PGMs session 5 Inference: Junction Trees and the Belief-Update Algorithm

In the previous assignment, we have looked at belief propagation in cluster graphs; in this assignment, we look at two related concepts:

- Junction trees, which are basically tree-structured cluster graphs
- The belief-update algorithm, which is an alternative to the belief-propagation algorithm

1 Preparation: Watch the Koller videos on variable elimination and clique trees

Watch the following videos in the Coursera course Probabilistic Graphical Models 2: Inference, Week 2:

- *Clique Trees: Clique Tree Algorithm Correctness (18:23)*
- *Clique Trees: Clique Tree Algorithm Computation (16:18)*
- *Clique Trees: Clique Trees and Independence (15:21)*
- *Clique Trees: Clique Trees and VE (16:17)*

Notes on these videos:

- What is elsewhere known as a 'junction tree', Koller calls a 'clique tree', a very apt name since it is a tree of the maximal cliques.
- The message-passing algorithm Koller uses here is belief propagation (also known as the Shafer-Shenoy or sum-product algorithm), which you encountered in Session 4.
- With an appropriate message schedule, we only need to pass a message *once* in both directions over each link to achieve convergence to the *exact* marginal distribution. This is in contrast to the more general loopy cluster graphs (of Session 4), where inference is *approximate* and takes *several* message passing iterations to converge.
- Variable elimination plays a key role in determining the clique/junction tree structure. Different elimination orderings can result in different induced graphs, with possible major differences in computational complexity.
- Each sepset separates the resulting tree in two parts with no paths linking them except through this sepset. For many problems of practical interest, the size of the largest sepset makes it intractable to calculate and store the messages sent across this sepset, even for a well-chosen elimination ordering; this is the main drawback of the clique/junction tree.

2 Preparation: Read Barber Chapter 6 – The Junction Tree Algorithm

Read Chapter 6 in Barber. The discussion below will give you an indication of which parts of Chapter 6 to focus on.

- Section 6.1: Make sure you understand the concept of reparameterisation, which lays the groundwork for the absorption/belief-update algorithm.
- Section 6.2: Take note of the following terms:
  - Barber's 'clique graph' is similar to Koller's 'cluster graph', except that it does not necessarily obey the running intersection property (RIP)
  - Barber's 'clique tree' does not necessarily obey the RIP – it is therefore not the same thing as Koller's 'clique tree' (see Definition 6.4 on p.113)
  - Barber's 'junction tree' is the same thing as Koller's 'clique tree'

  Pay close attention to the absorption algorithm (Subsection 6.2.1), which is what we call the 'belief-update algorithm' (and which is also known as the 'Lauritzen-Spiegelhalter algorithm'). Specifically, take note of the following:
  - In the belief-propagation algorithm of Session 4, the cluster potentials consisted of factors that did not change during message passing; in the absorption/belief-update algorithm, the cluster and sepset potentials are updated during message passing.
  - Make sure you understand what a valid absorption schedule is for a tree-structured graph (Subsection 6.2.2).
- Sections 6.3 to 6.6 cover much of the same ground as Koller video *Clique Trees: Clique Trees and VE (16:17)*, but from a different perspective and using a somewhat different procedure. We do not focus on algorithms to construct junction/clique trees in this module; just take note that they exist – there is no need to study these sections in detail. Subsection 6.6.5 discusses Shafer-Shenoy propagation, which is the same as the belief-propagation/sum-product algorithm Koller uses.
- You can leave out Sections 6.7 and 6.8 (Section 6.7 is the topic of Session 6). Section 6.9 makes an important point that you should take note of.

3 Preparation: Read Section 19.1 of the EMDW Development Guide

Pay close attention to Subsection 19.1.3 Loopy belief *update* (a.k.a. the Lauritzen-Spiegelhalter algorithm).

- The belief-update algorithm here is applicable to loopy cluster graphs, whereas in Barber it was only applied to tree-structured graphs.
- Terminology: cluster and sepset *belief* here correspond to cluster and sepset *potentials* in Barber.
- Make sure you understand how to initialise the cluster and sepset beliefs (for loopy cluster graphs).

4 Practical: Clique trees and loopy belief update (LBU) for the Hamming (7,4) error-correction code

Objectives: To do the following for the Hamming (7,4) error-correction code problem:

- Manually implement belief-propagation/the Shafer-Shenoy algorithm in a clique tree
- Manually implement the belief-update/absorption/Lauritzen-Spiegelhalter algorithm in a clique tree
- Manually implement the belief-update/absorption/Lauritzen-Spiegelhalter algorithm in a loopy cluster graph
- Use `emdw`'s built-in functionality to implement the belief-update/absorption/Lauritzen-Spiegelhalter algorithm in a loopy cluster graph

Task 1: Use the variable naming convention from the previous tasks for the Hamming (7,4) code; i.e., the check bits, b_4, b_5, b_6, the three message bits each only shared by two parity checking potentials, b_1, b_2, b_3, and the message bit shared by all the parity checking potentials, b_0.

(a) Determine the induced clique/junction tree for the following elimination ordering: $b_4, b_5, b_6, b_1, b_2, b_3, b_0$. Your clique tree should look something like that in Figure 1. You can absorb the $\Phi_P(b_i)$ factors – which are the $p(r_i|b_i)$ factors that have been reduced with the observed received bit r_i – into the larger factors Φ_C before you continue.

(b) Specialise the manual (i.e., using only the basic factor operators) belief-propagation/Shafer-Shenoy/sum-product message passing system you implemented in Session 4 to the clique/junction tree you created here; i.e., use a proper message-passing schedule that only sends a message when all the incoming messages it relies on are finalised. Also perform only one pass in each direction of each link.

(c) After message passing, calculate the marginal beliefs. Compare the resultant decodings to that obtained from the full joint distribution in Session 3, as well as those from using the belief-propagation/Shafer-Shenoy/sum-product algorithm on a loopy graph (Session 4's assignment).

Task 2: Use the same clique/junction tree you set up in Task 1.

(a) Manually implement the belief-update/absorption/Lauritzen-Spiegelhalter algorithm. Make sure you follow a valid message-passing schedule as discussed in Subsection 6.2.2 in Barber.

(b) After message passing, extract the marginal beliefs and compare them with that of Task 1 above.

Task 3: Now use the same loopy cluster graph you used for Session 4's assignment.

(a) Manually implement the belief-update/absorption/Lauritzen-Spiegelhalter algorithm. Make sure you initialise the cluster and sepset beliefs/potentials as described in Subsection 19.1.3 of the EMDW Development Guide. Choose a message-passing schedule similar to the one you used in Session 4's assignment. Also use a similar convergence criterium to the one you used in Session 4's assignment.

(b) After message passing, extract the marginal beliefs and compare them with that of Task 2 above as well as Task 1 of Session 4.

Something to ponder: For which type of cluster graph structure do you expect belief update to be more efficient than belief propagation? (Hint: Compare the way messages are calculated.)

Task 4: Use the same loopy cluster graph you used for Session 4's assignment (and for Task 3 above), and now use `emdw`'s built-in functionality to implement the belief-update/absorption/Lauritzen-Spiegelhalter algorithm. (This simply involves taking your code for Task 2 of Session 4 and replacing the function `loopyBP_CG` with `loopyBU_CG`, as well as replacing the function `queryLBP_CG` with `queryLBU_CG`.) Extract the marginal beliefs and compare them with that of Task 3 above.

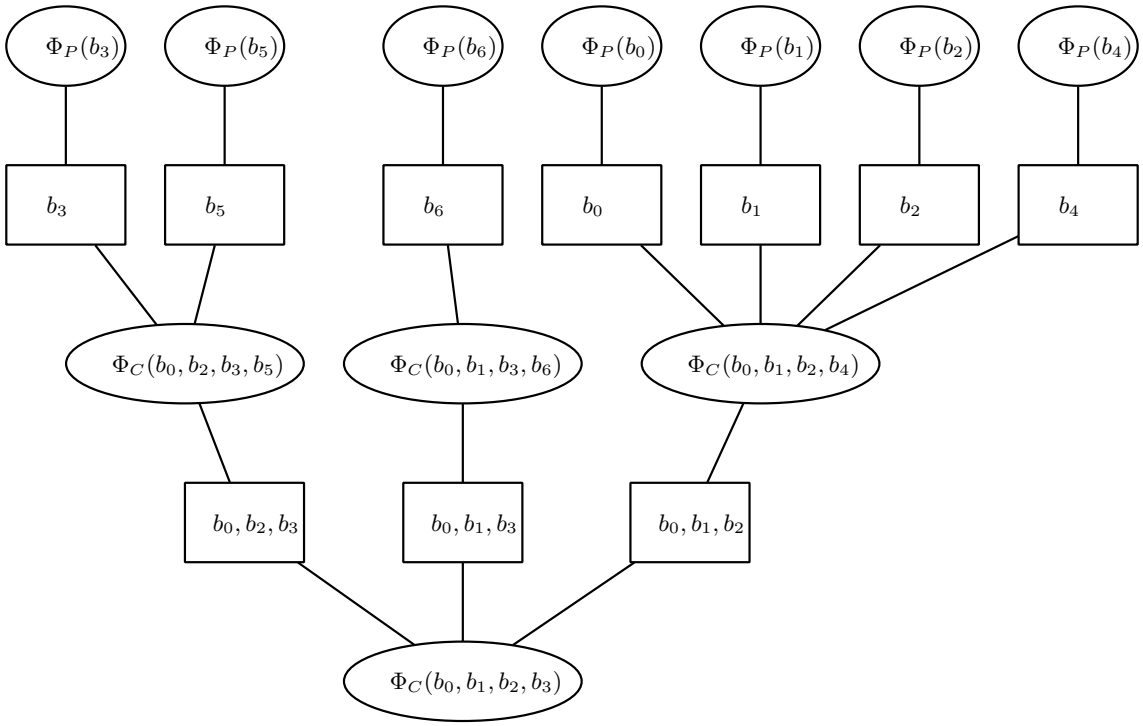

Figure 1: Junction Tree for Hamming (7,4) error correcting code using elimination order $b_4, b_5, b_6, b_1, b_2, b_3, b_0$.