets at any given stage of computation. The `Idx2` field is a `std::pair<unsigned,unsigned>` indicating between which two factors this sepset resides.