1 Inference in Graphical Models

Motivation

Below we start with defining hidden Markov models, and highlight their connection with robot localization. We then show how to efficiently perform inference by converting the Bayes net (with evidence) to a factor graph. We show both full posterior inference, MPE, and MAP estimation. After that, we generalize to completely general inference in Bayes nets, introducing the elimination algorithm.

1.1 Bayes Filtering

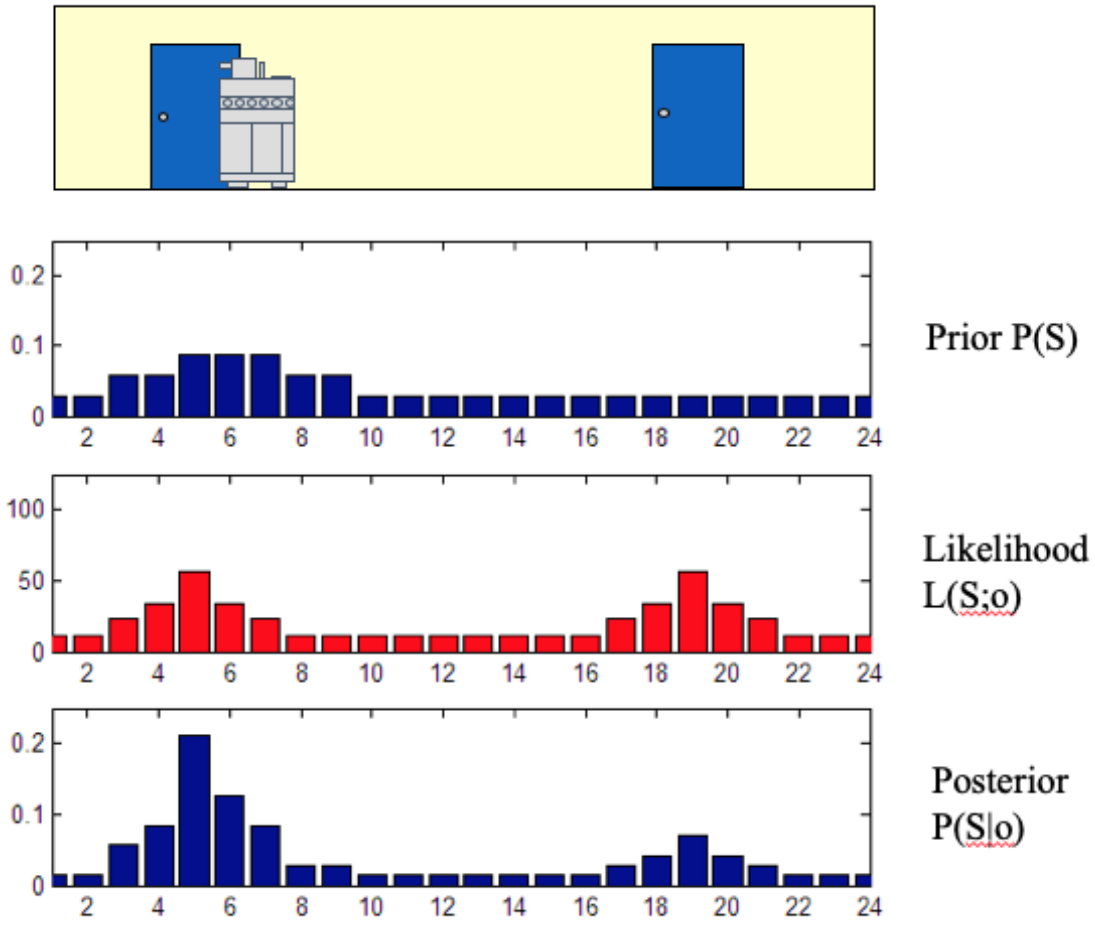

Figure 1: One-dimensional example of Markov localization.

First we will start off with the Bayes filter, which is a very simple recursive inference scheme. In this case we present the discrete version, which was used in robotics for localization under the name **Markov localization**. But the same general scheme underlies the venerable Kalman filter, and Monte Carlo Localization, which is based on a particle filter. The latter two are typically used in continuous settings, so we will revisit them later.

In the Bayes filter, there are two phases:

1. The prediction phase

$$P(S_t) = \sum_{s_{t-1}} P(S_t|s_{t-1}, a_{t-1})P(s_{t-1})$$

   where we predict a prior distribution on the current state S_t from a distribution on the previous state $P(s_{t-1})$ and a given action $A_{t-1} = a_{t-1}$. This is done by marginalizing out the previous state S_{t-1}.

2. The measurement phase:

$$\begin{aligned} P(S_t|O_t = o_t) &\propto P(o_t|S_t)P(S_t) \\ &\propto L(S_t|o_t)P(S_t) \end{aligned}$$

   where we upgrade this prior to a posterior via Bayes law, given an observation $O_t = o_t$.

Above and where it is clear from context we abbreviate $P(S_t|S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1})$ as $P(S_t|s_{t-1}, a_{t-1})$, in the interest of a more succinct notation. But always remember that S_t is a random variable, and s_t is a *value* assigned to the random variable.

The Bayes filter above is great if you are only interested in the posterior distribution over the current state. However, if you are interested in the posterior over the entire state *trajectory*, we need to consider more general inference schemes. To this end, we introduce hidden Markov models in the next section.

1.2 Hidden Markov Models

A **hidden Markov model** or HMM is a dynamic Bayes net that has two types of variables: states $\mathcal{X}$ and measurements $\mathcal{Z}$. The states $\mathcal{X}$ are connected sequentially and satisfy the what is called the **Markov property**: the probability of a state is only dependent on the value of the previous state. We call this a **Markov chain** of *hidden* states, as typically we cannot directly observe their values. Instead, they are indirectly observed through the measurements $\mathcal{Z}$, where we have one measurement per hidden state. When these two properties are satisfied, we call this probabilistic model a hidden Markov model.

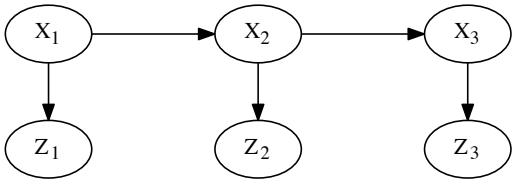

Figure 2: An HMM, unrolled over three time-steps, represented by a Bayes net.

Figure 2 shows an example of an HMM for three time steps, i.e.., $\mathcal{X} = \{X_1, X_2, X_3\}$ and $\mathcal{Z} = \{Z_1, Z_2, Z_3\}$. As we discussed, in a Bayes net each node is associated with a conditional distribution: the Markov chain has the prior $P(X_1)$ and transition probabilities $P(X_2|X_1)$ and

$P(X_3|X_2)$, whereas the measurements Z_t depend only on the state X_t, modeled by conditional distributions $P(Z_t|X_t)$. In other words, the Bayes net encodes the following joint distribution $P(\mathcal{X}, \mathcal{Z})$:

$$P(\mathcal{X}, \mathcal{Z}) = P(X_1)P(Z_1|X_1)P(X_2|X_1)P(Z_2|X_2)P(X_3|X_2)P(Z_3|X_3)$$

Note that we can also write this more succinctly as

$$P(\mathcal{X}, \mathcal{Z}) = P(\mathcal{Z}|\mathcal{X})P(\mathcal{X}) \tag{1}$$

where

$$P(\mathcal{X}) = P(X_1, X_2, X_3) = P(X_1)P(X_2|X_1)P(X_3|X_2)$$

is the prior over state *trajectories*.

We want to infer the states $\mathcal{X}$ given values $\mathfrak{z} = \{z_1, z_2, z_3\}$ for $\mathcal{Z}$, but we cannot know them exactly. As we saw, one way to do this is to simply apply Bayes' rule to Equation 1, and get an expression for the *posterior* probability distribution over the state trajectory $\mathcal{X}$, given the measurements $\mathcal{Z} = \mathfrak{z}$:

$$\begin{aligned} P(\mathcal{X}|\mathcal{Z}) &\propto P(\mathcal{Z} = \mathfrak{z}|\mathcal{X})P(\mathcal{X}) \\ &= L(\mathcal{X}; \mathcal{Z} = \mathfrak{z})P(\mathcal{X}) \end{aligned} \tag{2}$$

where $P(\mathcal{X})$ is given above, and the likelihood $(\mathcal{X}; \mathcal{Z} = \mathfrak{z})$ of $\mathcal{X}$ given $\mathcal{Z} = \mathfrak{z}$ is the following function of $\mathcal{X}$:

$$\begin{aligned} L(\mathcal{X}; \mathcal{Z} = \mathfrak{z}) &= P(z_1|X_1)P(z_2|X_2)P(z_3|X_3) \\ &= L(X_1; Z_1)L(X_2; Z_2)L(X_3; Z_3) \end{aligned}$$

As we saw before, a naive implementation for finding the most probable explanation (MPE) for $\mathcal{X}$ would tabulate all possible trajectories $\mathcal{X}$, which is exponential in the number of states, and calculate the posterior (2) for each one. Not only is this computationally prohibitive for long trajectories, but intuitively it is clear that for many of these trajectories we are computing the same values over and over again. In fact, there are three different approaches to speed this up:

1. Branch & bound
2. Dynamic programming
3. Inference using factor graphs

Branch and bound is a powerful technique but will not generalize to continuous variables, like the other two approaches will. And, we will see that dynamic programming, which underlies the classical inference algorithms in the HMM literature, is just a special case of the last approach. Hence, here we will dive in and immediately go for the most general approach: inference in factor graphs.

1.3 Factor Graphs

Again referring to the example from Figure 2, let us consider the posterior (2) again. Since the measurements $\mathcal{Z}$ are *known*, the posterior is proportional to the product of six **factors**, three of which derive from the the Markov chain, and three likelihood factors defined as before:

$$P(\mathcal{X}|\mathcal{Z}) \propto P(X_1)L(X_1; z_1)P(X_2|X_1)L(X_2; z_2)P(X_3|X_2)L(X_3; z_3) \tag{3}$$

Note that in (3) some of the factors depend on just one hidden variable, for example $L(X_2; z_2)$, whereas others depend on two variables, e.g., the transition model $P(X_3|X_2)$. Finally, note that once we are given the measurements $\mathcal{Z}$, they merely function as known parameters in the likelihoods $L(X_t; z_t)$.

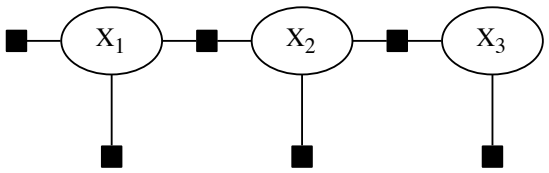

Figure 3: An HMM with observed measurements, unrolled over time, represented as a factor graph.

This motivates a different graphical model, a **factor graph**, in which we only represent the *hidden* variables X_1, X_2, and X_3, connected to factors that encode probabilistic information on them, as in Figure 3. It should be clear from the figure that the connectivity of a factor graph encodes, for each factor ϕ_i, which subset of variables $\mathcal{X}_i$ it depends on. We write:

$$\phi(\mathcal{X}) = \phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2)\phi_4(X_2)\phi_5(X_2, X_3)\phi_6(X_3)$$

where the factors are defined to correspond one-to-one to Equation 3. For example, $\phi_6(X_3) \stackrel{\Delta}{=} L(X_3; z_3)$. All measurements are associated with unary factors, and the Markov chain is associated mostly with binary factors, with the exception of the unary factor $\phi_1(X_1)$. Note that in defining the factors we can omit any normalization factors, which in many cases results in computational savings.

Formally a factor graph is a bipartite graph $F = (\mathcal{U}, \mathcal{V}, \mathcal{E})$ with two types of nodes: **factors** $\phi_i \in \mathcal{U}$ and **variables** $X_j \in \mathcal{V}$. Edges $e_{ij} \in \mathcal{E}$ are always between factor nodes and variables nodes. The set of variable nodes adjacent to a factor ϕ_i is written as $\mathcal{N}(\phi_i)$, and we write $\mathcal{X}_i$ for an assignment to this set. With these definitions, a factor graph F defines the factorization of a global function $\phi(\mathcal{X})$ as

$$\phi(\mathcal{X}) = \prod_i \phi_i(\mathcal{X}_i). \tag{4}$$

In other words, the independence relationships are encoded by the edges e_{ij} of the factor graph, with each factor ϕ_i a function of *only* the variables $\mathcal{X}_i$ in its adjacency set $\mathcal{N}(\phi_i)$.

1.4 Converting Bayes Nets into Factor Graphs.

Every Bayes net can be trivially converted to a factor graph. Recall that every node in a Bayes net denotes a conditional density on the corresponding variable and its parent nodes. Hence, the conversion is quite simple: every Bayes net node splits in *both* a variable node and a factor node in the corresponding factor graph. The factor is connected to the variable node, as well as the variable nodes corresponding to the parent nodes in the Bayes net. If some nodes in the Bayes net are evidence nodes, i.e., they are given as known variables, we omit the corresponding variable nodes: the known variable simply becomes a fixed parameter in the corresponding factor.

Figure 4: Converting a Bayes net into a factor graph.

Once we convert a Bayes net with evidence into a factor graph where the evidence is all implicit in the factors, we can support a number of different computations. First, given any factor graph defining an unnormalized density $\phi(X)$, we can easily **evaluate** it for any given value, by simply evaluating every factor and multiplying the results. Evaluation opens up the way to **optimization**, e.g., finding the most probable explanation or MPE, as we will do below. In the case of discrete variables, graph search methods can be applied, but we will use a different approach.

While local or global maxima of the posterior are often of most interest, **sampling** from a probability density can be used to visualize, explore, and compute statistics and expected values associated with the posterior. However, the ancestral sampling method we discussed earlier only applies to directed acyclic graphs. There are however more general sampling algorithms that can be used for factor graphs, more specifically Markov chain Monte Carlo (MCMC) methods. One such method is Gibbs sampling, which proceeds by sampling one variable at a time from its conditional density given all other variables it is connected to via factors. This assumes that this conditional density can be easily obtained, which is in fact true for discrete variables.

Below we use factor graphs as the organizing principle for probabilistic inference. In later chapters we will expand their use to continuous variables, and will see that factor graphs aptly describe the independence assumptions and sparse nature of the large nonlinear least-squares problems arising in robotics. But their usefulness extends far beyond that: they are at the core of the sparse linear solvers we use as building blocks, they clearly show the nature of filtering and incremental inference, and lead naturally to distributed and/or parallel versions of robotics.

Exercise

1. Convert the dynamic Bayes net from Figure **??** into a factor graph.

1. Finally, do the same again, but now assume the *states* are given. Reflect on the remarkable phenomenon that happens.

1.5 The Max-Product Algorithm for HMMs

Given a factor graph, the max-product algorithm is an $O(n)$ algorithm to find the maximum probable explanation or MPE.

We will use the example from Figure 3 to give the intuition. To find the MPE for $\mathcal{X}$ we need to *maximize* the product

$$\phi(X_1, X_2, X_3) = \prod \phi_i(\mathcal{X}_i) \tag{5}$$

i.e., the value of the factor graph. The **max-product algorithm** proceeds one variable at a time.

Eliminating X_1

We start by considering the first state X_1, and we form a product factor $\psi(X_1, X_2)$ that only has factors connected to X_1:

$$\psi(X_1, X_2) = \phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2).$$

Note that because one of those factors (the state transition model) is also connected to the second state X_2, the product factor is a function of *both* X_1 and X_2. The key observation in the max-product algorithm is that we can now eliminate X_1 from the problem, by looking at all possible values x_2 of X_2, and creating a lookup table for the best possible value of X_1 :

$$g_1(X_2) = \arg\max_{x_1} \psi(x_1, X_2).$$

We also record the value of the product factor for that maximum, so we can use it down the line for taking into account the consequence of each choice:

$$\tau(X_2) = \max_{x_1} \psi(x_1, X_2).$$

In practice, of course, both steps can be implemented in a single function. We then put this new factor $\tau(X_2)$ back into the graph, essentially summarizing the result of eliminating X_1 from the problem entirely, obtaining the reduced graph

$$\Phi_{2:3} = \tau(X_2)\phi_4(X_2)\phi_5(X_2, X_3)\phi_6(X_3)$$

Eliminating X_2

We now do exactly the same for the state X_2. The product factor $\psi(X_2, X_3)$ only has factors connected to X_2:

$$\psi(X_2, X_3) = \tau(X_2)\phi_4(X_2)\phi_5(X_2, X_3),$$

but now includes the factor $\tau(X_2)$ from the previous step. We can now eliminate X_2 from the problem, by looking at all possible values x_3 of X_3, and creating a lookup table for the best possible value of X_2 :

$$g_2(X_3) = \arg\max_{x_2} \psi(x_2, X_3),$$

and record the value of the product factor for that maximum:

$$\tau(X_3) = \max_{x_2} \psi(x_2, X_3).$$

We then put this new factor $\tau(X_3)$ back into the graph:

$$\Phi_{3:3} = \tau(X_3)\phi_6(X_3)$$

Eliminating X_3

Finally, we eliminate X_3, where the product factor is now the entire remaining graph and only depends on X_3, as all other states have already been eliminated:

$$\psi(X_3) = \tau(X_3)\phi_6(X_3).$$

We again obtain a lookup table,

$$g_3(\emptyset) = \arg\max_{x_3} \psi(x_3),$$

and a new factor:

$$\tau(\emptyset) = \max_{x_1} \psi(x_3).$$

Note however that now the value does not depend on any arguments! This is indicated by making the argument list equal to the empty set $\emptyset$. Indeed, g_3 just tells us what the best value for X_3 is, and τ tells us the corresponding value. Because it incorporates the factors from the previous elimination steps, this will in fact be exactly the solution to Problem 5.

Back-substitution

Once we know the value for X_3, we can simply plug it into the lookup table $g_2(X_3)$ to get the value for X_2, which we can then plug into the lookup table g_1 to get the value for X_1, and we recover the MPE in one single backward pass.

The Entire Algorithm

Algorithm 1 The Max-Product Algorithm for HMMs

function ELIMINATE($\Phi_{1:n}$) ▷ given an HMM with n states
  for $j = 1...n$ **do** ▷ for all states
    $g_j(S_j), \Phi_{j+1:n} \leftarrow \text{EliminateOne}(\Phi_{j:n}, X_j)$ ▷ eliminate X_j
  return $\{g_1(S_1)g(S_2)\dots g_n(\emptyset)\}$ ▷ return DAG of lookup tables

Algorithm 2 Eliminate state X_j from an HMM $\Phi_{j:n}$.

function ELIMINATEONE($\Phi_{j:n}, X_j$) ▷ given reduced graph $\Phi_{j:n}$
  Remove all factors $\phi_i(\mathcal{X}_i)$ that are adjacent to X_j
  $\mathcal{S}(X_j) \leftarrow$ all variables involved excluding X_j ▷ the separator
  $\psi(X_j, S_j) \leftarrow \prod_i \phi_i(\mathcal{X}_i)$ ▷ create the product factor ψ
  $g_j(S_j), \tau(S_j) \leftarrow \psi(X_j, S_j)$ ▷ **perform argmax, max**
  Add the new factor $\tau(S_j)$ back into the graph
  return $g_j(S_j), \Phi_{j+1:n}$ ▷ lookup table and reduced graph

The HMM max-product algorithm for any value of n is given in Algorithm 1, where we used the shorthand notation $\Phi_{j:n} \triangleq \phi(X_j, \dots, X_n)$ to denote a reduced factor graph. The algorithm proceeds by eliminating one hidden state X_j at a time, starting with the complete HMM factor graph $\Phi_{1:n}$. As we eliminate each variable X_j, the function ELIMINATEONE produces a single lookup table $g_j(S_j)$, as well as a reduced factor graph $\Phi_{j+1:n}$ on the remaining variables. After all variables have been eliminated, the algorithm returns a chain of lookup tables that can be used to recover the MPE in reverse elimination order.

1.6 The Elimination Algorithm

There exists a general algorithm that, given any (preferably sparse) factor graph, can compute the corresponding posterior density $p(X|Z)$ on the unknown variables $\mathcal{X}$ in a form that allows easy recovery of the MPE or MAP solutions to the problem. As we saw, a factor graph represents the unnormalized posterior $\phi(X) \propto P(X|Z)$ as a product of factors, typically generated directly from the measurements. The elimination algorithm is a recipe for converting a factor graph back to a Bayes net, but now *only* on the unknown variables $\mathcal{X}$. This then allows for easy MPE inference.

In particular, the **variable elimination** algorithm is a way to factorize any factor graph of the form

$$\phi(\mathcal{X}) = \phi(X_1, \ldots, X_n) \tag{6}$$

into a factored Bayes net probability density of the form

$$p(\mathcal{X}) = p(X_1|S_1)p(X_2|S_2)\ldots p(X_n) = \prod_j p(X_j|S_j), \tag{7}$$

where S_j denotes an assignment to the **separator** $\mathcal{S}(X_j)$ associated with variable X_j under the chosen variable ordering $X_1, \ldots, X_n$. The separator is defined as the set of variables on which X_j is conditioned, after elimination. While this factorization is akin to the chain rule, eliminating a sparse factor graph will typically lead to small separators.

Algorithm 3 The Variable Elimination Algorithm

function ELIMINATE($\Phi_{1:n}$) ▷ given a factor graph on n variables
  for $j = 1...n$ **do** ▷ for all variables
    $p(X_j|S_j), \Phi_{j+1:n} \leftarrow$ EliminateOne($\Phi_{j:n}, X_j$) ▷ eliminate X_j
  return $p(X_1|S_1)p(X_2|S_2)\ldots p(X_n)$ ▷ return Bayes net

Algorithm 4 Eliminate variable X_j from a factor graph $\Phi_{j:n}$.

function ELIMINATEONE($\Phi_{j:n}, X_j$) ▷ given reduced graph $\Phi_{j:n}$
  Remove all factors $\phi_i(\mathcal{X}_i)$ that are adjacent to X_j
  $\mathcal{S}(X_j) \leftarrow$ all variables involved excluding X_j ▷ the separator
  $\psi(X_j, S_j) \leftarrow \prod_i \phi_i(\mathcal{X}_i)$ ▷ create the product factor ψ
  $p(X_j|S_j)\tau(S_j) \leftarrow \psi(X_j, S_j)$ ▷ factorize the product ψ
  Add the new factor $\tau(S_j)$ back into the graph
  return $p(X_j|S_j), \Phi_{j+1:n}$ ▷ Conditional and reduced graph

The elimination algorithm is listed as Algorithm 3, where we used the shorthand notation $\Phi_{j:n} \triangleq \phi(X_j, \ldots, X_n)$ to denote a partially eliminated factor graph. The algorithm proceeds by eliminating one variable X_j at a time, starting with the complete factor graph $\Phi_{1:n}$. As we eliminate each variable X_j, the function ELIMINATEONE produces a single conditional $p(X_j|S_j)$, as well as a reduced factor graph $\Phi_{j+1:n}$ on the remaining variables. After all variables have been eliminated, the algorithm returns the resulting Bayes net with the desired factorization.

1.7 Complexity

The elimination algorithm has exponential complexity in the size of the largest separator.

The elimination ordering can affect the complexity dramatically. Some orderings lead to smaller separators, and unfortunately it is hard to find an optimal ordering - although it can be done for small graphs. A good heuristic is to greedily eliminate the variables with the smallest separator first. Another is to recursively split the graph, and eliminate starting from the leaves of the binary tree that is formed by the splitting process. We will go into more details in a later Chapter.

In the special case of HMMs, and in fact any singly connected graph, the complexity is linear in the number of nodes. The reason is easiest to see for an HMM, as after conversion to a factor graph the graph is just a chain. You can easily prove, by induction, that the size of the separator is always one. It must be, by the way, as the resulting DAG is also singly connected.

1.8 MAP Estimation

Maximum a posteriori estimation is at least as expensive as MPE or calculating the full posterior. Remember that MAP estimation is only interested in a subset of the variables, and we partition the variables into three sets: the variables of interest $\mathcal{X}$, the nuisance variables $\mathcal{Y}$, and the observed variables $\mathcal{Z}$. The elimination algorithm is easy to modify to do MAP estimation, simply *by making sure that the variables of interest $\mathcal{X}$ are eliminated last.*

Why does this work? We can easily see this if we take a "30,000 feet view" of the elimination algorithm. In MAP estimation, we are interested in maximizing $P(\mathcal{X}|\mathcal{Z} = \mathfrak{z})$, but the Bayes net gives us the joint distribution $P(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$. The first step is to instantiate the evidence $\mathfrak{z}$and convert to a factor graph $f(\mathcal{X}, \mathcal{Y}; \mathcal{Z} = \mathfrak{z})$. When we eliminate using the sum-product algorithm, using the elimination order $\mathcal{Y}$,$\mathcal{X}$, we obtain a DAG encoding the resulting posterior as

$$P(\mathcal{Y}|\mathcal{X}, \mathcal{Z} = \mathfrak{z})P(\mathcal{X}|\mathcal{Z} = \mathfrak{z})$$

where $P(\mathcal{X}|\mathcal{Z} = \mathfrak{z})$ is the desired marginal distribution on $\mathcal{X}$. Using max-product, the resulting DAG for the MPE is

$$\pi(\mathcal{Y}|\mathcal{X}, \mathcal{Z} = \mathfrak{z})\pi(\mathcal{X}|\mathcal{Z} = \mathfrak{z})$$

where the lookup table $\pi(\mathcal{X}|\mathcal{Z} = \mathfrak{z})$ corresponds to the MAP estimate.

The higher complexity of MAP estimation derives from the fact that not all elimination orderings are allowed anymore. In particular, the optimal ordering, or even approximately optimal orderings, may all be incompatible with eliminating $\mathcal{X}$ last.

Exercises

1. Do MAP estimation for...
2. Construct a small example where MAP estimation is more expensive than MPE, even when using optimal orderings for both.

Summary

We briefly summarize what we learned in this section:

1. HMMs.
2. Factor graphs