

CS 3630!



Lecture 8: A Vacuum Cleaning Robot: Sensing, and Perception

Lecture 8 recap

For our vacuum robot, we looked at

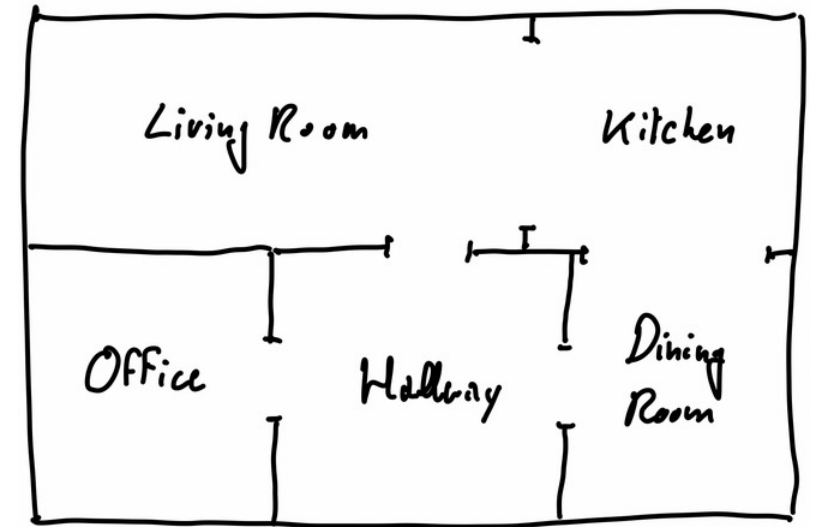
- Simple sensing
- (Dynamic) Bayes Nets
- HMMs as a special case
- Most probable explanation (MPE)
- Factor graphs

Vacuuming robot sensor

- A single sensor that detects light levels, and returns a measurement z :

$X1$	dark	medium	light
Living Room	0.1	0.1	0.8
Kitchen	0.1	0.1	0.8
Office	0.2	0.7	0.1
Hallway	0.8	0.1	0.1
Dining Room	0.1	0.8	0.1

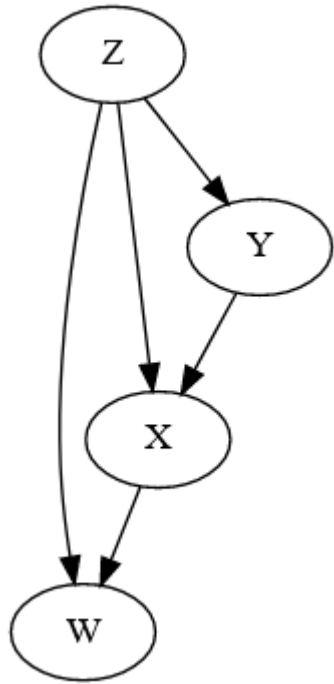
- Bright, $z = 2$
- Medium, $z = 1$
- Dark, $z = 0$



- Sun is to the south, so plenty of light for living room and kitchen.
- Office and Dining room are poorly lit via windows.
- Hallway has no windows and is always dark.

- For Hallway, $(z = 0 | H) = 0.8$, MLE will do the job!*
- For $z = 1, z = 2$, there's really no way to uniquely identify state from one measurement.*

The Magic of Bayes Nets



For a Bayes net with variables $X_1 \dots X_n$, the joint distribution is given by:

$$P(X_1 \dots X_n) = \prod_i P(X_i | \mathcal{P}(X_i))$$

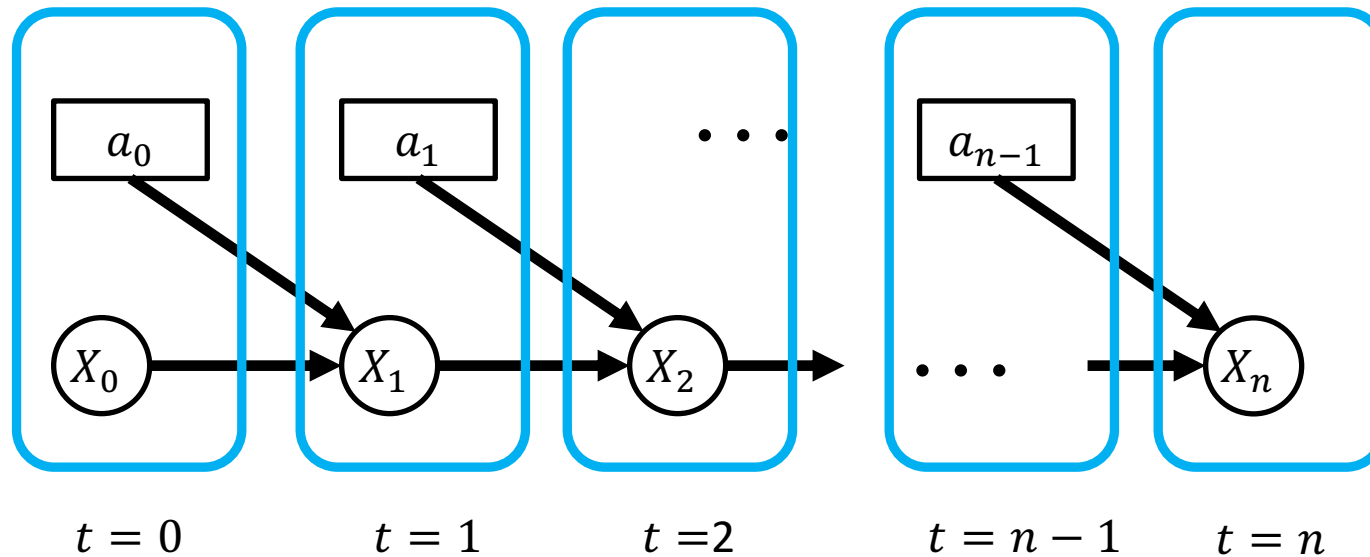
where $\mathcal{P}(X_i)$ denotes the set of parents of node X_i

For this specific network, the joint distribution is given by

$$P(W, X, Y, Z) = P(W | X, Z) P(X | Y, Z) P(Y | Z) P(Z)$$

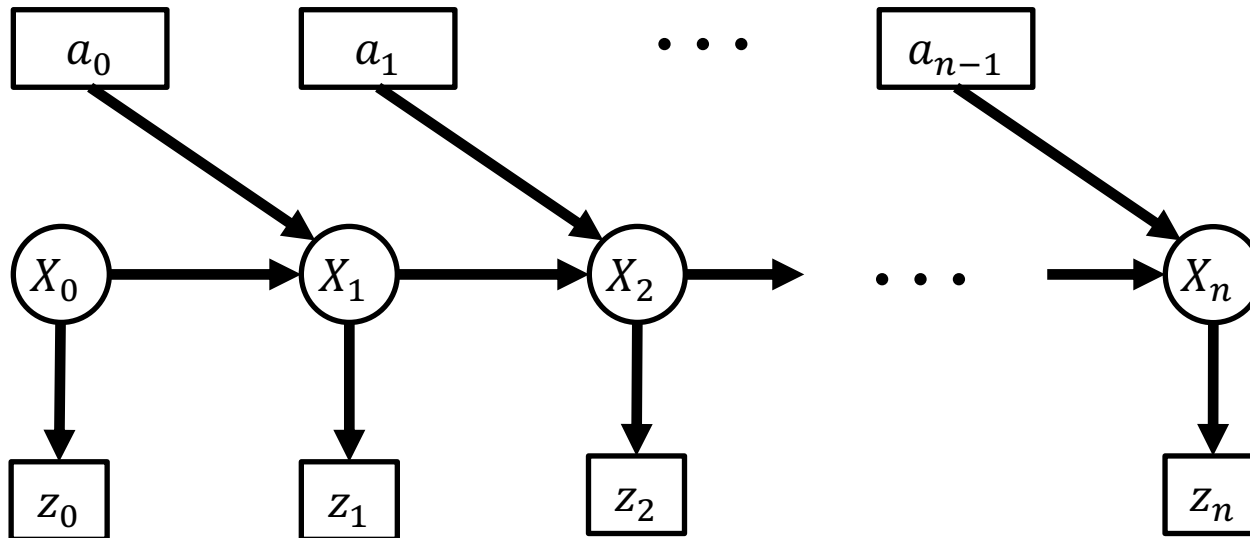
Dynamic Bayes Nets

- Bayes nets can be used to represent systems that evolve over time.
- Our vacuum cleaning robot is an example of such a system, at any time t , we have x_t and a_t and together, these determine (probabilistically) what happens for x_{t+1} .
- A dynamic Bayes net has a simple structure that repeats at each time step:



Hidden Markov Models (HMMs)

- Notice that in the system shown below,
 - we know $\mathbf{Z}_t = \mathbf{z}_t$ for all t
 - We know \mathbf{a}_t for all t
- We do not know any of $\mathbf{X}_0 \dots \mathbf{X}_n$, but we do know that the states form a Markov chain.
- We say that the states, $\mathbf{X}_0 \dots \mathbf{X}_n$, are hidden.



HMMs are a good model for speech recognition systems:

- Spoken words behave like a Markov chain (if you know the current word, you know a lot about what will be the next word).
- Measurements are audio signals.

Note: If we increase the relevant history, e.g., so that state X_t depends on $X_{t-1}, X_{t-2} \dots X_{t-n}$, we have an n th order Markov chain. Larger n gives better prediction.

Most Probable Explanation

We are given $Z_t = z_t$, and a_t for all t .

For every possible value of x_0, \dots, x_n , compute

$$P(\mathbf{X}, \mathbf{Z}, \mathbf{A}) = P(\mathbf{Z}_0 = \mathbf{z}_0 | \mathbf{X}_0 = \mathbf{x}_0) P(\mathbf{X}_0 = \mathbf{x}_0) \prod_i P(\mathbf{Z}_i = \mathbf{z}_i | \mathbf{X}_i = \mathbf{x}_i) P(\mathbf{X}_i = \mathbf{x}_i | \mathbf{X}_{i-1} = \mathbf{x}_{i-1}, \mathbf{a}_i)$$

Our estimate is given by

$$\mathbf{X}^* = \arg \max_{\mathbf{X}} P(\mathbf{X}, \mathbf{Z}, \mathbf{A})$$

Not the most efficient algorithm, but in principle, this gets the job done.

Inference in Factor Graphs

We take an HMM, modeled as a Bayes net.

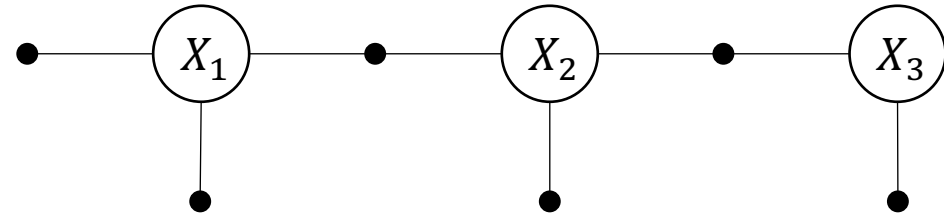
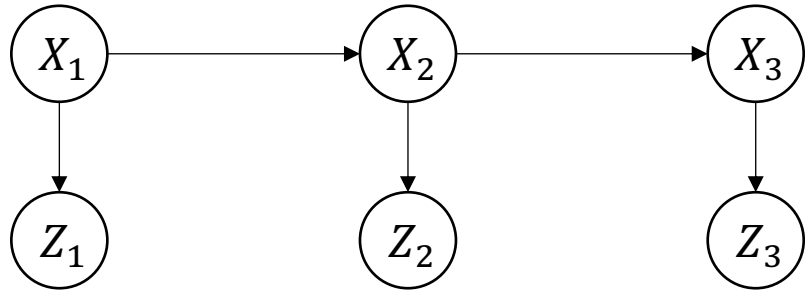
We would like to do MPE as well as full Bayesian inference.

We already know a naïve algorithm for this: brute-force enumeration of the posterior.

When we remove all variables that are given to us we obtain a **factor graph**.

➤ *The factor graph is a great tool to exploit the sparse model structure to obtain computationally efficient inference algorithms.*

Factor Graphs



- Measurements are given – get rid of them!

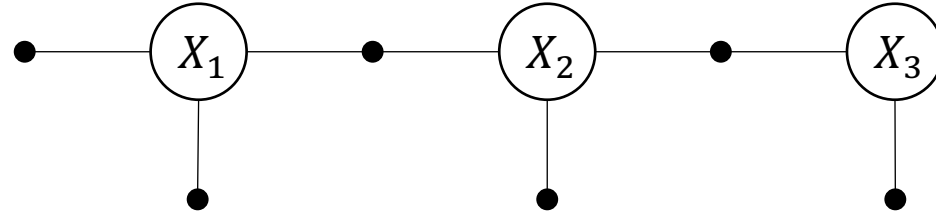
$$P(\mathcal{X}|\mathcal{Z}) \propto P(X_1)L(X_1; z_1)P(X_2|X_1)L(X_2; z_2)P(X_3|X_2)L(X_3; z_3)$$

- This becomes:

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

Each factor defines a function ϕ which is a function only of its (non-factor node) neighbors.

General definition of Factor graphs



- Bipartite graph of variables and factors

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

- Each \mathcal{X}_i is the subset of variables connected to factor ϕ_i

Subsets here are:

$$\mathcal{X}_1 = \{X_1\}$$

$$\mathcal{X}_2 = \{X_1\}$$

$$\mathcal{X}_3 = \{X_1, X_2\}$$

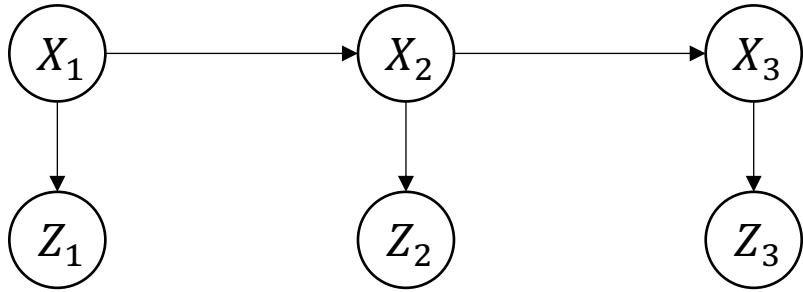
$$\mathcal{X}_4 = \{X_2\}$$

$$\mathcal{X}_5 = \{X_2, X_3\}$$

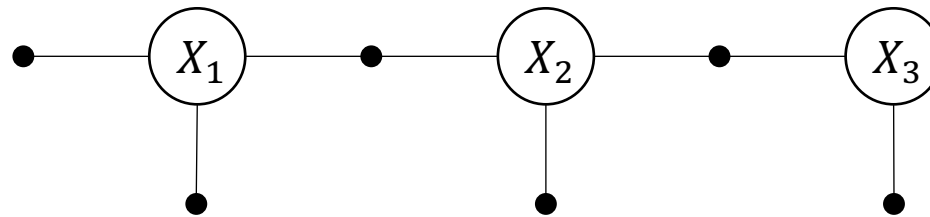
$$\mathcal{X}_6 = \{X_3\}$$

Example

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



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

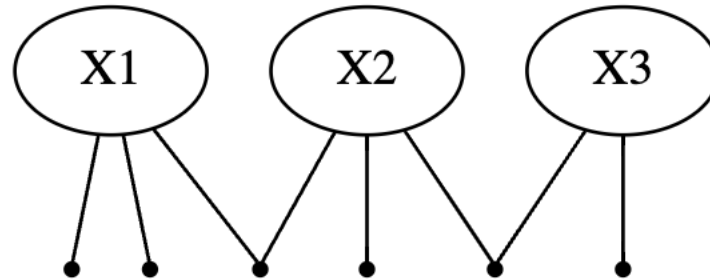


$$\phi_6(X_3) \doteq L(X_3; z_3).$$

Factor Graphs in GTSAM

- See https://www.roboticsbook.org/S34_vacuum_perception.html#factor-graphs-in-gtsam

```
show(graph)
```



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

Computing with Factor Graphs

- Can evaluate for any X : $\phi(X_1, X_2, X_3) = \prod \phi_i(x_i)$
- Hence, naïve MPE is also simple, e.g.:

```
mpe_value = 0
mpe_trajectory = None
for x1 in vacuum.rooms:
    for x2 in vacuum.rooms:
        for x3 in vacuum.rooms:
            trajectory = VARIABLES.assignment({X[1]: x1, X[2]: x2, X[3]: x3})
            value = graph(trajectory)
            if value > mpe_value:
                mpe_value = value
                mpe_trajectory = trajectory
print(f"found MPE solution with value {mpe_value:.4f}:")
pretty(mpe_trajectory)
```

found MPE solution with value 0.3277:

Variable	value
X1	Hallway
X2	Dining Room
X3	Kitchen

Max-Product for MPE

- We can “eliminate” one variable at a time, e.g., X_1 :

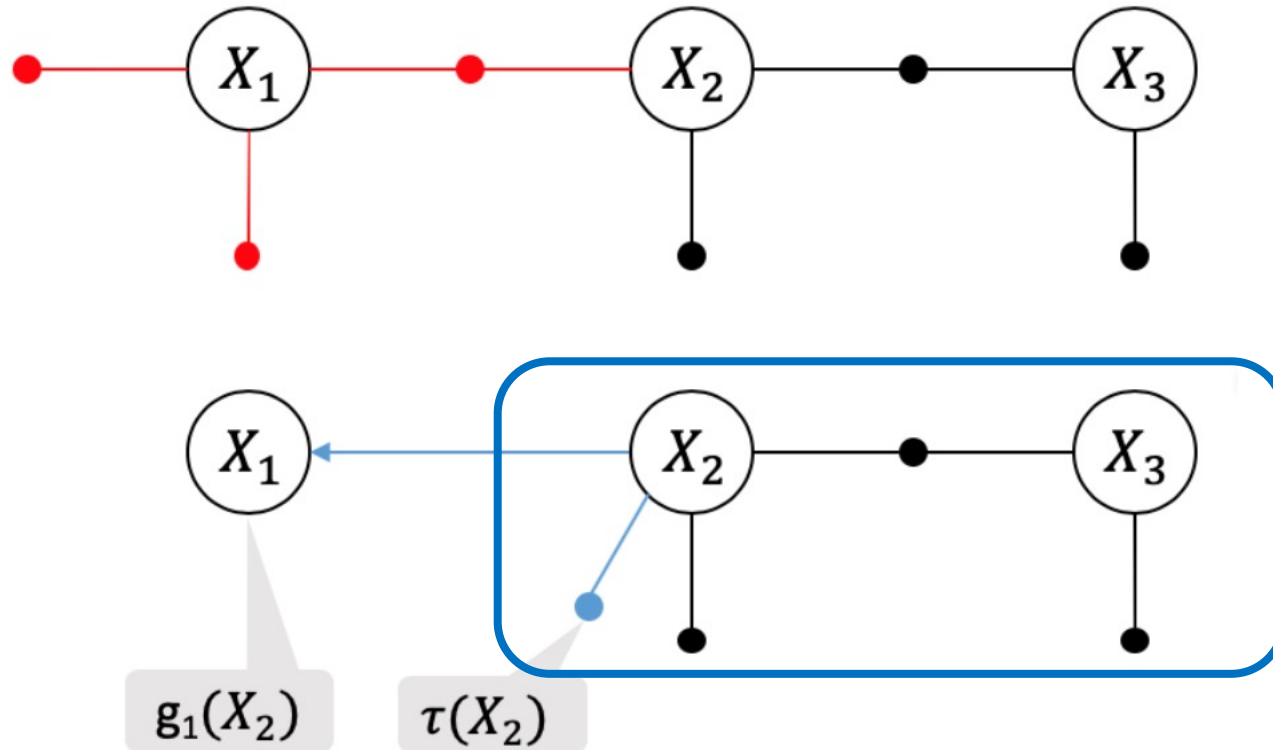
$$\begin{aligned}\max_{\mathcal{X}} \prod \phi_i(\mathcal{X}_i) &= \max_{X_1, X_2, X_3} \phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2) && \phi_4(X_2)\phi_5(X_2, X_3)\phi_6(X_3) \\ &= \max_{X_2, X_3} \left\{ \max_{X_1} \phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2) \right\} && \phi_4(X_2)\phi_5(X_2, X_3)\phi_6(X_3) \\ &= \max_{X_2, X_3} \tau(X_2) && \phi_4(X_2)\phi_5(X_2, X_3)\phi_6(X_3)\end{aligned}$$

- The new quantity $\tau(X_2)$ is no longer a function of X_1 , as for any given value of X_2 we “memoize” the maximum value of the product of three factors that involve X_1 .
- We **eliminated** X_1 !!!!

A graphical elimination game

- We can represent the elimination of X_1 graphically:

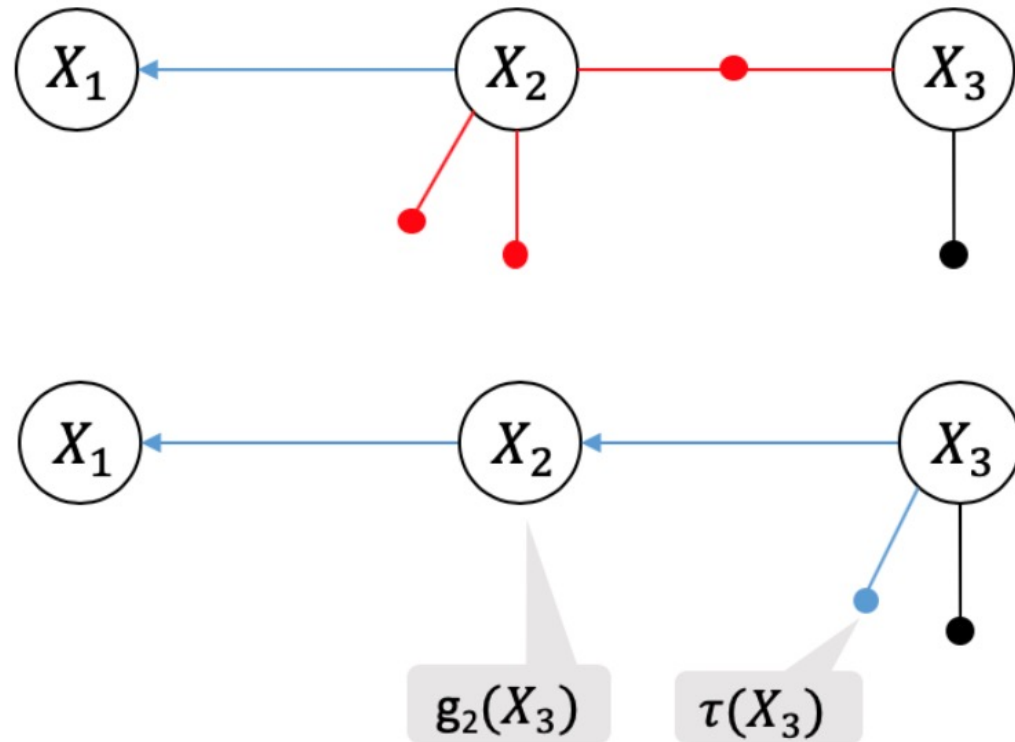
$$\phi(X_1, X_2) = \phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2).$$



$$g_1(X_2) = \arg \max_{x_1} \phi(x_1, X_2) \quad \tau(X_2) = \max_{x_1} \phi(x_1, X_2)$$

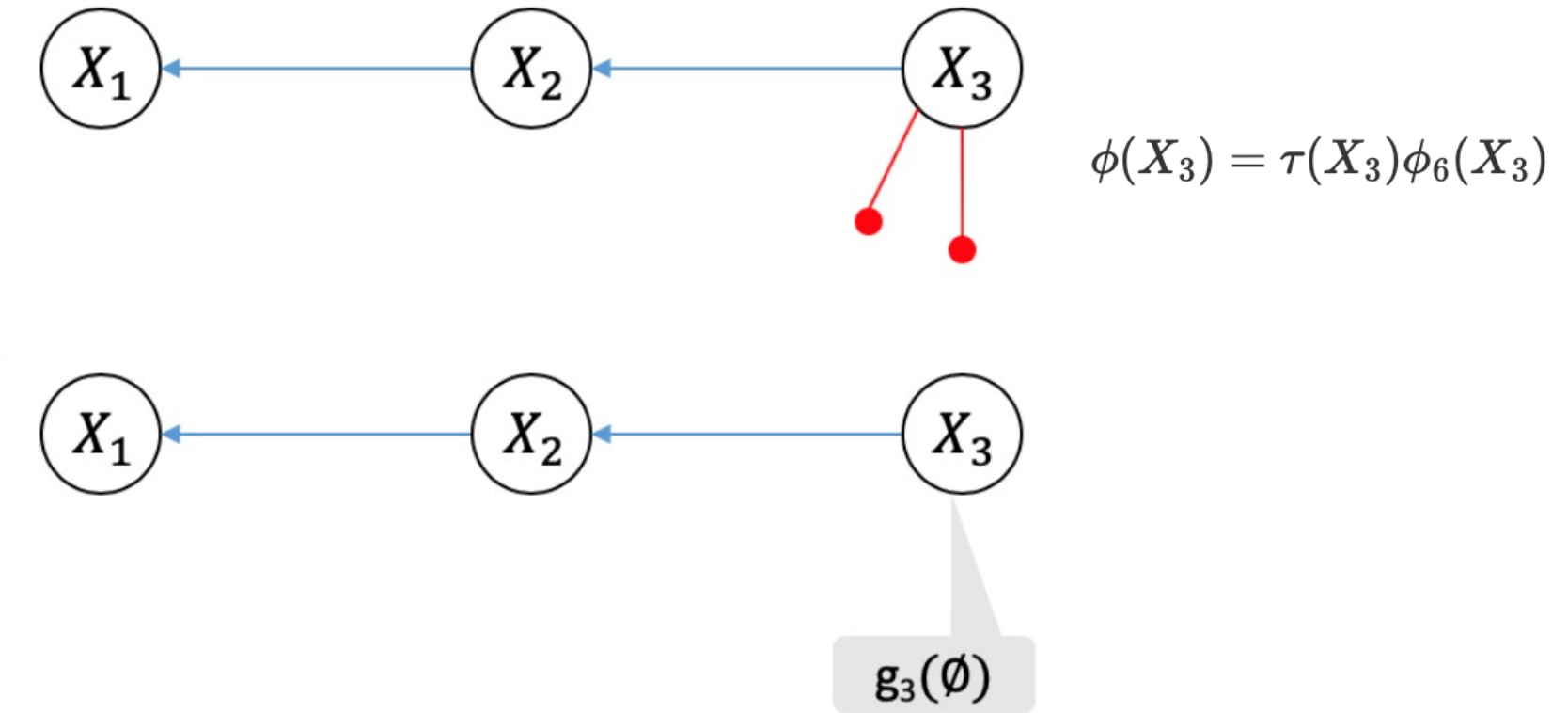
A graphical elimination game

- We then recurse on the new, smaller factor graph by eliminating X_2 :



A graphical elimination game

- And finally, we eliminate X_3 :



Max-Product Revisited

- We decide on an elimination ordering, then for every elimination step we calculate a product and a maximization:

MaxProductHMM ($\Phi_{1:n}$):

- for $j = 1 \dots n$:
 - $g_j(X_{j+1}), \Phi_{j+1:n} \leftarrow \text{CreateLookupTable}(\Phi_{j:n}, X_j)$
- return $g_1(X_1)g(X_2) \dots g_n(\emptyset)$

CreateLookupTable ($\Phi_{j:n}, X_j$):

- Remove all factors $\phi_i(\mathcal{X}_i)$ that contain X_j
- Form the product factor $\phi(X_j, X_{j+1}) \leftarrow \prod_i \phi_i(\mathcal{X}_i)$
- Eliminate X_j : $g_j(X_{j+1}), \tau(X_{j+1}) \leftarrow \phi(X_j, X_{j+1})$
- Add new factor $\tau(X_{j+1})$ back into the graph $\Phi_{j+1:n}$
- return $g_j(X_{j+1}), \Phi_{j+1:n}$

Back-substitution

- At the end, we recover the MPE value, but what about the MPE assignment of X ?
- Lookup tables $g()$ to the rescue

MaxProductHMM ($\Phi_{1:n}$):

- for $j = 1 \dots n$:
 - $g_j(X_{j+1}), \Phi_{j+1:n} \leftarrow \text{CreateLookupTable}(\Phi_{j:n}, X_j)$
- return $g_1(X_1)g(X_2) \dots g_n(\emptyset)$

Max-product for MP in GTSAM

- While I expect you to understand the max-product algorithm, the implementation of is conveniently done by us in GTSAM:

```
mpe = graph.optimize()  
pretty(mpe)
```

Variable	value
X1	Hallway
X2	Dining Room
X3	Kitchen

Sum-product for HMMs

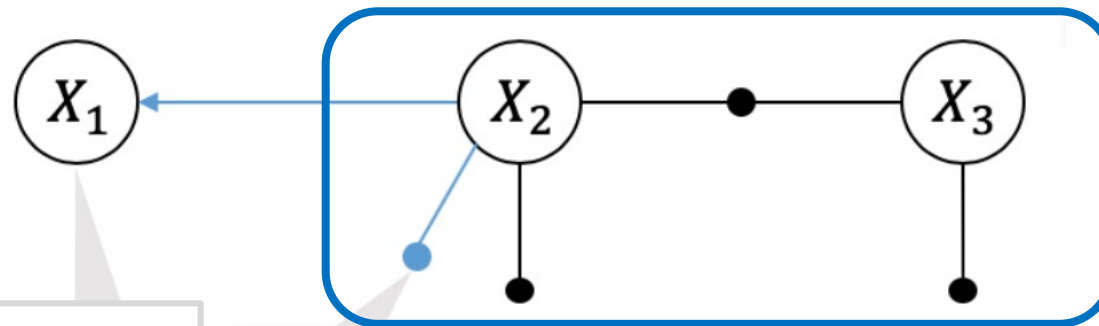
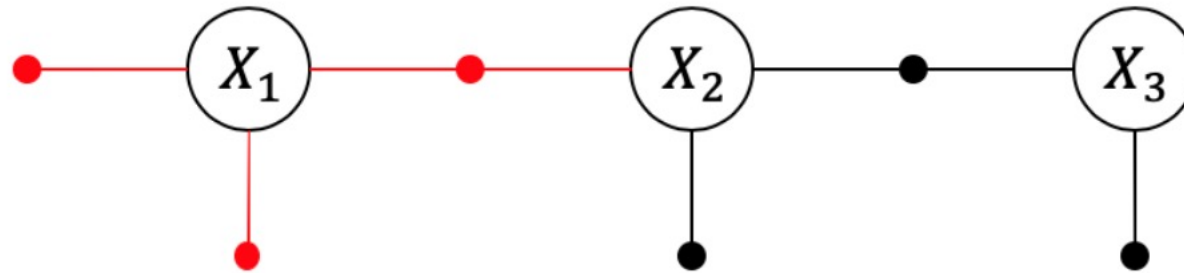
- Similar dynamic programming idea:

$$\begin{aligned} P(\mathcal{X}|\mathcal{Z}) &\propto \prod \phi_i(\mathcal{X}_i) \\ &\propto \phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2)\phi_4(X_2)\phi_5(X_2, X_3)\phi_6(X_3) \\ &\propto \{\phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2)\} \phi_4(X_2)\phi_5(X_2, X_3)\phi_6(X_3) \\ &\propto \{P(X_1|X_2, \mathcal{Z})\tau(X_2)\} \phi_4(X_2)\phi_5(X_2, X_3)\phi_6(X_3) \\ &= P(X_1|X_2, \mathcal{Z}) P(X_2, X_3|\mathcal{Z}) \end{aligned}$$

Sum-product looks the same graphically:

- But eliminating now involves a sum and a conditional

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



$$P(X_1|X_2) = \frac{\phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2)}{\tau(X_2)}.$$

$\tau(X_2)$

$$\tau(X_2) \doteq \sum_{X_1} \phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2)$$

Comparing max and sum-product:

- Almost identical, replace max with sum:

MaxProductHMM ($\Phi_{1:n}$):

- for $j = 1 \dots n$:
 - $g_j(X_{j+1}), \Phi_{j+1:n} \leftarrow \text{CreateLookupTable}(\Phi_{j:n}, X_j)$
- return $g_1(X_1)g(X_2) \dots g_n(\emptyset)$

CreateLookupTable ($\Phi_{j:n}, X_j$):

- Remove all factors $\phi_i(\mathcal{X}_i)$ that contain X_j
- Form the product factor $\phi(X_j, X_{j+1}) \leftarrow \prod_i \phi_i(\mathcal{X}_i)$
- Eliminate X_j : $g_j(X_{j+1}), \tau(X_{j+1}) \leftarrow \phi(X_j, X_{j+1})$
- Add new factor $\tau(X_{j+1})$ back into the graph $\Phi_{j+1:n}$
- return $g_j(X_{j+1}), \Phi_{j+1:n}$

SumProductHMM ($\Phi_{1:n}$):

- for $\{j = 1 \dots n\}$:
 - Eliminate X_j : $P(X_j|X_{j+1}), \Phi_{j+1:n} \leftarrow \text{ApplyChainRule}(\Phi_{j:n}, X_j)$
- return Bayes net $P(X_1|X_2)P(X_2|X_3) \dots P(X_n)$

ApplyChainRule ($\Phi_{j:n}, X_j$):

- Remove all factors $\phi_i(\mathcal{X}_i)$ that contain X_j
- Create product factor $\phi(X_j, X_{j+1}) \leftarrow \prod_i \phi_i(\mathcal{X}_i)$
- Factorize the product $P(X_j|X_{j+1})\tau(X_{j+1}) \leftarrow \phi(X_j, X_{j+1})$
- Add the new factor $\tau(X_{j+1})$ back into the graph
- return the conditional $P(X_j|X_{j+1})$ and reduced graph $\Phi_{j+1:n}$

- Spot the differences ☺

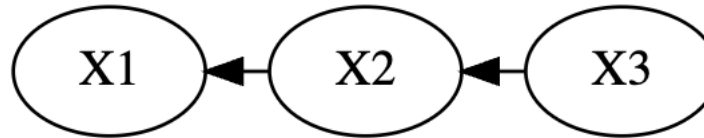
Max and sum-product in GTSAM

- ***optimize*** yields an assignment, ***sumProduct*** yields a distribution!

```
mpe = graph.optimize()  
pretty(mpe)
```

Variable	value
X1	Hallway
X2	Dining Room
X3	Kitchen

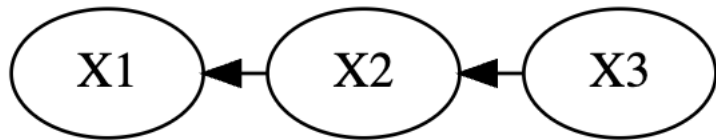
```
posterior = graph.sumProduct()  
show(posterior, hints={"X": 1})
```



Power of posteriors

- *We can sample from the posterior:*

```
posterior = graph.sumProduct()  
show(posterior, hints={"X": 1})
```



```
sample = posterior.sample()  
pretty(sample)
```

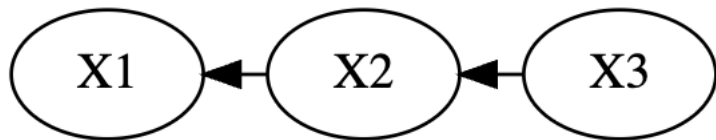
✓ 0.1s

Variable	value
X1	Hallway
X2	Dining Room
X3	Kitchen

Power of posteriors

- We can sample from the posterior, e.g., sample 1000 alternate histories:*

```
posterior = graph.sumProduct()  
show(posterior, hints={"X": 1})
```



```
sample = posterior.sample()  
pretty(sample)
```

✓ 0.1s

Variable	value
X1	Hallway
X2	Dining Room
X3	Kitchen

```
counts = np.zeros((3, 5))  
num_samples = 1000  
for i in range(num_samples):  
    sample = posterior.sample()  
    for k in range(1, 3+1):  
        (variable) room_index: Any  
        room_index = sample[key]  
        counts[k-1][room_index] += 1 # base 0!
```

✓ 0.9s

```
pd.DataFrame(data=100*counts/num_samples,  
             index=range(1, N+1), columns=vacuum.rooms)
```

✓ 0.7s

	Living Room	Kitchen	Office	Hallway	Dining Room
1	2.0	2.3	2.6	82.5	10.6
2	0.5	3.8	0.6	4.5	90.6
3	5.0	91.9	0.6	0.0	2.5

Next Lecture

- Use the power of this full posterior inference (sum product!) together with a reward/cost framework to **act optimally**.