

# CS 228, Winter 2009

## Problem Set #1 Solutions

---

### 1. Partially Directed Acyclic Graphs (PDAGs) [10 points]

All graphs that have the same skeleton and immoralities are I-equivalent. In this problem we will investigate a compact representation for such an equivalence class. Using this representation, we can provide a quick check for edges with required directionality.

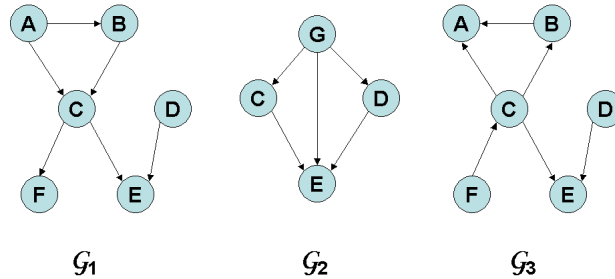


Figure 1: Two Bayesian Networks

- (a) [3 points] First consider the case of  $\mathcal{G}_1$  above. For which edges do there exist graphs in the same I-equivalence class in which the edge points in the other direction? Conversely, which edges have fixed direction across all graphs in the I-equivalence class? Based on this example, can you construct a simple but general sufficient condition for some edges to have fixed direction? The condition applies to a local structure of three nodes with fixed skeleton and immorality. If we find this structure in the graph, we can fix the direction of its two edges.

**Answer:**  $C \rightarrow E$  and  $D \rightarrow E$  are fixed. The rest are not as we can see in  $\mathcal{G}_3$ , which is I-equivalent to  $\mathcal{G}_1$  but with all the edges reversed except for  $C \rightarrow E$  and  $D \rightarrow E$ .

General sufficient condition: if  $ABC$  is an immorality with  $B$  in the middle, then  $AB$  and  $BC$  have fixed direction.

Error codes:

- (1.1) [1 points] Incorrect edges
- (1.2) [2 points] Incorrect general condition

- (b) [3 points] Now assume the edge from  $A$  to  $B$  and the edge from  $B$  to  $C$  in  $\mathcal{G}_1$  have fixed direction for some reason (i.e., in all I-equivalent graphs, the edge must take the direction from  $A$  to  $B$  and also from  $B$  to  $C$ ). As a result of this, which additional

edge in the graph is forced to have a fixed direction? Now with this additional edge fixed, there is one more edge that is also forced to have fixed direction. Can you find that one? Provide the two (general) sufficient conditions for a fixed edge that you applied in this example. The first condition applies to a local structure of three nodes with three edges, two of which have fixed direction. The second condition applies to a local structure of three nodes with two edges, one of which has fixed direction. If we find either of the above structures in the graph, we can fix the direction of the remaining edge.

**Answer:**  $A \rightarrow C$  has a fixed direction to avoid a cycle in the Bayesian network.

General sufficient condition: if  $A \rightarrow B$  and  $B \rightarrow C$  have a fixed direction, then the edge between  $A$  and  $C$  must have a fixed direction from  $A$  to  $C$ .

$C \rightarrow F$  is also fixed to avoid creating immorality at  $FCA$  and  $FCB$ .

General sufficient condition: if  $A \rightarrow B$  has a fixed direction and there is no edge between  $A$  and  $C$ , then the edge between  $B$  and  $C$  must have a fixed direction from  $B$  to  $C$  if there is no immorality at  $ABC$  in the original graph.

Error codes:

(1.3) [1 points] Incorrect edges

(1.4) [1 points] Incorrect general condition (1 point for each condition)

- (c) [4 points] Now suppose we take the case of  $\mathcal{G}_2$ . Applying the general conditions determined in (a) and (b), we get several edges with fixed directionality. However, there is one more edge that is not covered by the previous conditions, but still has fixed direction in all I-equivalent graphs. Which edge is that? Can you determine the (final) general sufficient condition that will cover this case? The condition applies to a local structure of four nodes with five edges, two of which have fixed direction. If we find this structure in the graph, we can fix the direction of one more edge.

**Answer:**  $C \rightarrow E$  and  $D \rightarrow E$  are fixed by the previous conditions. In this case,  $G \rightarrow E$  is also fixed. If any graph has  $E \rightarrow G$ , it must have  $C \rightarrow G$  and  $D \rightarrow G$  to avoid cycle. However, this will create a new immorality, which makes the new graph not I-equivalent to the original one. Therefore, we have to have  $G \rightarrow E$ .

General sufficient condition: if a subgraph has the form of  $\mathcal{G}_2$ , i.e.  $CED$  is the only immorality for this structure of four nodes with five edges, then the edge between  $G$  and  $E$  must have a fixed direction from  $G$  to  $E$ .

Error codes:

(1.5) [4 points] Incorrect answer

2. **Variable Elimination for Pairwise Marginals**[10 points] Consider the task of using a calibrated clique tree  $\mathcal{T}$  over factors  $\mathcal{F}$  to compute all of the pairwise marginals of variables,  $P_{\mathcal{F}}(X, Y)$  for all  $X, Y$ . Assume that our probabilistic network consists of a chain  $X_1 - X_2 - \dots - X_n$ , and that our clique tree has the form  $C_1 - \dots - C_{n-1}$  where  $Scope[C_i] = \{X_i, X_{i+1}\}$ . Also assume that each variable  $X_i$  has  $|Val(X_i)| = d$ .

- (a) [2 points]

What is the total cost of doing variable elimination over the tree, as described in the algorithm of Figure 1 of the supplementary handout (see online), for all  $\binom{n}{2}$  variable pairs? Describe the time complexity in term of  $n$  and  $d$ .

**Answer:** To compute  $P_{\mathcal{F}}(X_i, X_j)$  using CTree-Query in Figure 2, we need to eliminate  $j - i - 1 = O(n)$  variables. The largest factor generated has three nodes. Therefore, the complexity for each query is  $O(nd^3)$ , assuming each of  $X_1, \dots, X_n$  has domain size  $d$ . Since we need to run the query  $\binom{n}{2} = O(n^2)$  times, the total time complexity is  $O(n^3d^3)$ .

Error codes:

(2.1) [1 points] Minor error in runtime calculation (e.g.,  $O(n^3d^2)$ )

(2.2) [2 points] Major error in runtime calculation (e.g.,  $O(d^n)$ )

(b) [8 points]

Since we are computing marginals for all variable pairs, we may store any computations done for the previous pairs and use them to save time for the remaining pairs. Construct such a dynamic programming algorithm that achieves a running time which is asymptotically significantly smaller. Describe the time complexity of your algorithm.

**Answer:** Due to the conditional independence properties implied by the network, we have that, for  $i < j - 1$ :

$$\begin{aligned} P(X_i, X_j) &= \sum_{X_{j-1}} P(X_i, X_{j-1}, X_j) \\ &= \sum_{X_{j-1}} P(X_i, X_{j-1})P(X_j | X_i, X_{j-1}) \\ &= \sum_{X_{j-1}} P(X_i, X_{j-1})P(X_j | X_{j-1}) \end{aligned}$$

The term  $P(X_j | X_{j-1})$  can be computed directly from the marginals in the calibrated clique tree, while  $P(X_i, X_{j-1})$  is computed and stored from a previous step if we arrange the computation in a proper order. Following is the algorithm:

```

-----
//  $\pi_j = P(X_j, X_{j+1})$ : calibrate potential in clique  $C_j$ .
//  $\mu_{j-1,j} = P(X_j)$ : message between clique  $C_{j-1}$  and  $C_j$ .
 $\mu_{0,1} = \sum_{X_2} \pi_1(X_1, X_2)$ 
for j = 1 to n - 1 do
     $\psi(X_j) = \frac{\pi_j}{\mu_{j-1,j}}$ 
     $\phi(X_j, X_{j+1}) = \pi_j$ 
for i = 1 to n - 2 do
    for j = i + 2 to n do
         $\phi(X_i, X_j) = \sum_{X_{j-1}} \phi(X_i, X_{j-1}) \times \psi(X_{j-1})$ 
-----

```

where  $\psi(X_j) = P(X_{j+1}|X_j)$  and  $\phi(X_i, X_j) = P(X_i, X_j)$ .

The algorithm run through the double loop  $i, j$  for  $O(n^2)$  times. Each time, it performs a Product-Sum of two factors. The immediate factor has three variables and thus it costs  $O(d^3)$  time. Therefore, the total time complexity is  $O(n^2d^3)$  which is asymptotically significantly smaller.

Error codes:

- (2.3) [1 points] Minor error in runtime calculation carried over from first part (e.g.,  $O(n^2 d^2)$ )
- (2.4) [2 points] Other minor error in runtime calculation
- (2.5) [6 points] Missing or major error in runtime analysis
- (2.6) [2 points] Incorrect dynamic programming algorithm (not better than naive algorithm)
- (2.7) [2 points] Underspecified algorithm (e.g., don't show how to compute new marginals or unspecified base case/initialization )

**Estimate:** 1.5 pages

**3. Greedy Ordering for Variable Elimination [35 points]**

In this problem we will consider the computational implications of various variable elimination ordering strategies.

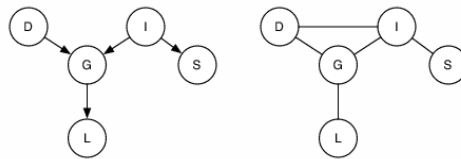


Figure 2:  $\mathcal{G}$  and  $\mathcal{M}[\mathcal{G}]$

- (a) [10 points] Consider the student BN  $\mathcal{G}$  above and its undirected moralized graph  $\mathcal{M}[\mathcal{G}]$ . As discussed in 7.3.2, variable elimination can be understood in terms of operations on an undirected graph (or for a BN, transformations on its undirected moralized graph). Create an elimination ordering  $\prec$  over the nodes in  $\mathcal{M}[\mathcal{G}]$  and show the resulting induced graph  $\mathcal{I}_{\mathcal{G}, \prec}$ .

**Answer:** Eliminating  $G$  adds edges  $D - L$  and  $I - L$ . Eliminating  $D$  then adds no edges because  $I$  and  $L$  are already connected. Eliminating  $I$  adds an edge  $S - L$  (NOT an edge  $S - D$  since  $D$  has already been eliminated). Then eliminating  $L$  and  $S$  add no extra edges.

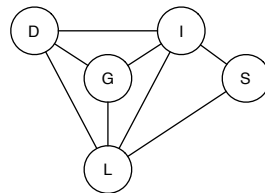


Figure 3:  $\mathcal{I}_{\mathcal{G}, \prec}$

- (b) [25 points] Consider the following greedy algorithm for producing a good elimination ordering:

**Greedy VE Ordering**

- initialize all nodes as unmarked

- for  $k = 1 : n$ 
  - choose the unmarked node that minimizes some greedy function  $f$
  - assign it to be  $X_k$  and mark it
  - add edges between all of the unmarked neighbors of  $X_k$
- output  $X_1 \dots X_k$

Consider three greedy functions  $f$  for the above algorithm:

- $f_A(X_i) =$  number of unmarked neighbors of  $X_i$
- $f_B(X_i) =$  size of the intermediate factor produced by eliminating  $X_i$  at this stage
- $f_C(X_i) =$  number of added edges caused by marking  $X_i$

Show that none of these functions produces an algorithm that dominates the others. That is, give an example (a BN  $\mathcal{G}$ ) where  $f_A$  produces a more efficient elimination ordering than  $f_B$ . Then give an example where  $f_B$  produces a better ordering than  $f_C$ . Finally, provide an example where  $f_C$  is more efficient than  $f_A$ . For each case, define the undirected graph, the factors over it, and the number of values each variable can take. From these three examples, argue that none of the above heuristic functions are optimal.

**Answer:** I will show an example where  $f_B$  is better than both  $f_A$  and  $f_C$  and an example where  $f_A$  is better than  $f_C$ .

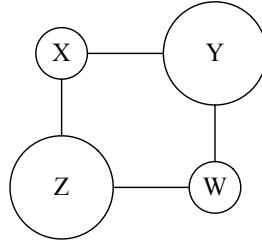


Figure 4:  $f_B$  better than  $f_A$  and  $f_C$

Consider Figure (4). Suppose pairwise factors  $\phi(X, Y)$ ,  $\phi(Y, W)$ ,  $\phi(X, Z)$ , and  $\phi(Z, W)$ . Suppose  $|Val(X)| = |Val(W)| = d$ ,  $|Val(Y)| = |Val(Z)| = D$ , and  $D \gg d$ .

$f_B$  could choose the ordering  $Y, Z, X, W$  (it's ensured to pick one of  $Z$  or  $Y$  first to avoid creating a large factor over both  $Z$  and  $Y$ ). The cost of variable elimination under this ordering is  $Dd^2$  multiplications and  $(D-1)d^2$  additions to eliminate  $Y$ ,  $Dd^2$  multiplications and  $(D-1)d^2$  additions to eliminate  $Z$ ,  $d^2$  multiplications and  $(d-1)d$  additions to eliminate  $X$ , and  $(d-1)$  additions to eliminate  $W$ .

$f_A$  and  $f_C$  could each choose the ordering  $X, Y, Z, W$  (any possible ordering is equally attractive to these heuristics because they don't consider information about domain sizes). The cost of variable elimination under this ordering is  $D^2d$  multiplications and  $D^2(d-1)$  additions to eliminate  $X$ ,  $D^2d$  multiplications and  $(D-1)Dd$  additions to eliminate  $Y$ ,  $Dd$  multiplications and  $(D-1)d$  additions to eliminate  $Z$ , and  $(d-1)$  additions to eliminate  $W$ .

Since  $D \gg d$  the  $D^2(d-1)$  terms in the cost of variable elimination under an ordering produced by  $f_C$  or  $f_A$  outweigh the cost of variable elimination under an ordering produced by  $f_B$ . So neither  $f_A$  nor  $f_C$  dominates. The intuition in this example is

that  $f_A$  and  $f_C$  can create unnecessarily large factors because they don't consider the domain sizes of variables.

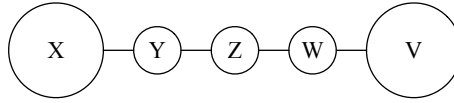


Figure 5:  $f_A$  better than  $f_B$

Consider Figure (5). Suppose pairwise factors  $\phi(X, Y)$ ,  $\phi(Y, Z)$ ,  $\phi(Z, W)$ , and  $\phi(W, V)$ . Suppose  $|Val(Y)| = |Val(Z)| = |Val(W)| = d$ ,  $|Val(X)| = |Val(V)| = D$ ,  $D = 13$ , and  $d = 2$ .

$f_A$  could choose the ordering  $X, Y, Z, W, V$  (it's ensured to only pick from the ends of chain). The cost of variable elimination under this ordering is  $(D - 1)d$  additions to eliminate  $X$ ,  $d^2$  multiplications and  $(d - 1)d$  additions to eliminate  $Y$ ,  $d^2$  multiplications and  $(d - 1)d$  additions to eliminate  $Z$ ,  $Dd$  multiplications and  $(d - 1)D$  additions to eliminate  $W$ , and  $(D - 1)$  additions to eliminate  $V$ . This is 34 multiplications and 53 additions.

$f_B$  could choose the ordering  $Z, X, Y, W, V$  (it's ensured to pick  $Z$  first since the size of a factor over  $X, Z, W$  is 8, which is less than that for eliminating  $X$  or  $V$  (26) and less than that for eliminating  $Y$  or  $W$  (52)). The cost of variable elimination under this ordering is  $d^3$  multiplications and  $(d - 1)d^2$  additions to eliminate  $Z$ ,  $Dd$  multiplications and  $(D - 1)d$  additions to eliminate  $X$ ,  $d^2$  multiplications and  $(d - 1)d$  additions to eliminate  $Y$ ,  $Dd$  multiplications and  $(d - 1)D$  additions to eliminate  $W$ , and  $(D - 1)$  additions to eliminate  $V$ . This is 64 multiplications and 55 additions.

So  $f_B$  doesn't dominate. The intuition in this example is that  $f_B$  can avoid dealing with large intermediate factors that it will eventually have to deal with anyways, and in the meantime create a bigger mess for itself.

Error codes:

- (3.1) [0 points] No elimination ordering provided
- (3.2) [2 points] Incorrect induced graph
- (3.3-5) [6 points] Didn't show that  $f_A$ ,  $f_B$  or  $f_C$  doesn't dominate.(respectively)
- (3.6) [1 points] Minor error.
- (3.7) [5 points] Unclear justification or reasoning

#### 4. Markov Networks for Image Segmentation [18 points]

In class we mentioned that Markov Networks are commonly used for many image processing tasks. In this problem, we will investigate how to use such a network for image segmentation. The goal of image segmentation is to divide the image into large contiguous regions (segments), such that each segment is internally consistent in some sense.

We begin by considering the case of a small  $3 \times 2$  image. We define a variable  $X_i$  for each node (pixel) in the network, and each can take on the values  $1 \dots K$  for  $K$  image segments. The resulting Markov Network  $\mathcal{M}$  is shown in Figure 6.

We will perform our segmentation by assigning factors to this network over pairs of variables. Specifically, we will introduce factors  $\phi_{i,j}$  over neighboring pixels  $i$  and  $j$  that

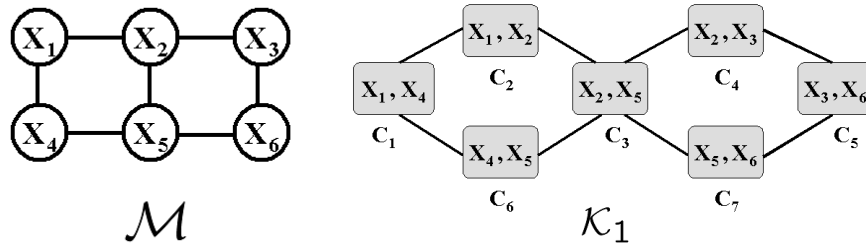


Figure 6: Markov Network for image segmentation.

quantify how strong the affinity between the two pixels are (i.e., how strongly the two pixels in question want to be assigned to the same segment).

We now want to perform inference in this network to determine the segmentation. Because we eventually want to scale to larger networks, we have chosen to use loopy belief propagation on a cluster graph. Figure 6 also shows the cluster graph  $\mathcal{K}_1$  that we will use for inference.

- (a) [1 points] Write the form of the message  $\delta_{3 \rightarrow 6}$  that cluster  $C_3$  will send to cluster  $C_6$  during the belief update, in terms of the  $\phi$ 's and the other messages.

**Answer:**

$$\delta_{3 \rightarrow 6}(X_5) = \sum_{X_2} \phi_{2,5}(X_2, X_5) \delta_{2 \rightarrow 3}(X_2) \delta_{4 \rightarrow 3}(X_2) \delta_{7 \rightarrow 3}(X_5)$$

Error codes:

(4.1) [1 points] Wrong message.

- (b) [6 points] Now, consider the form of the initial factors. We are ambivalent to the actual segment labels, but we want to choose factors that make two pixels with high “affinity” (similarity in color, texture, intensity, etc.) more likely to be assigned together. In order to do this, we will choose factors of the form (assume  $K = 2$ ):

$$\phi_{i,j}(X_i, X_j) = \begin{bmatrix} \alpha_{i,j} & 1 \\ 1 & \alpha_{i,j} \end{bmatrix} \quad (1)$$

Where  $\alpha_{i,j}$  is the affinity between pixels  $i$  and  $j$ . A large  $\alpha_{i,j}$  makes it more likely that  $X_i = X_j$  (i.e., pixels  $i$  and  $j$  are assigned to the same segment), and a small value makes it less likely. With this set of factors, compute the initial message  $\delta_{3 \rightarrow 6}$ , assuming that this is the first message sent during loopy belief propagation. Note that you can renormalize the messages at any point, because everything will be rescaled at the end anyway. What will be the final marginal probability,  $P(X_4, X_5)$ , in cluster  $C_6$ ? What’s wrong with this approach?

**Answer:**

$$\delta_{3 \rightarrow 6}(X_5) = \begin{bmatrix} 1 + \alpha_{2,5} \\ 1 + \alpha_{2,5} \end{bmatrix} \propto \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Here we see that the initial message (and indeed all subsequent messages) will be equal to the  $\mathbf{1}$  message. This means that the cluster beliefs will never be updated. Therefore:

$$P(X_4, X_5) \propto \pi_6(X_4, X_5) = \phi_{4,5}$$

$$P(X_4, X_5) = \begin{cases} \frac{1}{2+2\alpha_{4,5}} & \text{if } X_4 \neq X_5, \\ \frac{\alpha_{4,5}}{2+2\alpha_{4,5}} & \text{if } X_4 = X_5, \end{cases}$$

In this approach, because the initial potentials are symmetric, and the messages are over single variables, no information is ever propagated around the cluster graph. This results in the final beliefs being the same as the initial beliefs. This construction is, as a result, completely useless for this task.

Error codes:

- (4.2) [2 points] Incorrect message  $\delta_{3 \rightarrow 6}(X_5)$ .
  - (4.3) [1 points] Fail to point out  $\delta_{3 \rightarrow 6}(X_5)$  equals to 1.
  - (4.4) [2 points] Incorrect or missing  $P(X_4, X_5)$ .
  - (4.5) [1 points] Un-normalized  $P(X_4, X_5)$  or incorrect normalization.
  - (4.6) [2 points] Not getting the conclusion that the belief state isn't updated.
- (c) [6 points] In order to overcome this problem, we will augment our cluster graph  $\mathcal{K}_1$  by *adding clusters* to create a new cluster graph  $\mathcal{K}_2$ . In  $\mathcal{K}_2$ , the factors will be assigned to the same clusters that they were in  $\mathcal{K}_1$ , but the graph will also contain clusters over  $2 \times 2$  neighborhoods of pixels. In addition to adding these clusters and any necessary edges, you may remove edges that appeared in  $\mathcal{K}_1$ . Draw the cluster graph  $\mathcal{K}_2$  that we will use for this round of inference. Compute the fixed-point (converged) messages that the new clusters will send to cluster  $C_3$ , in terms of the  $\alpha$ 's.

**Answer:** The cluster graph that we use is shown in Figure 7.

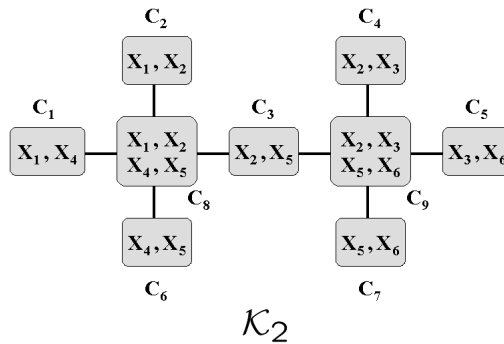


Figure 7: Markov Network for image segmentation.

Note that this is, in fact, a clique tree. Exact inference is possible. We will therefore get an exact result in this case. The variables in each sepset are both of the variables in the pairwise cluster that the edge touches (each edge connects a pairwise to a

four-variable cluster). Because we are now sending messages over pairs of variables, information will propagate, and we will get a meaningful segmentation.

The message sent from clique  $C_8$  to  $C_3$  is computed using:

$$\begin{aligned} \delta_{8 \rightarrow 3} &= \sum_{X_1, X_4} \begin{bmatrix} \alpha_{1,2} & 1 \\ 1 & \alpha_{1,2} \end{bmatrix} \cdot \begin{bmatrix} \alpha_{1,4} & 1 \\ 1 & \alpha_{1,4} \end{bmatrix} \cdot \begin{bmatrix} \alpha_{4,5} & 1 \\ 1 & \alpha_{4,5} \end{bmatrix} \\ &= \sum_{X_1, X_4} \begin{bmatrix} & 00 & 01 & 10 & 11 \\ 00 & \alpha_{1,2}\alpha_{1,4}\alpha_{4,5} & \alpha_{1,4}\alpha_{4,5} & \alpha_{4,5} & \alpha_{1,2}\alpha_{4,5} \\ 01 & \alpha_{1,2}\alpha_{1,4} & \alpha_{1,4} & 1 & \alpha_{1,2} \\ 10 & \alpha_{1,2} & 1 & \alpha_{1,4} & \alpha_{1,2}\alpha_{1,4} \\ 11 & \alpha_{1,2}\alpha_{4,5} & \alpha_{4,5} & \alpha_{1,4}\alpha_{4,5} & \alpha_{1,2}\alpha_{1,4}\alpha_{4,5} \end{bmatrix} \\ &= \begin{bmatrix} \alpha_{1,2}\alpha_{1,4}\alpha_{4,5} + \alpha_{1,2} + \alpha_{4,5} + \alpha_{1,4} & \alpha_{1,2}\alpha_{1,4} + \alpha_{1,2}\alpha_{4,5} + 1 + \alpha_{1,4}\alpha_{4,5} \\ \alpha_{1,4}\alpha_{4,5} + 1 + \alpha_{1,2}\alpha_{4,5} + \alpha_{1,2}\alpha_{1,4} & \alpha_{1,4} + \alpha_{4,5} + \alpha_{1,2} + \alpha_{1,2}\alpha_{1,4}\alpha_{4,5} \end{bmatrix} \end{aligned}$$

Similarly, we have

$$\delta_{9 \rightarrow 3} = \begin{bmatrix} \alpha_{2,3}\alpha_{3,6}\alpha_{5,6} + \alpha_{2,3} + \alpha_{5,6} + \alpha_{3,6} & \alpha_{2,3}\alpha_{3,6} + \alpha_{2,3}\alpha_{5,6} + 1 + \alpha_{3,6}\alpha_{5,6} \\ \alpha_{3,6}\alpha_{5,6} + 1 + \alpha_{2,3}\alpha_{5,6} + \alpha_{2,3}\alpha_{3,6} & \alpha_{3,6} + \alpha_{5,6} + \alpha_{2,3} + \alpha_{2,3}\alpha_{3,6}\alpha_{5,6} \end{bmatrix}$$

Note that this message is still symmetric! This means that we can renormalize to consider a new  $\hat{\alpha}$  that defines the message.

Error codes:

- (4.7) [1 points] Incorrect graph  $\mathcal{K}_2$
  - (4.8) [4 points] Incorrect or incomplete message  $\delta_{8 \rightarrow 3}$ .
  - (4.9) [1 points] Minor errors on the result of  $\delta_{8 \rightarrow 3}$ .
  - (4.10) [1 points] Missing or incorrect message  $\delta_{9 \rightarrow 3}$
- (d) [5 points] We will now think about how such a construction will scale to larger images. We can start by trying to extend the form of cluster graph  $\mathcal{K}_2$  over an image grid of size  $N \times N$ . Can we use the same architecture as described above to create a cluster graph  $\mathcal{K}_3$  over such a grid? How many clusters would  $\mathcal{K}_3$  contain in the case of an  $N \times N$  image? If this cluster graph will work, prove it. If not, provide a counterexample and explain why loopy belief propagation cannot be applied.

**Answer:** Figure 8 shows a proposal for what  $\mathcal{K}_3$  might look like, for the case of a  $3 \times 3$  image.

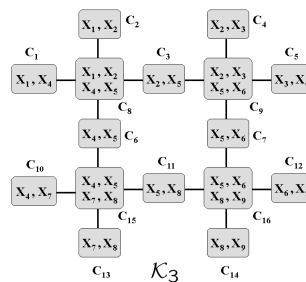


Figure 8: Markov Network for image segmentation.

In general, a graph created using such a method will have  $N(N-1)$  pairwise clusters for vertical edges, the same number for horizontal edges, and  $(N-1)(N-1)$  clusters over four variables in a grid cell. The total number of clusters is therefore  $2N(N-1) + (N-1)^2 = 3N^2 - 4N + 1$ .

Unfortunately, we notice that this cluster graph violates our relaxed running intersection property. Note that  $X_5$  appears all the way around a loop in the middle of the cluster graph. As a result, loopy belief propagation cannot be applied in this case. In order to convert this into a valid cluster graph, we must remove  $X_5$  from one of the sepsets in that central loop.

Error codes:

(4.11) [1 points] Incorrect graph  $\mathcal{K}_3$ .

(4.12) [3 points] Wrong conclusion that it will work.

(4.13) [2 points] Wrong reasoning on why it won't work.

(4.14) [2 points] Incorrect number of clusters that  $\mathcal{K}_3$  would contain.

## 5. Data Association and Collapsed MCMC [20 points]

Consider a data association problem with the airplane tracking application:

We have a set of  $K$  sensor measurements blips on a radar screen  $\mathcal{U} = \{u_1, \dots, u_K\}$  and another set of  $M$  airplanes  $\mathcal{V} = \{v_1, \dots, v_M\}$ , and we wish to map  $\mathcal{U}$ 's to  $\mathcal{V}$ 's. We introduce a set of correspondence variables  $\mathcal{C} = \{C_1, \dots, C_K\}$  such that  $Val(C_i) = \{1, \dots, M\}$ . Here,  $C_i = j$  indicates that  $u_i$  is matched to  $v_j$  (The fact that each variable  $C_i$  takes on only a single value implies that each measurement is derived from only a single object. But the mutual exclusion constraints in the other direction are not forced.). In addition, for each  $u_i$ , we have a set of readings denoted as a vector  $\mathbf{B}_i = (B_{i1}, B_{i2}, \dots, B_{iL})$ ; for each  $v_j$ , we have a three dimensional random vector  $\mathbf{A}_j = (A_{j1}, A_{j2}, A_{j3})$  corresponding to the location of the airplane  $v_j$  in the space. (Assume that  $|Val(A_{jk})| = d$ , which is not too large in the sense that summing over all values of  $\mathbf{A}_j$  is tractable.)

Now suppose that we have a prior distribution over the location of  $v_j$ , i.e., we have  $P(\mathbf{A}_j)$ , and we have observed all  $\mathbf{B}_i$ , and a set of  $\phi_{ij}(\mathbf{A}_j, \mathbf{B}_i, C_i)$  such that  $\phi_{ij}(\mathbf{a}_j, \mathbf{b}_i, C_i) = 1$  for all  $\mathbf{a}_j, \mathbf{b}_i$  if  $C_i \neq j$ . The model contains no other potentials.

We wish to compute the posterior over  $\mathbf{A}_j$  using collapsed Gibbs sampling, where we sample the  $C_i$ 's but maintain a closed form posterior over the  $\mathbf{A}_j$ 's.

- (a) [1 points] Briefly explain why sampling the  $C_i$ 's would be a good idea.

**Answer:**

To compute the posterior over all joint  $\mathbf{A}_j$ 's or a single  $\mathbf{A}_j$  using exact inference is hard, since it requires exponentially large computation. We use sampling based method to approximate the posterior by sampling  $C_i$ 's resulting in the conditional independence between  $\mathbf{A}_j$ 's given  $\mathbf{B}_i$ 's and  $C_i$ 's. This factorization gives us a closed form over each  $\mathbf{A}_j$ , which makes the computation tractable. Moreover, since  $C_i$ 's serve as selector variables, the context after sampling them further reduces many factors to be uniform (context independence:  $\mathbf{A}_j$  doesn't change the belief over  $\mathbf{B}_i$  if  $C_i \neq j$ ). In addition, the space over the joint  $\mathbf{A}_j$ 's ( $d^{3M}$ ) might be larger than that of  $C_i$ 's ( $M^K$ ), since  $d$  is usually much larger than  $M$  and  $K$ , and  $M$  is likely to be larger than  $K$ . Sampling in a lower dimensional space is better in terms of smaller number of samples and better approximation.

Error codes:

- (5.1) [0.5 points] Only compared the sampling space but didn't mention the conditional independence between  $\mathbf{A}_j$ 's given  $C_i$ 's.
- (b) [6 points] Show clearly the sampling distribution for the  $C_i$  variables and the corresponding Markov chain kernels.

**Answer:**

Let  $\Phi$  denote the set of all factors (note that  $P(\mathbf{A}_j)$ 's are also the factors in our model), then the distribution over  $C_i$ 's which we want to sample from is

$$P(C_1, C_2, \dots, C_K | \mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_K) = P_{\Phi[\bar{\mathbf{b}}]}(C_1, C_2, \dots, C_K) \quad (2)$$

$$= \frac{1}{Z} \sum_{\mathbf{A}'_s} \prod_j P(\mathbf{A}_j) \prod_i \phi_{ij}(\mathbf{A}_j, \mathbf{b}_i, C_i) \quad (3)$$

where  $Z$  is the partition function. In order to get samples from the posterior above, we can sample from a Markov chain whose stationary distribution is our target. To do so, we define the Markov chain kernel for each  $C_k$  as:

$$P_{\Phi[\bar{\mathbf{b}}]}(C_k | \mathbf{c}_{-k}) = \frac{P_{\Phi[\bar{\mathbf{b}}]}(C_k, \mathbf{c}_{-k})}{\sum_{C'_k} P_{\Phi[\bar{\mathbf{b}}]}(C'_k, \mathbf{c}_{-k})}. \quad (4)$$

For each value of  $C_k = c_k, c_k = 1, 2, \dots, M$ , we have

$$P_{\Phi[\bar{\mathbf{b}}]}(C_k = c_k | \mathbf{c}_{-k}) = \frac{P_{\Phi[\bar{\mathbf{b}}]}(c_k, \mathbf{c}_{-k})}{P_{\Phi[\bar{\mathbf{b}}]}(\mathbf{c}_{-k})} \quad (5)$$

$$= \frac{1}{Z} \sum_{\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_M} P_{\Phi[\bar{\mathbf{b}}]}(c_k, \mathbf{c}_{-k}, \mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_M) \quad (6)$$

$$= \frac{1}{Z} \sum_{\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_M} \prod_{j=1}^M P(\mathbf{A}_j) \prod_{i:c_i=j} \phi_{ij}(\mathbf{A}_j, \mathbf{b}_i, j) \quad (7)$$

$$= \frac{1}{Z} \prod_{j=1}^M \sum_{\mathbf{A}_j} \left( P(\mathbf{A}_j) \prod_{i:c_i=j} \phi_{ij}(\mathbf{A}_j, \mathbf{b}_i, j) \right) \quad (8)$$

$$= \frac{1}{Z} \prod_{j=1}^M \sum_{\mathbf{A}_j} \Psi_{j, [c_1, c_2, \dots, c_K]}(\mathbf{A}_j) \quad (9)$$

where

$$\Psi_{j, [c_1, c_2, \dots, c_K]}(\mathbf{A}_j) = P(\mathbf{A}_j) \prod_{i:c_i=j} \phi_{ij}(\mathbf{A}_j, \mathbf{b}_i, j) \quad (10)$$

and

$$Z = \sum_{c_k} \prod_{j=1}^M \sum_{\mathbf{A}_j} \Psi_{j, [c_k, \mathbf{c}_{-k}]}(\mathbf{A}_j) \quad (11)$$

Note that to the summation over  $\mathbf{A}_j$ 's can be pushed into the product so that when we sum over a specific  $\mathbf{A}_j$ , the summation is only over a factor  $\Psi_{j, [c_1, c_2, \dots, c_K]}(\mathbf{A}_j)$

whose scope only contains  $\mathbf{A}_j$ . And as mentioned in the question, we assume that summing over a single  $\mathbf{A}_j$  is tractable.

Error codes:

- (5.2) **[3 points]** Missing  $P(\mathbf{A}_j)$ . This error code also applies to the following two parts.
  - (5.3) **[4 points]** Didn't show why the computation is tractable.
  - (5.4) **[5 points]** The basic expression for the kernel is wrong.
  - (5.5) **[2 points]** Missing normalization. This error code also applies to the following two parts.
  - (5.9) **[2 points]** The product in the denominator or numerator has grouped by  $\mathbf{A}_j$ , but the sum wasn't pushed in to show the summation over all  $\mathbf{A}_j$ 's is tractable.
- (c) **[8 points]** Give a closed form equation for the distribution over the  $\mathbf{A}_j$  variables given the assignment to the  $C_i$ 's. (A closed form equation must satisfy two criteria: it must contain only terms whose values we have direct access to, and it must be tractable to compute.)

**Answer:**

Since  $A_j$ 's are independent given all  $C_i$ 's, the joint distribution over  $A_j$ 's can be factorized as the product of the marginal distributions over each single  $A_j$ . Therefore, we only specify the marginal distribution over a single  $A_j$ :

$$P(\mathbf{A}_j | \bar{\mathbf{b}}, \bar{\mathbf{c}}) = \frac{1}{Z} P(\mathbf{A}_j) \prod_{i:c_i=j} \phi_{ij}(\mathbf{A}_j, \mathbf{b}_i, c_i). \quad (12)$$

where,  $Z = \sum_{\mathbf{A}_j} P(\mathbf{A}_j) \prod_{i:c_i=j} \phi(\mathbf{A}_j, \mathbf{b}_i, c_i)$ ,  $\bar{\mathbf{b}} = \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_K\}$  and  $\bar{\mathbf{c}} = \{c_1, \dots, c_K\}$ .

Error codes:

- (5.6) **[8 points]** The whole expression is wrong.
  - (5.8) **[1 points]** Minor errors. This applies to any part in the question.
  - (5.11) **[2 points]** Wrote the joint distribution, but didn't show the partition function which involves the summation over all  $A_j$ 's is tractable.
- (d) **[5 points]** Show the equation for computing the posterior over  $\mathbf{A}_j$  based on the two steps above.

**Answer:**

After the Markov chain converges to its stationary distribution, i.e., the posterior over  $C_i$ 's, we can collect  $M$  instances each of which is denoted by  $\bar{\mathbf{c}}[m]$ ,  $m = 1, 2, \dots, M$ , and with their distributional parts  $P(\mathbf{A}_j | \bar{\mathbf{b}}, \bar{\mathbf{c}}[m])$ , we can estimate the posterior over  $\mathbf{A}_j$  as:

$$\hat{P}(\mathbf{A}_j = \mathbf{a}_j | \bar{\mathbf{b}}) = \frac{1}{M} \sum_{m=1}^M \frac{1}{Z[m]} P(\mathbf{a}_j) \prod_{i:c_i[m]=j} \phi_{ij}(\mathbf{a}_j, \mathbf{b}_i, c_i[m]) \quad (13)$$

$$= \frac{1}{M} \sum_{m=1}^M \frac{P(\mathbf{a}_j) \prod_{i:c_i[m]=j} \phi_{ij}(\mathbf{a}_j, \mathbf{b}_i, c_i[m])}{\sum_{\mathbf{a}'_j} P(\mathbf{a}'_j) \prod_{i:c_i[m]=j} \phi_{ij}(\mathbf{a}'_j, \mathbf{b}_i, c_i[m])} \quad (14)$$

where  $\mathbf{a}_j$  is a specific value that  $\mathbf{A}_j$  takes.

Error codes:

- (5.7) [5 points] The whole expression is wrong.
- (5.10) [2 points] The expression is correct, but the text explanation is wrong or unclear.

**Estimate:** 1.5 pages

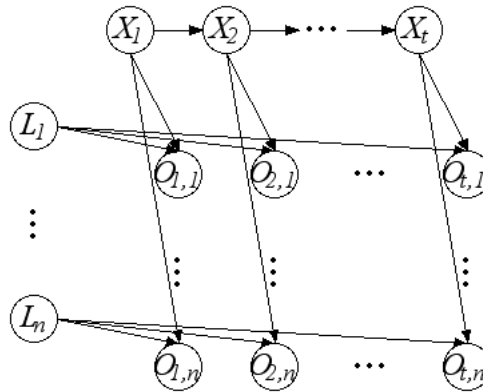
**6. Collapsed MCMC for Particle Filtering [22 points]**

Consider a robot moving around in a space with  $n$  stationary landmarks. The positions of the robot and the landmarks are unknown. The position of the robot (also called a *pose*) at time  $t$  is denoted  $X^{(t)}$ . The (fixed) position of landmark  $i$  is denoted  $L_i$ . At each time  $t$ , the robot obtains a (noisy) measurement  $O_i^{(t)}$  of its current position relative to each landmark  $k$ . The model is parameterized using two components. Robot poses evolve via a *motion model*:  $P(X^{(t+1)} | X^{(t)})$ . Sensor measurements are governed by a *measurement model*:  $P(O_i^{(t)} | X^{(t)}, L_i)$ . Both the motion model and the measurement model are known.

Denote  $X^{(1:t)} = \{X^{(1)}, \dots, X^{(t)}\}$ ,  $\mathbf{L} = \{L_1, \dots, L_n\}$ , and  $\mathbf{O}^{(1:t)} = \{O_1^{(1)}, \dots, O_n^{(1)}, O_1^{(2)}, \dots, O_n^{(t)}\}$ . We want to use the measurements  $\mathbf{O}^{(1:t)}$  to localize both the robot and landmarks; i.e., we want to compute  $P(X^{(t)}, \mathbf{L} | \mathbf{o}^{(1:t)})$ .

- (a) [2 points] Draw a DBN model for this problem. What is the dimension (number of variables) of the belief state for one time slice in this problem? (Assume that the belief state at time  $t$  is over all variables about which we are uncertain at time  $t$ .)

**Answer:** The simplest way to represent this is to extend our model for a DBN to allow some static variables, which are not part of each timeslice. We can think of them as existing before the first timeslice. Unrolled, it looks like this:



But to store it compactly, whereas we previously needed only representations of  $\mathcal{B}_0$  and  $\mathcal{B}_\rightarrow$ , representing the initial state and transition model, here we have a third network, representing the static variables, which we will denote  $\mathcal{B}_s$ . Note that in principle, with this extension to the notion of a DBN, the static variables could actually be a standard Bayes net, with some of them depending on each other; but in our case they will not, so their CPDs have no parents to condition on. However, the CPDs in  $\mathcal{B}_\rightarrow$  can then use the variables in  $\mathcal{B}_s$  as parents, in addition to any inter-timeslice and intra-timeslice edges.

Of course, we could alternately represent it by including all the static variables in the 2-TBN, each with a deterministic CPD simply propagating its values forward, never changing.

A belief state now corresponds to a set of beliefs for the static variables as well as the state variables of the current timeslice, so in our DBN, the belief state will have a dimension of  $n + 1$ .

Error codes:

(6.1) **[2 points]** Not a correct graph.

(6.2) **[1 points]** Not getting  $n + 1$  for dimension of belief state.

We want to use particle filtering to solve our tracking problem, but particle filtering tends to break down in very high dimensional spaces. To address this issue, we plan to use particle filtering with distributional particles. For our localization and mapping problem, we have two obvious ways of constructing these distributional particles. We now consider one such approach in detail, and briefly consider the implications of the other.

(b) **[15 points]** We select the robot pose trajectory  $X^{(1:t)}$  as the set of sampled variables, and maintain a closed form distribution over  $\mathbf{L}$ . Thus, at time  $t$  we have a set of weighted particles, each of which specifies a full sampled trajectory  $\mathbf{x}^{(1:t)}[m]$ , a distribution over landmark locations  $P_m^{(t)}(\mathbf{L})$ , and a weight  $w^{(t)}[m]$ .

i. The distribution over  $\mathbf{L}$  is still exponentially large in principle. Show how you can represent it compactly in closed form. (Hint: Write out the full form of  $P_m^{(t)}(\mathbf{L})$  recalling the definition of distributional particles, and note that our samples represent entire pose trajectories  $X^{(1:t)}$ .)

**Answer:** Since each particle instantiates all of  $x^{(1:t)}[m]$  and  $\mathbf{o}^{(1:t)}[m]$ , there is no active trail from  $L_i$  to  $L_j$  for  $i \neq j$ , and so the  $L_i$  variables are all independent, given a particle. Also, each landmark's location is independent of the observations of the other landmarks, given a trajectory. Thus we can decompose our distribution over the landmarks' locations as the following product of marginals:

$$\begin{aligned} P_m^{(t)}(\mathbf{L}) &= P(\mathbf{L} \mid x^{(1:t)}[m], \mathbf{o}^{(1:t)}[m]) \\ &= \prod_i P(L_i \mid x^{(1:t)}[m], \mathbf{o}^{(1:t)}[m]) \\ &= \prod_i P(L_i \mid x^{(1:t)}[m], o_i^{(1:t)}[m]) \\ &= \prod_i P_m^{(t)}(L_i) \end{aligned}$$

Note that it is not in closed form yet. We will show how to compute  $P_m^{(t)}(L_i)$  in part ii.

Error codes:

(6.3) **[3 points]** Fail to properly decompose as product of marginals.

ii. Describe how to perform a forward propagation step in our distributional particle filtering algorithm. Specifically show how to:

- generate a new set of particles for time  $t + 1$

- compute the weight  $w^{(t+1)}[m]$  of each particle;
- compute the distribution  $P_m^{(t+1)}(\mathbf{L})$ .

**Answer:** To generate new particles, we basically just follow standard particle filtering – normalize the weights of the particles from the previous timeslice; select a particle under that normalized distribution; then sample its forward propagation based on the motion model  $P(X^{(t+1)} | X^{(t)})$ .

To see how to compute our new distribution over  $\mathbf{L}$ , we first make the following observation:

$$\begin{aligned}
 P_m^{(t+1)}(L_i) &= P(L_i | x^{(1:t+1)}[m], o_i^{(1:t+1)}[m]) \\
 &= \frac{1}{Z} P(o_i^{(1:t+1)}[m] | L_i, x^{(1:t+1)}[m]) P(L_i | x^{(1:t+1)}[m]) \\
 &= \frac{1}{Z} P(o_i^{(1:t+1)}[m] | L_i, x^{(1:t+1)}[m]) P(L_i) \\
 &= \frac{1}{Z} P(L_i) \prod_{t'=1}^{t+1} P(o_i^{(t')}[m] | L_i, x^{(t')}[m])
 \end{aligned}$$

for some normalizing constant

$$Z = \sum_{l_i} P(l_i) \prod_{t'=1}^{t+1} P(o_i^{(t')}[m] | l_i, x^{(t')}[m])$$

which does not depend on  $L_i$ . Note that  $P(L_i)$  is the prior for the location of the landmark, as encoded by  $\mathcal{B}_s$  (or  $\mathcal{B}_0$  if using deterministic CPDs to propagate the  $L_i$  variables through all timeslices); and each factor of the product is just a lookup in the known measurement model. However, since

$$P_m^{(t)}(L_i) = \frac{1}{Z'} P(L_i) \prod_{t'=1}^t P(o_i^{(t')}[m] | L_i, x^{(t')}[m])$$

for some other normalizing constant

$$Z' = \sum_{l_i} P(l_i) \prod_{t'=1}^t P(o_i^{(t')}[m] | l_i, x^{(t')}[m])$$

which similarly does not depend on  $L_i$ , then

$$P_m^{(t+1)}(L_i) = \frac{1}{Z''} P_m^{(t)}(L_i) P(o_i^{(t+1)}[m] | L_i, x^{(t+1)}[m])$$

for some normalizing constant  $Z'' = \frac{Z}{Z'}$  which does not depend on  $L_i$ . Thus, to update our beliefs about the location of the landmarks, we just multiply in the probability of the new observations from the measurement model and renormalize.

Having updated our distribution over the locations of the landmarks, we now need to compute the weight for the particle. In standard particle filtering, we use the likelihood of the new observations given the complete particle; here, our particle does not contain the location of the landmarks, so we must compute our weights using the other given information. We can consider the reweighting to be given all of the

previously sampled variables together with all observations. Using this insight, we can compute the particles by marginalizing out the landmark locations:

$$\begin{aligned}
w^{(t+1)}[m] &= P(\mathbf{o}^{(t+1)}[m] \mid x^{(1:t+1)}[m], \mathbf{o}^{(1:t)}[m]) \\
&= \sum_{\mathbf{l} \in \text{Val}(\mathbf{L})} P(\mathbf{o}^{(t+1)}[m], \mathbf{l} \mid x^{(1:t+1)}[m], \mathbf{o}^{(1:t)}[m]) \\
&= \sum_{\mathbf{l} \in \text{Val}(\mathbf{L})} P(\mathbf{l} \mid x^{(1:t+1)}[m], \mathbf{o}^{(1:t)}[m]) P(\mathbf{o}^{(t+1)}[m] \mid x^{(1:t+1)}[m], \mathbf{o}^{(1:t)}[m], \mathbf{l}) \\
&= \sum_{\mathbf{l} \in \text{Val}(\mathbf{L})} P(\mathbf{l} \mid x^{(1:t)}[m], \mathbf{o}^{(1:t)}[m]) P(\mathbf{o}^{(t+1)}[m] \mid x^{(t+1)}[m], \mathbf{l}) \\
&= \sum_{\mathbf{l} \in \text{Val}(\mathbf{L})} \prod_{i=1}^n P(l_i \mid x^{(1:t)}[m], o_i^{(1:t)}[m]) P(o_i^{(t+1)}[m] \mid x^{(t+1)}[m], l_i) \\
&= \sum_{\mathbf{l} \in \text{Val}(\mathbf{L})} \prod_{i=1}^n P_m^{(t)}(l_i) P(o_i^{(t+1)}[m] \mid x^{(t+1)}[m], l_i)
\end{aligned}$$

Thus we see how to compute the weight for the particle in the new timeslice using the normalized distribution  $P_m^{(t)}(\mathbf{L})$  from the previous timeslice, together with our measurement model  $P(o_i^{(t+1)}[m] \mid x^{(t+1)}[m], l_i)$ .

Error codes:

(6.4) [3 points] Incorrect method for generating particles.

(6.5) [4 points] Incorrect probabilities.

(6.6) [5 points] Incorrect weights.

- (c) [5 points] As another alternative, we can select the landmark positions  $\mathbf{L}$  as our set of sampled variables. Thus, for each particle  $\mathbf{l}[m]$ , we maintain a distribution  $P_m^{(t)}(X^{(t)})$ . Without describing the details of how to perform a forward propagation step in our distributional particle filtering algorithm, we can think about the effectiveness of such an approach. In this algorithm, what will happen to the particles and their weights eventually (i.e., after a large number of time steps)? What are the necessary conditions for this algorithm to converge to the correct map?

**Answer:** Since we never generate any new assignments  $\mathbf{l}[m]$ , on each forward propagation, with some nonzero probability, one or more of the particles will “die off”. Thus eventually, we will be left with a single particle  $\mathbf{l}[m]$  with a weight of 1.

The single map to which we converge will be the *correct* map only if our initial generation of particles included the correct map among its hypotheses. Since we never generate any new hypotheses, we certainly cannot converge on the right one if we failed to start with it as a hypothesis. However, although this is necessary, it is not sufficient, since even if we start with the correct hypothesis as an available option, it is possible that by chance we might discard it, especially early on when, with few observations, its weight may not be much more than that of the other hypotheses.

As there are exponentially many assignments to  $\mathbf{L}$ , unless we have a very well-specified prior over them, it is highly unlikely the correct solution will be generated as an initial hypothesis.

Error codes:

- (6.7) [**2 points**] Not getting that we eventually must end up with just one particle.
- (6.8) [**3 points**] Not getting that we only converge to the correct map if we happen to have started with the correct map as one of our initial particles.
- (6.9) [**3 points**] Right general idea, but need to be more specific.