Contents

1 Inference in Graphical Models

Motivation

Bayes nets are great for *modeling*, but for inference we need better data structures. Below we start the important special case of filtering, and introduce the Bayes filter. To reason about trajectories we then define hidden Markov models, and highlight their connection with robot localization. We then show how to efficiently perform inference by converting any Bayes net (with evidence) to a factor graph. We show both full posterior inference, MPE, and MAP estimation for HMMs. After that, we generalize to completely general inference in Bayes nets, introducing the elimination algorithm.

1.1 Bayes Filtering

We 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 we are interested in estimating the state of the robot at the current time t, given all the measurements up to and including time t. Assuming that we have a probability distribution $P(S_{t-1})$ over the previous state S_{t-1}, we proceed in two phases:

1. In the **prediction phase** we predict a **predictive distribution** on the current state S_t given the distribution $P(s_{t-1})$ and a given action $A_{t-1} = a_{t-1}$. This is done by marginalizing out the previous state S_{t-1}, by summing over all possible values s_{t-1} for S_{t-1}:

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

2. In the **measurement phase** we upgrade this prior to a posterior via Bayes' rule, given an observation $O_t = o_t$:

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

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.

To be correct, we should really indicate exactly what information we are conditioning on when giving the formulas for the Bayes filter. To that end, we introduce the notation $\mathcal{O}^t = \{o_1, o_2, \dots o_t\}$, i.e., all measurements o up to and including time t. Similarly, we define the action $\mathcal{A}^t = \{a_1, a_2, \dots a_t\}$. The Bayes filter is then stated succinctly as

$$P(S_t|\mathcal{O}^{t-1}, \mathcal{A}^{t-1}) = \sum_{s_{t-1}} P(S_t|s_{t-1}, a_{t-1}) P(s_{t-1}|\mathcal{O}^{t-1}, \mathcal{A}^{t-2}) \tag{1}$$

$$P(S_t|\mathcal{O}^{t}, \mathcal{A}^{t-1}) \propto L(S_t|o_t) P(S_t|\mathcal{O}^{t-1}, \mathcal{A}^{t-1}), \tag{2}$$

and it is seen to be perfectly recursive.

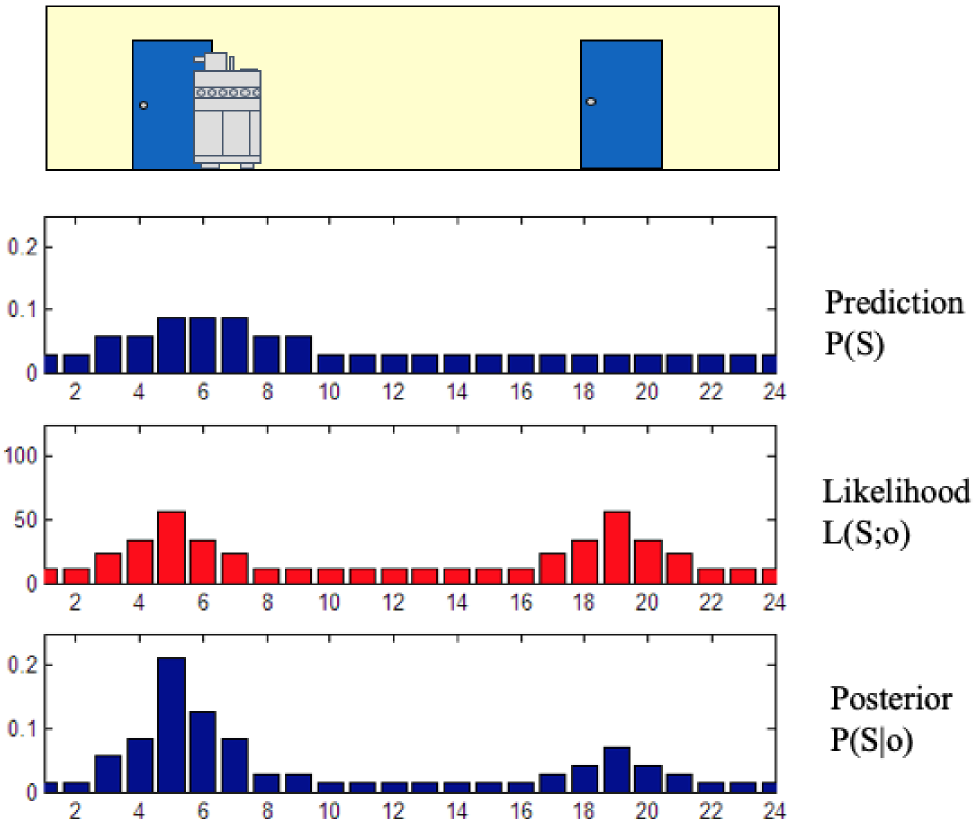

Figure 1: One-dimensional example of the measurement phase in Markov localization, the discrete version of the BAyes filter.

Figure 1 illustrates the measurement phase for a simple 1D example. In this environment there are two doors, and the robot has a door sensor. The predictive distribution $P(S)$ encodes that we believe the robot to be near the left door. The likelihood $L(S; o)$, where o indicates that we *did* perceive a door, models the fact that we are more likely to be near a door given $O = o$, but also allows for the fact that the door sensor could misfire at any location. Note that the likelihood is unnormalized and there is no need for it to sum up to 1. Finally, the posterior $P(S|o)$ is obtained, via Bayes' rule, as the product of the prediction $P(S)$ and the likelihood $L(S;o)$, and is shown at the bottom as a normalized probability distribution. As you can see, the most probable explanation for the robot state is $S=5$, but there is a second mode at $S=17$ due to the bimodal nature of the likelihood. However, that second mode is less probable because of our prior belief over S.

The Bayes filter above is a simple and computationally attractive scheme *if* we are only interested in the posterior distribution over the current state. However, if we 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 x_t is only dependent on the value of the previous state x_{t-1}. We call a sequence of random variables with this property a **Markov chain.** In addition, in an HMM we refer to the states $\mathcal{X}$ as *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.

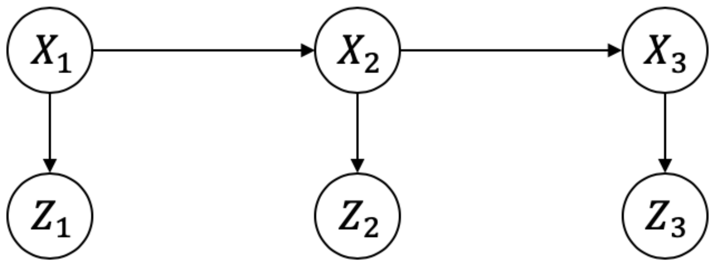

Figure 2: An HMM, unrolled over three time-steps, represented by a Bayes net.

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 measurement models $P(Z_t|x_t)$. 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

$$P(\mathcal{X},\mathcal{Z}) = P(\mathcal{Z}|\mathcal{X})P(\mathcal{X}) \tag{3}$$

where

$$P(\mathcal{X}) = P(x_1,x_2,x_3) = P(x_1)P(x_2|x_1)P(x_3|x_2) \tag{4}$$

is the prior over state *trajectories.*

1.3 Naive Inference in HMMs

In inference, we might want to infer the maximum probable explanation (MPE) for the states $\mathcal{X}$ given values $\mathfrak{z} = \{z_1, z_2, z_3\}$ for $\mathcal{Z}$. As we saw before, one way to perform inference is to apply Bayes' rule to Equation 3, and get an expression for the *posterior* probability distribution over the state trajectory $\mathcal{X}$, given the measurements $\mathcal{Z} = \mathfrak{z}$:

$$\begin{aligned} P(\mathcal{X}|\mathcal{Z}) &\propto P(\mathcal{Z} = \mathfrak{z}|\mathcal{X})P(\mathcal{X}) \\ &= L(\mathcal{X}; \mathcal{Z} = \mathfrak{z})P(\mathcal{X}) \end{aligned} \tag{5}$$

where $P(\mathcal{X})$ is given above in Equation 4, and the **likelihood** $(\mathcal{X}; \mathcal{Z} = \mathfrak{z})$ of $\mathcal{X}$ given $\mathcal{Z} = \mathfrak{z}$ is defined as before, yielding the following function of $\mathcal{X}$:

$$\begin{aligned} L(\mathcal{X}; \mathcal{Z} = \mathfrak{z}) &\stackrel{\Delta}{=} P(\mathcal{Z} = \mathfrak{z}|\mathcal{X}) \\ &= P(z_1|x_1)P(z_2|x_2)P(z_3|x_3) \\ &= L(x_1; Z_1)L(x_2; Z_2)L(x_3; Z_3) \end{aligned}$$

As we saw, a naive implementation for finding the most probable explanation (MPE) for $\mathcal{X}$ would tabulate all possible trajectories $\mathcal{X}$ and calculate the posterior (5) for each one. Unfortunately the number of entries in this giant table is exponential in the number of states. 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 improve on this:

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.

1.4 Factor Graphs

We first introduce the notion of factors. Again referring to the example from Figure 2, let us consider the posterior (5). 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 are likelihood factors as defined before:

$$P(\mathcal{X}|\mathcal{Z}) \propto P(x_1)L(x_1; z_1)P(x_2|x_1)L(x_2; z_2)P(x_3|x_2)L(x_3; z_3) \tag{6}$$

Some of these factors are unary factors, and some are binary factors. In particular, in (6) 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 $P(x_3|x_2)$. Measurements are not counted here, because once we are *given* the measurements $\mathcal{Z}$, they merely function as known parameters in the likelihoods $L(x_t; z_t)$, which are seen as functions of *just* the state x_t.

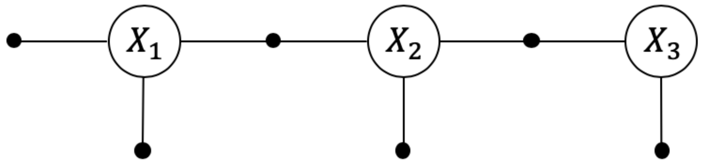

Figure 3: An HMM with observed measurements, unrolled over time, represented as a factor graph.

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. For our example with three hidden states, the corresponding factor graph is shown 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:

$$\phi(\mathcal{X}) = \phi_1(x_1)\phi_2(x_1)\phi_3(x_1, x_2)\phi_4(x_2)\phi_5(x_2, x_3)\phi_6(x_3) \tag{7}$$

where the factors in (7) are defined to correspond one-to-one to Equation 6. For example,

$$\phi_6(x_3) \overset{\Delta}{=} L(x_3; z_3).$$

All measurements are associated with unary factors, whereas 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 random variable nodes adjacent to a factor ϕ_i is written as $\mathcal{X}_i$. With these definitions, a factor graph F defines the factorization of a global function $\phi(\mathcal{X})$ as

$$\phi(\mathcal{X}) = \prod_i \phi_i(\mathcal{X}_i). \tag{8}$$

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. As example, for the factor graph in Figure 3 we have:

$$\begin{aligned}
\mathcal{X}_1 &= \{X_1\} \\
\mathcal{X}_2 &= \{X_1\} \\
\mathcal{X}_3 &= \{X_1, X_2\} \\
\mathcal{X}_4 &= \{X_2\} \\
\mathcal{X}_5 &= \{X_2, X_3\} \\
\mathcal{X}_6 &= \{X_3\}
\end{aligned}$$

1.5 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

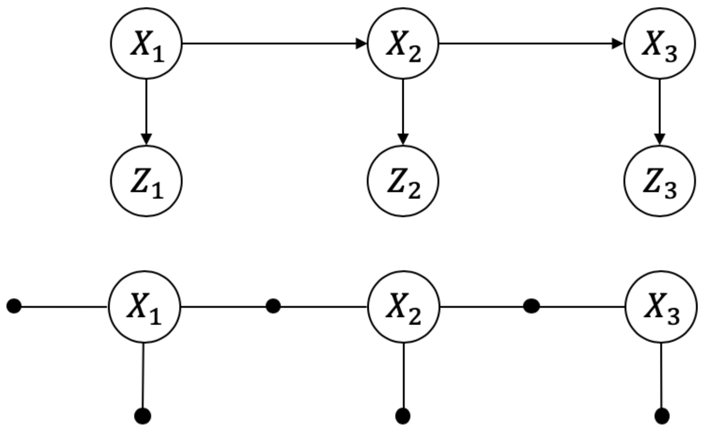

Figure 4: Converting a Bayes net into a factor graph, in the case that the variables $\mathcal{Z}$ are known.

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.

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. The factor graph represents the unnormalized posterior, i.e., $\phi(\mathcal{X}) \propto P(\mathcal{X}|\mathcal{Z})$. 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 the previous section into a factor graph, assuming no known variables.

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

1.6 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

$$\phi(x_1, x_2, x_3) = \prod \phi_i(\mathcal{X}_i) \tag{9}$$

i.e., the **value** of the factor graph. The **max-product algorithm** proceeds one variable at a time, and in an HMM we will proceed from left to right, i.e. we start with state x_1 and proceed until we processed all states.

Eliminating x_1

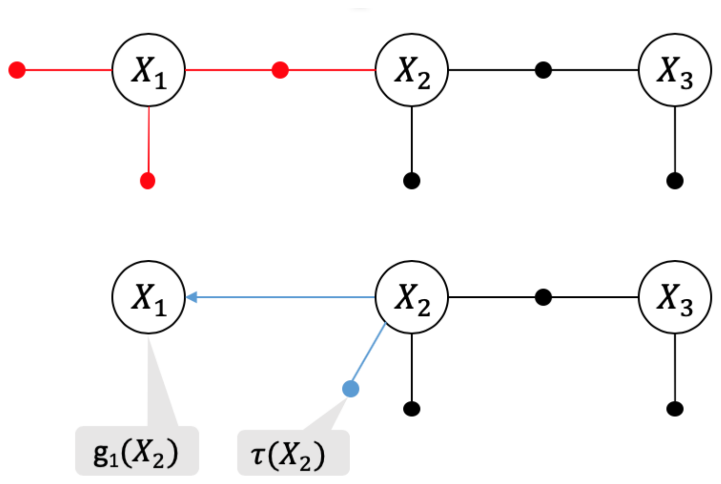

Figure 5: Max-product: eliminating x_1.

We start by considering the first state x_1, and we form a **product factor** $\psi(x_1, x_2)$ that collects *only* the factors connected to x_1:

$$\psi(x_1, x_2) = \phi_1(x_1)\phi_2(x_1)\phi_3(x_1, x_2).$$

When we use a factor in a product, we *remove* it from the original factor graph. Note that because one of those factors, the state transition model $\phi_3(x_1, x_2) \overset{\Delta}{=} P(x_2|x_1)$, is also connected to the second state x_2, the product factor is a function of *both* x_1 and x_2, i.e., it is a binary factor.

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 g_1 for the best possible value of x_1:

$$g_1(x_2) = \arg\max_{x_1} \psi(x_1, x_2).$$

The size of this lookup table is equal to the number of possible outcomes for x_2: in our grid-world example this is 100, as we are using a 10×10 grid. We can of course store this lookup table as a grid, as well. It might help your understanding to think about what this table will look like.

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**

$$\Phi_{2:3} = \tau(x_2)\phi_4(x_2)\phi_5(x_2, x_3)\phi_6(x_3). \tag{10}$$

Let us reflect on what happened above, because it is significant: we eliminated x_1 from consideration, and obtained a reduced problem that only depends on the remaining states x_2 and x_3. You can intuitively see that this algorithm will terminate after n steps, and in fact you could prove it by induction. In addition, the lookup table g_1 gives us a way that, once we know what the optimal value for x_2 is, we can just read off the optimal value for x_1. This is what we will do, in *reverse* elimination order, after the algorithm terminates.

Eliminating x_2

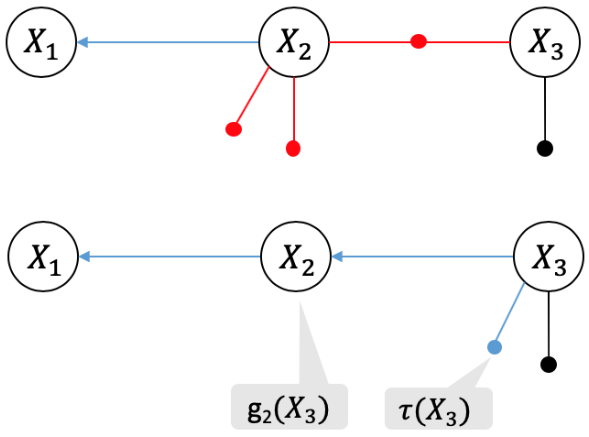

Figure 6: Max-product: eliminating x_2.

We now perform exactly the same steps for the state x_2. In this case, the product factor $\psi(x_2, x_3)$ has only 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. Note that since we started from the reduced graph (10), the product factor is guaranteed to not depend on the first state x_1: that was eliminated! In fact, we can now in turn eliminate x_2 from the problem, by looking at all possible values x_3 of x_3, and creating a lookup table g_2 for the best possible value of x_2, given x_3,

$$g_2(x_3) = \arg\max_{x_2} \psi(x_2, x_3),$$

and as above we also record the value of the product factor for that maximum in a new factor $\tau(x_3)$:

$$\tau(x_3) = \max_{x_2} \psi(x_2, x_3).$$

We then put this new factor $\tau(x_3)$ back into the graph, which is now reduced even more:

$$\Phi_{3:3} = \tau(x_3)\phi_6(x_3)$$

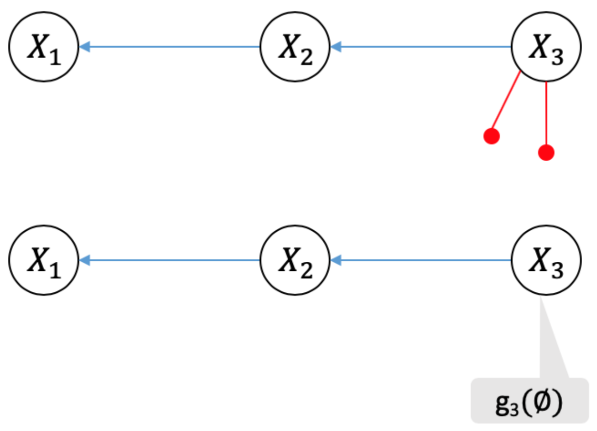

Figure 7: Max-product: eliminating x_3.

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 9.

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

```
1: function MaxProductHMM($\Phi_{1:n}$)                         ▷ given an HMM with $n$ states
2:     for $j = 1...n$ do                                        ▷ for all states
3:         $g_j(x_{j+1}), \Phi_{j+1:n} \leftarrow$ CreateLookupTable($\Phi_{j:n}, x_j$)     ▷ eliminate $x_j$
4:     return $\{g_1(x_1)g(x_2)\dots g_n(\emptyset)\}$            ▷ return DAG of lookup tables
```

Algorithm 2 Create lookup table g_j by eliminating state x_j from a factor graph $\Phi_{j:n}$.

```
1: function CreateLookupTable($\Phi_{j:n}, x_j$)                ▷ given reduced graph $\Phi_{j:n}$
2:     Remove all factors $\phi_i(\mathcal{X}_i)$ that are contain $x_j$
3:     $\psi(x_j, x_{j+1}) \leftarrow \prod_i \phi_i(\mathcal{X}_i)$                ▷ create the product factor $\psi$
4:     $g_j(x_{j+1}), \tau(x_{j+1}) \leftarrow \psi(x_j, x_{j+1})$                ▷ perform argmax, max
5:     Add the new factor $\tau(x_{j+1})$ back into the graph
6:     return $g_j(x_{j+1}), \Phi_{j+1:n}$                ▷ 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 $\Phi_{j:n} \overset{\Delta}{=} \phi(x_j, \ldots, x_n)$ 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 $g_j(x_{j+1})$, 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.

1.7 The Sum-Product Algorithm for HMMs

The sum-product algorithm for HMMs is a slight tweak on the max-product algorithm that instead produces a Bayes net that calculates the posterior probability $P(\mathcal{X}|\mathcal{Z})$. Whereas the max-product produces a DAG of lookup tables, the sum-product produces a DAG of conditionals, i.e., a Bayes net. This is particularly interesting if one is content with a maximum probable explanation or MPE, but instead wants the **full Bayesian probability distribution** of which assignments to the states are more probable than others. The fact that we recover this distribution in the form of a Bayes net again is satisfying, because as we saw that is an economical representation of a probability distribution.

One might wonder about the wisdom of all this: we started with a Bayes net, converted to a factor graph, and now end up with a Bayes net again? Indeed, but there are two important differences: the first Bayes net represents the joint distribution $P(\mathcal{X}, \mathcal{Z})$ and is very useful for modeling. However, the second Bayes represents the posterior $P(\mathcal{X}|\mathcal{Z})$, and only has nodes for the random variables in $\mathcal{X}$, hence it is much smaller. Finally, in many practical cases we do not even bother with the modeling step, but construct the factor graph directly from the measurements.

Algorithm 3 The Sum-Product Algorithm for HMMs

```
1: function SumProductHMM($\Phi_{1:n}$)                ▷ given an HMM with $n$ states
2:     for $j = 1...n$ do                ▷ for all states
3:         $p(x_j|x_{j+1}), \Phi_{j+1:n} \leftarrow$ ApplyChainRule($\Phi_{j:n}, x_j$)                ▷ eliminate $x_j$
4:     return $p(x_1|x_2)p(x_2|x_3)\ldots p(x_n)$                ▷ return Bayes net
```

The only tweak necessary is to replace the maximization and arg max in the elimination step with the chain rule. Indeed, we factor each product factor $\psi(x_j, x_{j+1})$ into a conditional $P(x_j|x_{j+1})$ and an (unnormalized) marginal $\tau(x_{j+1})$:

$$P(x_j|x_{j+1})\tau(x_{j+1}) \leftarrow \psi(x_j, x_{j+1}) \tag{11}$$

The algorithm is called the **sum-product algorithm** because the marginal is obtained by summing

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

```
1: function ApplyChainRule($\Phi_{j:n}, x_j$)                ▷ given reduced graph $\Phi_{j:n}$
2:     Remove all factors $\phi_i(\mathcal{X}_i)$ that contain $x_j$
3:     $\psi(x_j, x_{j+1}) \leftarrow \prod_i \phi_i(\mathcal{X}_i)$                ▷ create the product factor $\psi$
4:     $p(x_j|x_{j+1})\tau(x_{j+1}) \leftarrow \psi(x_j, x_{j+1})$                ▷ factorize the product $\psi$
5:     Add the new factor $\tau(x_{j+1})$ back into the graph
6:     return $p(x_j|x_{j+1}), \Phi_{j+1:n}$                ▷ conditional and reduced graph
```

over all values of the state x_j:

$$\tau(x_{j+1}) = \sum_{x_j} \psi(x_j, x_{j+1}) \tag{12}$$

We do not bother normalizing this into a proper distribution, as these marginals are just intermediate steps in the algorithm. However, when computing the conditional, we do normalize, and is it so happens the normalization constant is simply equal to $1/\tau(x_{j+1})$:

$$P(x_j|x_{j+1}) = \frac{\psi(x_j, x_{j+1})}{\tau(x_{j+1})} \tag{13}$$

In summary, the chain rule is implemented by 12 and 13. The entire algorithm is is listed as Algorithm 3.

Note that after we recover the Bayes net the algorithm terminates: there is no back-substitution step. However, one might consider ancestral sampling as a type of back-substitution: the reverse elimination order is always a topological sort of the resulting Bayes net! Hence, after the sum-product algorithm, we can sample as many realizations from the posterior as we want: rather than just one MPE, we now have thousands of plausible explanations, and ancestral sampling will yield them in exactly the correct frequencies.

Sidebar

When we can produce samples $\mathcal{X}^{(s)}$ from a posterior $P(\mathcal{X}|\mathcal{Z})$, we can calculate empirical means of any real-valued function $f(\mathcal{X})$ as follows:

$$E_{P(\mathcal{X}|\mathcal{Z})}[f(x)] \approx \sum f(\mathcal{X}^{(s)})$$

For example, we can calculate the posterior mean of how far the robot traveled, either in Euclidean or Manhattan distance. These estimators will have less variability than just calculating the distance for the MPE, as they average over the entire probability distribution.

1.8 The Variable Elimination Algorithm

There exists a general algorithm that, given *any* factor graph, can compute the corresponding posterior distribution $p(\mathcal{X}|\mathcal{Z})$ on the unknown variables $\mathcal{X}$. Above we saw that a factor graph represents the unnormalized posterior $\phi(\mathcal{X}) \propto P(\mathcal{X}|\mathcal{Z})$ as a product of factors, typically generated directly from the measurements. The variable elimination algorithm is a general recipe for converting any factor graph back to a Bayes net, but now *only* on the unknown variables $\mathcal{X}$.

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

$$\phi(\mathcal{X}) = \phi(x_1, \ldots, x_n) \tag{14}$$

into a factored Bayes net probability density of the form

$$p(\mathcal{X}) = p(x_1|\mathcal{S}_1)p(x_2|\mathcal{S}_2)\dots p(x_n) = \prod_j p(x_j|\mathcal{S}_j), \tag{15}$$

where the **separator** $\mathcal{S}(x_j)$ 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, although this depends on the chosen variable ordering $x_1, \dots, x_n$.

Algorithm 5 The Variable Elimination Algorithm

```
1: function ELIMINATE($\Phi_{1:n}$)                               ▷ given a factor graph on $n$ variables
2:     for $j = 1...n$ do                                         ▷ for all variables
3:         $p(x_j|\mathcal{S}_j), \Phi_{j+1:n} \leftarrow$ EliminateOne($\Phi_{j:n}, x_j$)   ▷ eliminate $x_j$
4:     return $p(x_1|\mathcal{S}_1)p(x_2|\mathcal{S}_2)\dots p(x_n)$            ▷ return Bayes net
```

Algorithm 6 Eliminate variable x_j from a factor graph $\Phi_{j:n}$.

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

The variable elimination algorithm is listed as Algorithm 5, where we again used the shorthand notation $\Phi_{j:n} \overset{\Delta}{=} \phi(x_j, \dots, x_n)$ 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 $p(x_j|\mathcal{S}_j)$, 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.

Above we gave the sum-product version of the variable elimination algorithm. The corresponding max-product version produces a DAG of lookup tables instead, supporting the computation of the MPE. In both cases, the complexity is similar but depends on the chosen variable ordering $x_1, \dots, x_n$, as we discuss next.

1.9 Complexity

The elimination algorithm has exponential complexity in the size of the largest separator. The chosen variable ordering $x_1, \dots, x_n$ 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, but we will not discuss that here.

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 (there is at most one path from any node to any other node). Algorithms 1 and 3 are the max-product and sum-product variants that we obtain when eliminating from left to right. However, it is possible to choose a different ordering, even for HMMs. For example, if we eliminate from right to left, we get an equally efficient algorithm, although all the intermediate product and marginal factors will be different.

Exercises

1. Perform symbolic elimination, i.e., just the graph part without computation, on some factor graphs of interest.
2. Come up with an ordering for an HMM which is neither left-right or right-left which nevertheless results in a singly-connected Bayes net.
3. Come up with an ordering to eliminate an HMM which does not lead to a singly-connected Bayes net.

1.10 MAP Estimation

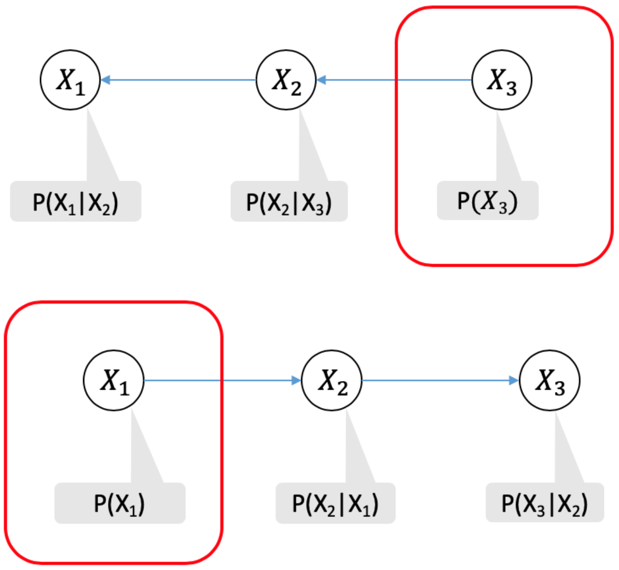

Figure 8: MAP estimation for x_3 and x_1, respectively: we simply change the elimination order to make sure the variable(s) of interest are eliminated last.

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 some factor graphs of interest.
2. Construct a small example where MAP estimation is more expensive than MPE, even when using optimal orderings for both.
3. Think about the complexity of MAP estimation in an HMM. When is it not more expensive than MPE?

Summary

We briefly summarize what we learned in this section:

1. Bayes filtering is a recursive state estimation scheme.
2. Hidden Markov models can be used to reason about a sequence of states observed indirectly via sensors.
3. Naive inference in HMMs can be quite expensive.
4. Factor graphs are a new graphical language that make measurements implicit.
5. Any Bayes net (with or without evidence) can be converted to a factor graph.
6. The max-product algorithm for HMMs is an efficient computation for the MPE.
7. The sum-product algorithm returns the full Bayesian posterior as a Bayes net.
8. The variable elimination algorithm is a generalization that works for *any* factor graph.
9. The complexity of variable elimination depends on the elimination order.
10. MAP estimation is always as least as expensive, as it constrains the elimination order.