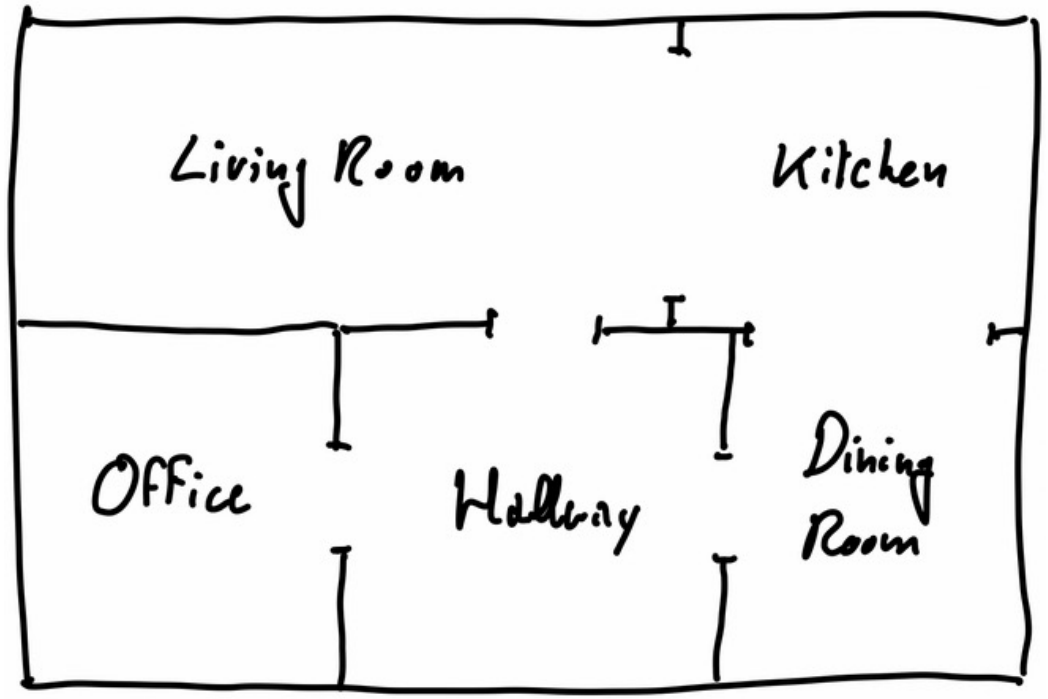
Lecture 10
Inference in factor graphs

Sparse factor graphs make Bayesian inference practical

Frank Dellaert

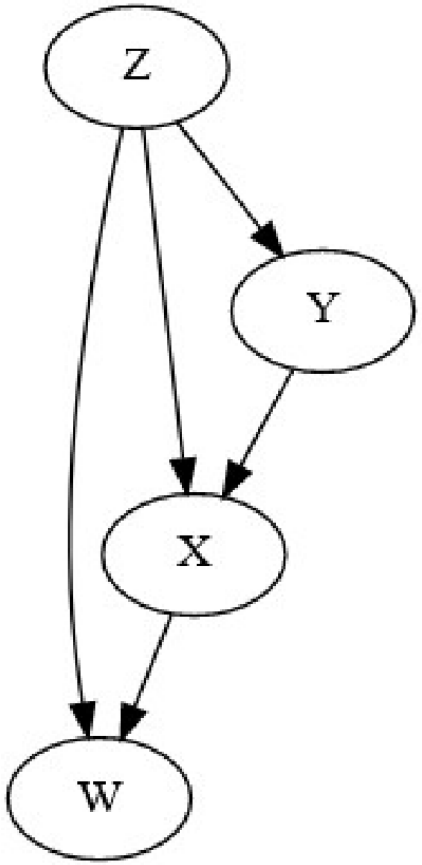
One light reading rarely identifies a unique room.



Current room	dark	medium	bright
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

The sensor likelihoods stay in the model even after the readings are known.

A Bayes net factors the joint into one conditional per node.



$$P(X_1, \ldots, X_n) = \prod_i P(X_i \mid \mathcal{P}(X_i))$$

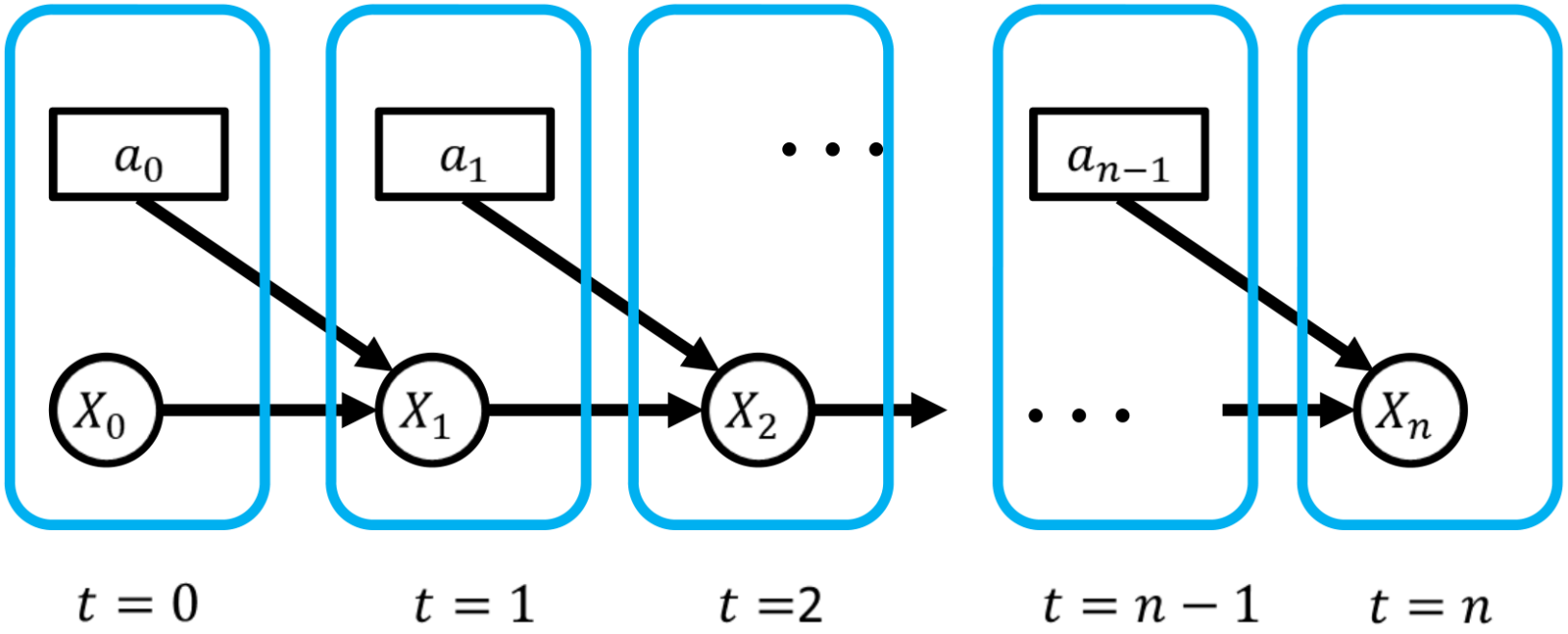
READ THE GRAPH

For each node, list its parents.
Those are precisely the variables after the conditioning bar.

$$P(W, X, Y, Z) = P(W \mid X, Z)P(X \mid Y, Z) \cdot P(Y \mid Z)P(Z)$$

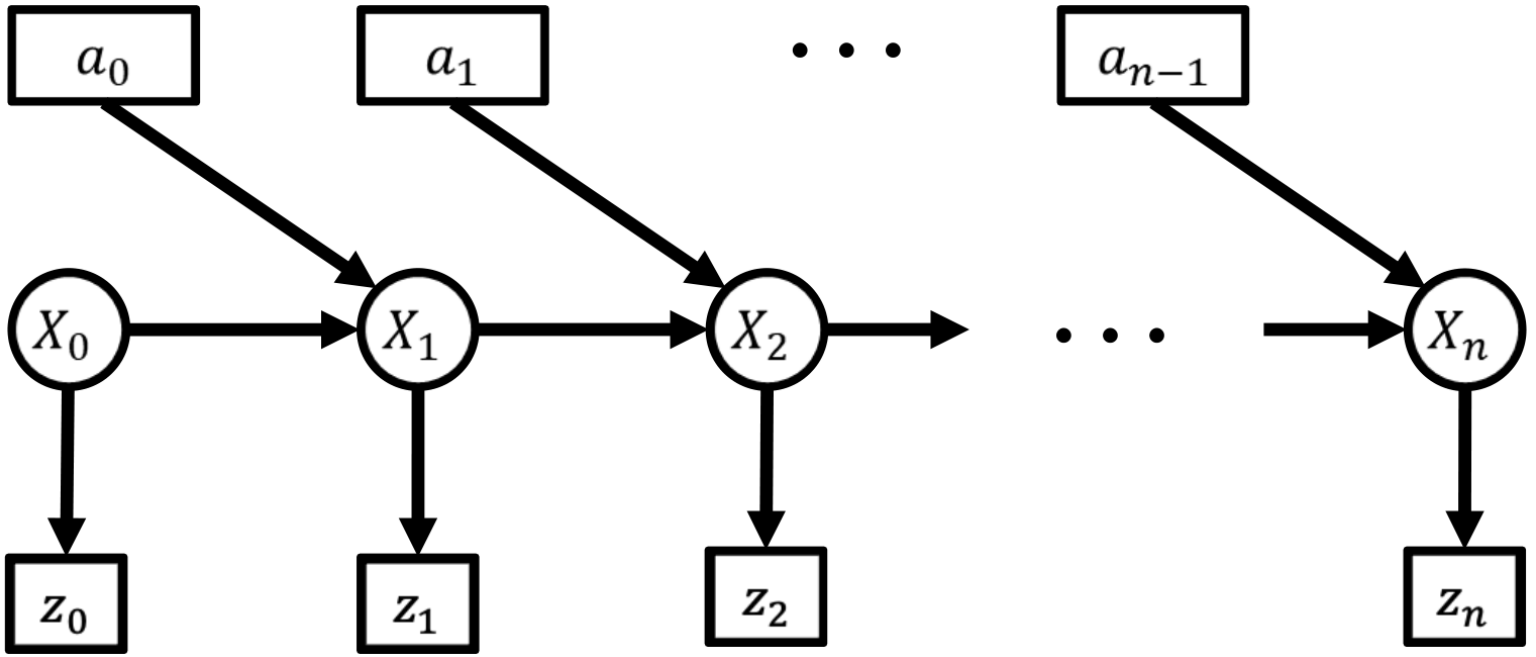
A dynamic Bayes net repeats the same structure at every time step.

The current state and action determine the distribution of the next state.



The drawing repeats in time; the motion model supplies each local conditional.

Hidden Markov Models: actions and measurements known; the states remain hidden.



Known actions

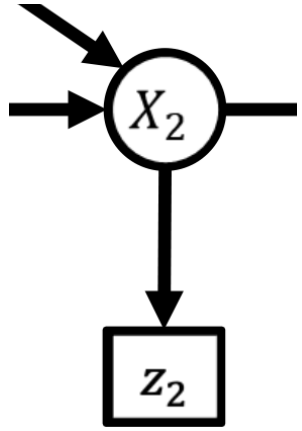
Most probable explanation (MPE)

Simple sensing

Today: exploit the graph structure instead of enumerating every history.

GTSAM stores a conditional probability table (CPT) as a *DiscreteConditional*.

Current room	dark	medium	bright
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



conditional = gtsam.DiscreteConditional(Z[2], [X[2]], vacuum.sensor_spec)

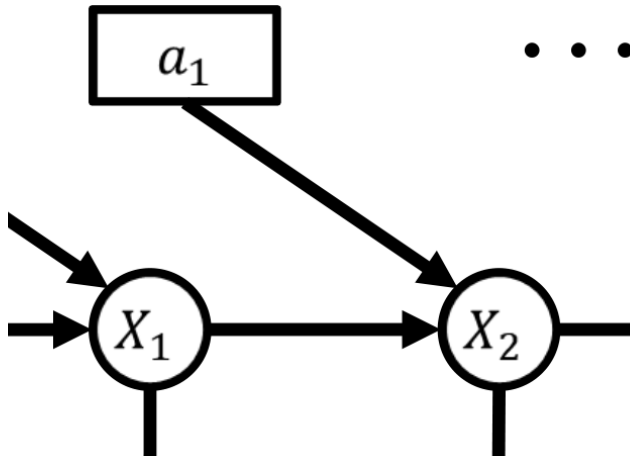
X[2], Z[2] are gtsam.Key variables, vacuum.sensor_spec is the table above with 15 entries

GTSAM stores a conditional probability table (CPT) as a *DiscreteConditional*.

```
# P(X[2]|X[1],A[1])
conditional = gtsam.DiscreteConditional(X[2], [X[1], A[1]], vacuum.action_spec)
```

We can narrow down to a 5x5 table by `choosing` the action:

```
actions = VARIABLES.assignment({A[1]: 'R', A[2]: 'U'})
conditional_a_1 = conditional.choose(actions)  # P(X[2]|X[1],A[1]='R')
```



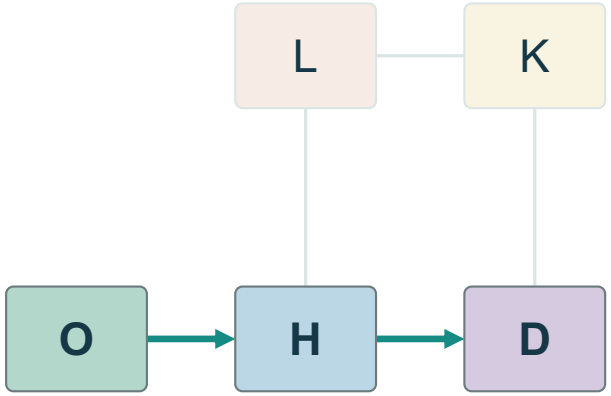
X[1], X[2], A[1] are gtsam.Key variables, vacuum.action_spec is the table with 100 entries

Score each possible sequence, take argmax.

KNOWN START: OFFICE · READINGS: MEDIUM, DARK, MEDIUM

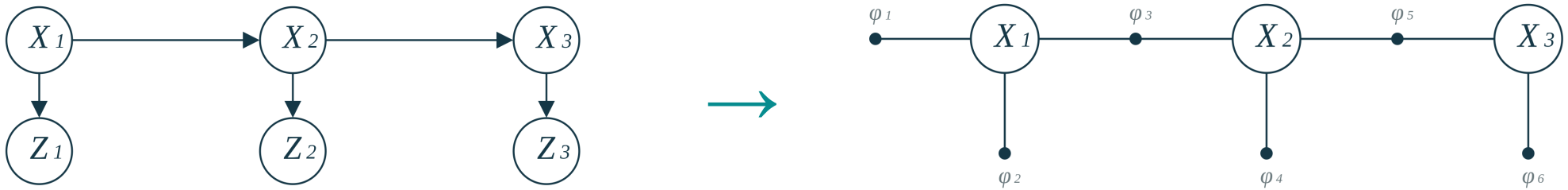
Candidate path	Joint score q(x)
O → O → O	0.00392
O → O → H	0.00224
O → H → H	0.00896
O → H → D	**0.28672**

MOST PROBABLE EXPLANATION



Same prior, actions, and measurements for every candidate.

Factor Graphs: focus on what we don't know.

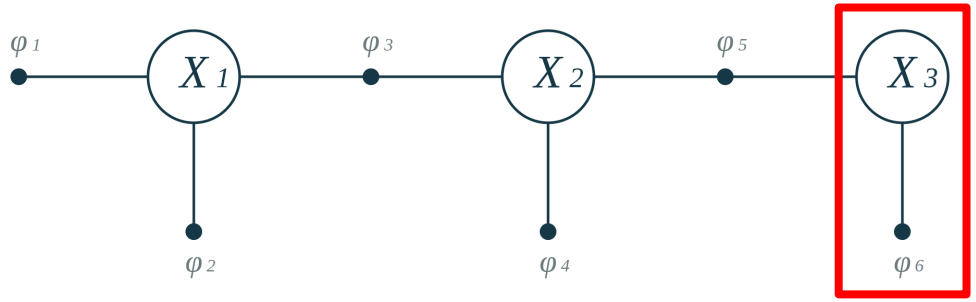


$$P(\mathcal{X} \mid \mathcal{Z}) \propto P(X_1)L(X_1; z_1)P(X_2 \mid X_1) \cdot L(X_2; z_2)P(X_3 \mid X_2)L(X_3; z_3)$$

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

Each observed reading survives as a likelihood factor on its hidden state.

The reading is fixed; its likelihood still depends on the room.



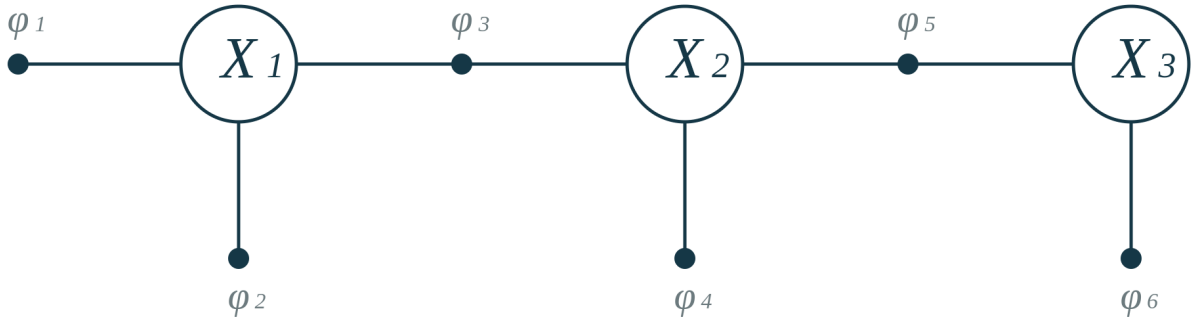
$$\phi_6(X_3) = L(X_3; z_3) = P(z_3 \mid X_3)$$

Current room	dark	medium	bright
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

Only X_3 varies now.
The known reading z_3 stays fixed.

Conditioning fixes the observation; it does not remove the evidence from the score.

Every factor is connected *only* to what it depends on



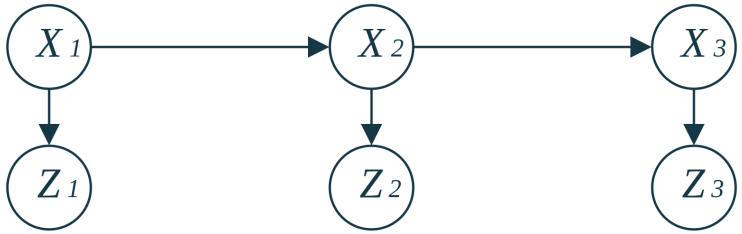
$$\phi(\mathcal{X}) = \prod_i \phi_i(\mathcal{X}_i)$$

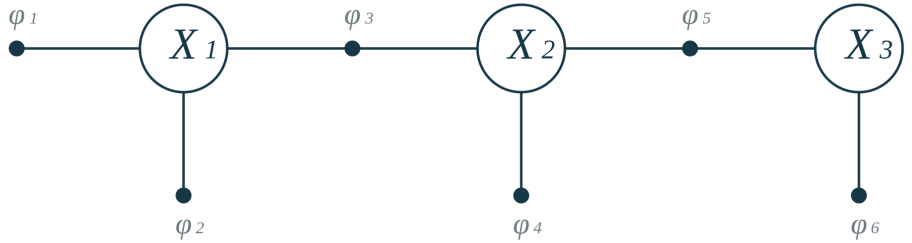
$$\mathcal{X}_1 = \{X_1\} \qquad \mathcal{X}_2 = \{X_1\}$$
$$\mathcal{X}_3 = \{X_1, X_2\}$$
$$\mathcal{X}_4 = \{X_2\}$$
$$\mathcal{X}_5 = \{X_2, X_3\}$$
$$\mathcal{X}_6 = \{X_3\}$$

A bipartite graph of variables and factors.
Each subset lists the neighbors of one factor.

The six subsets in this graph

Six factors encode the prior, the motion, and the evidence.



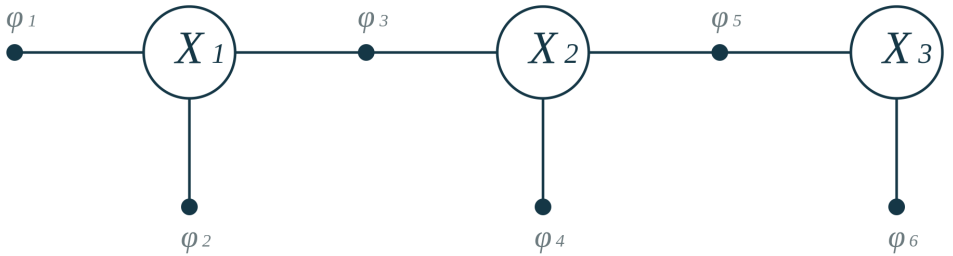


$$P(\mathcal{X} \mid \mathcal{Z}) \propto P(X_1)L(X_1; z_1)P(X_2 \mid X_1) \cdot L(X_2; z_2)P(X_3 \mid X_2)L(X_3; z_3)$$

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

GOLD: PRIOR **BLUE: TRANSITIONS** **TEAL: SENSOR LIKELIHOODS**

GTSAM stores a Factor Graph on discrete variables as a *DiscreteFactorGraph*.



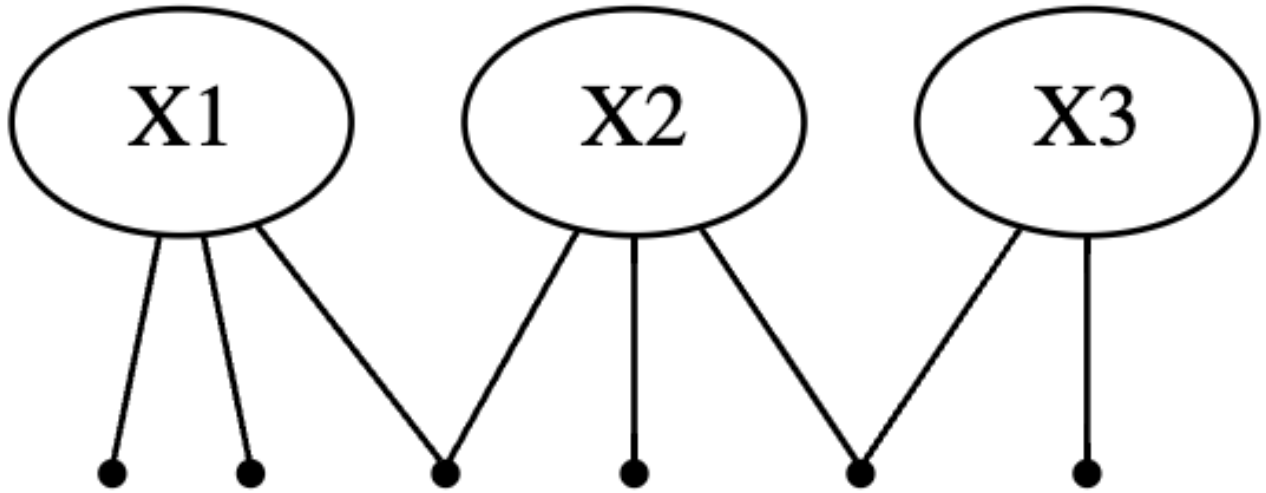

```
actions = VARIABLES.assignment({A[1]: 'R', A[2]: 'U'})
measurements = ['dark', 'medium', 'light']

graph = gtsam.DiscreteFactorGraph()
graph.add(X[1], "1 1 1 1 1")  # \phi(X_1) = P(X_1)
for k in range(1, N):
    conditional = gtsam.DiscreteConditional(X[k + 1], [X[k], A[k]], vacuum.action_spec)
    conditional_a_k = conditional.choose(actions)  # \phi(X,X+) = P(X+|X,A=a)
    graph.push_back(conditional_a_k) # every conditional can be used as a factor
for k, measurement in enumerate(measurements, start=1):
    conditional = gtsam.DiscreteConditional(Z[k], [X[k]], vacuum.sensor_spec)
    z_k = vacuum.light_levels.index(measurement)
    factor = conditional.likelihood(z_k)  # \phi(X) = P(Z=z|X) <<<<<< use `likelihood` for selecting a measurement
    graph.push_back(factor)
```

X[k], Z[k], A[k] are gtsam.Key variables

GTSAM stores the same product as a *DiscreteFactorGraph*.

```
show(graph)
```



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

THE GRAPH IS THE DATA STRUCTURE

Local factors are stored separately.
There is no expanded table of every possible history.

The same variables and local factors appear in the diagram and in the code.

A brute-force MPE solver is a simple loop

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

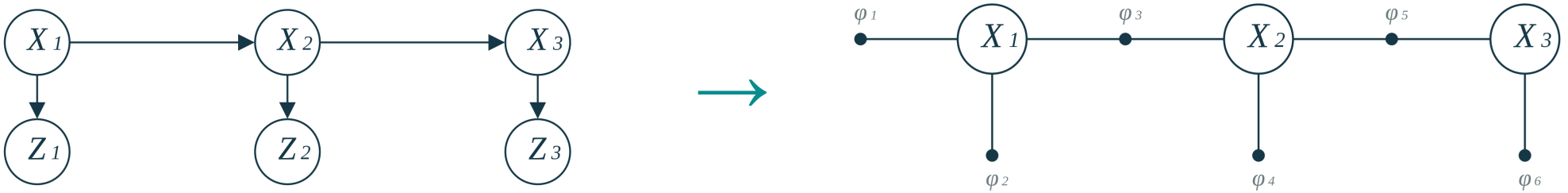
```
mpe_value = 0
mpe_trajectory = None
for x1 in vacuum.rooms:
    for x2 in vacuum.rooms:
        for x3 in vacuum.rooms:
            trajectory = VARIABLES.assignment({X[1]: x1, X[2]: x2, X[3]: x3})
            value = graph(trajectory)
            if value > mpe_value:
                mpe_value = 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

Can we do this in a better way?

After converting to a factor graph, it tells us which computations can stay local.



CONDITION ON WHAT IS GIVEN

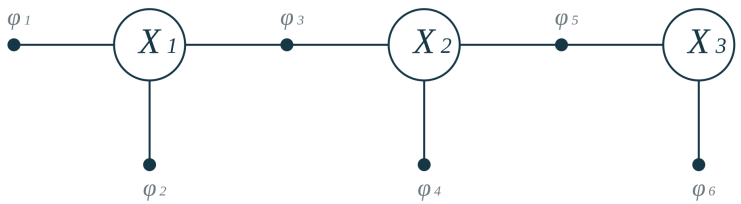
EXPLOIT THE REMAINING FACTORS

TWO INFERENCE OBJECTIVES

MPE keeps a best assignment.
Bayesian inference keeps the full posterior.

Only the factors touching X_1 belong inside the inner maximization.

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

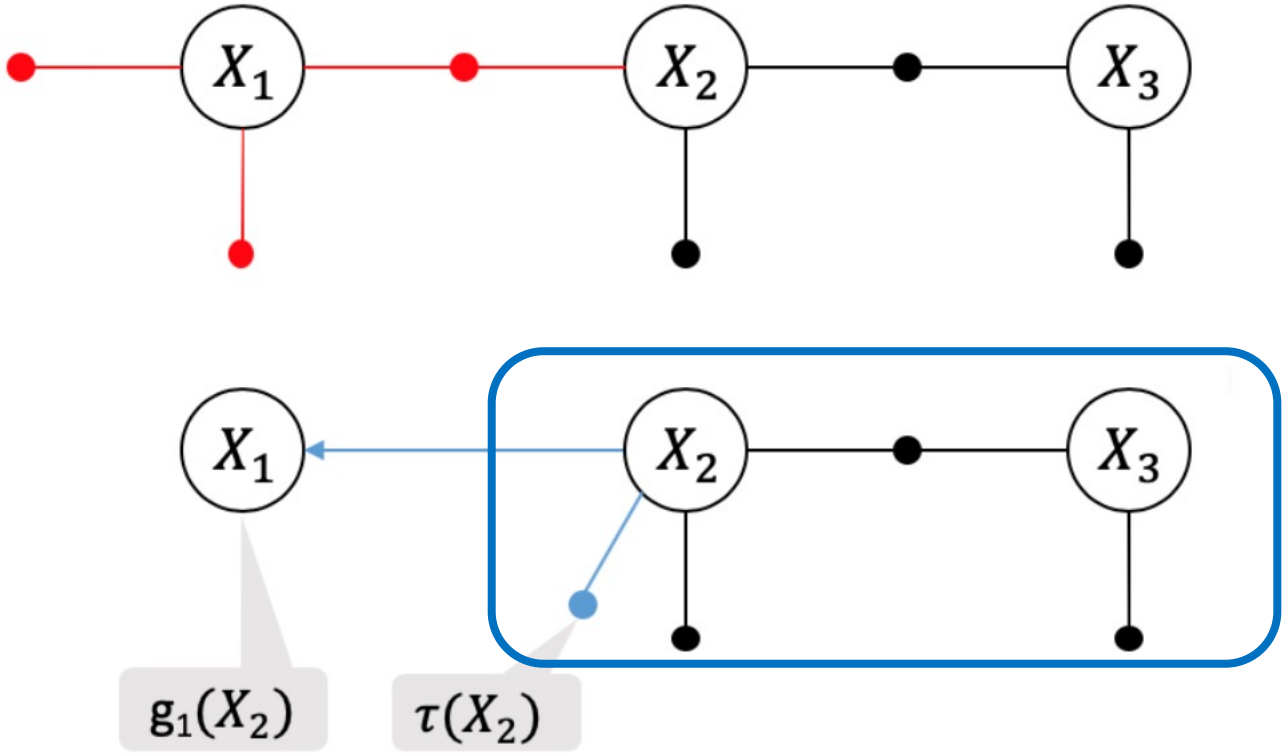


Red: the local product to eliminate.
Teal: its replacement factor.
The untouched factors remain outside the max.

A graphical elimination game

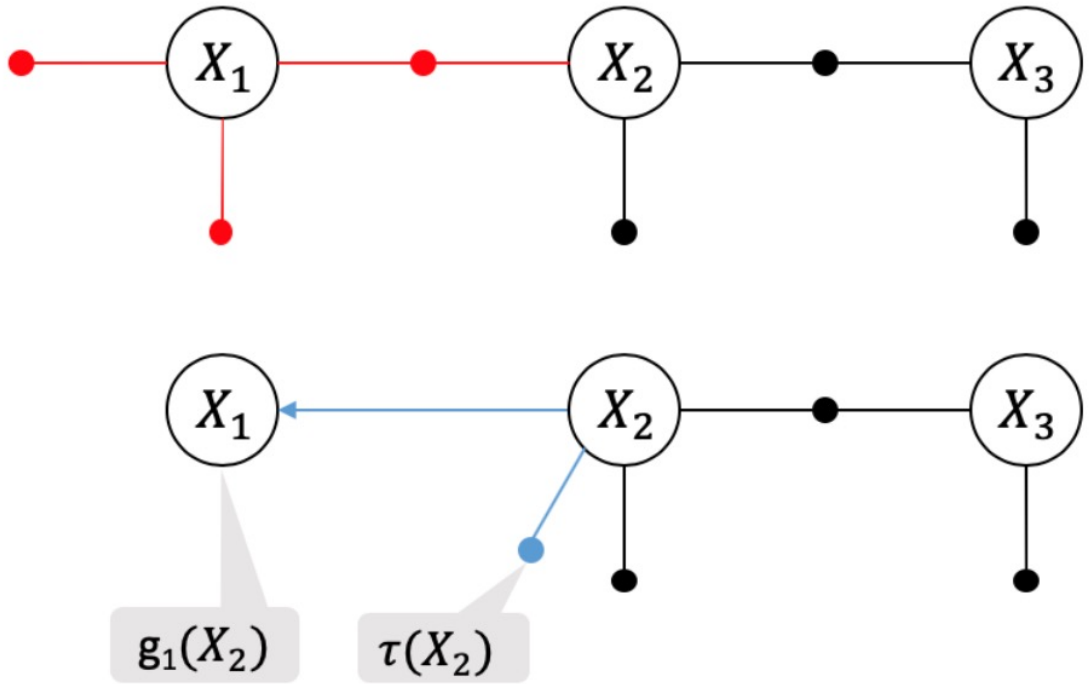
- We can represent the elimination of X_1 graphically:

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



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

Eliminating a variable means summarizing its effect on the variables left.



WHAT STAYS IN THE FACTOR GRAPH

A smaller factor $\tau(X_2)$ summarizes how well each possible X_2 can be supported.

WHAT WE SAVE FOR LATER

A lookup table $g_1(X_2)$ remembers which X_1 achieved that score.

Eliminate X_1 from the remaining computation—not from the final answer.

For each possible X_2, keep both a best score and the state that achieved it.

$$\tau(X_2) = \max_{X_1} \phi(X_1, X_2)$$

What is the best score?

$$g_1(X_2) = \arg\max_{X_1} \phi(X_1, X_2)$$

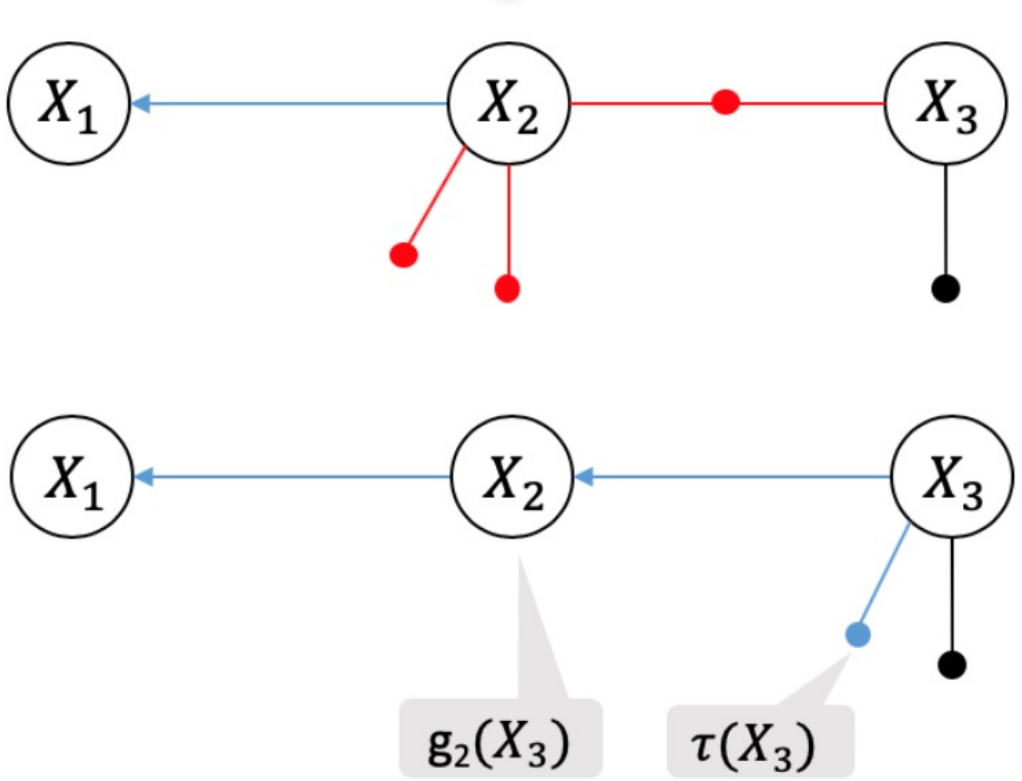
Which state achieved it?

X_2 =	L	K	O	H	D
score	$\tau(L)$	$\tau(K)$	$\tau(O)$	$\tau(H)$	$\tau(D)$
state	$g_1(L)$	$g_1(K)$	$g_1(O)$	$g_1(H)$	$g_1(D)$

The next room is not known yet. Store one answer for every possible next room.

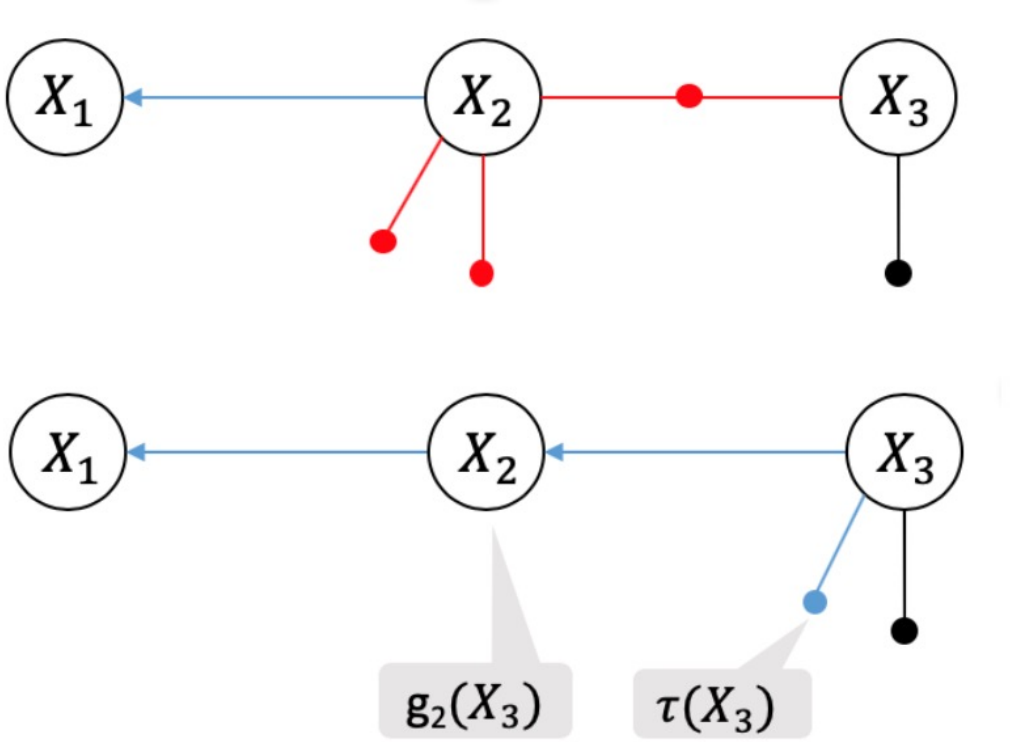
A graphical elimination game

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



Eliminating X_2 extends the directed structure and passes a new factor to X_3.

consume pass a factor save a dependency



MULTIPLY THE ACTIVE FACTORS

$$\phi(X_2, X_3) = \tau(X_2)\phi_4(X_2) \cdot \phi_5(X_2, X_3)$$

PASS THE REDUCED FACTOR

$$\tau(X_3) = \max_{X_2} \phi(X_2, X_3)$$

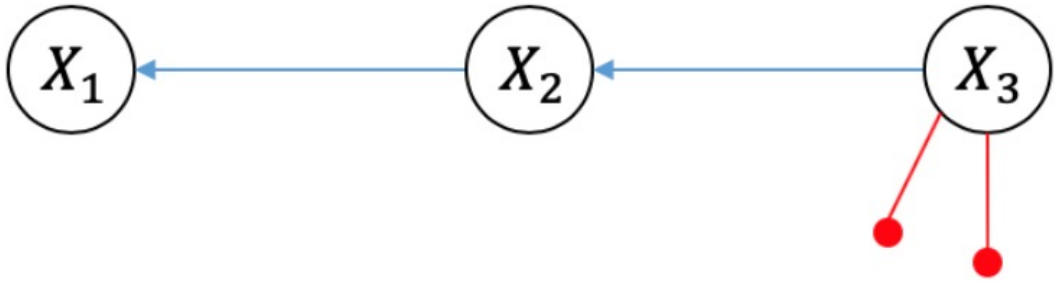
SAVE THE ELIMINATED VARIABLE

$$g_2(X_3) = \arg\max_{X_2} \phi(X_2, X_3)$$

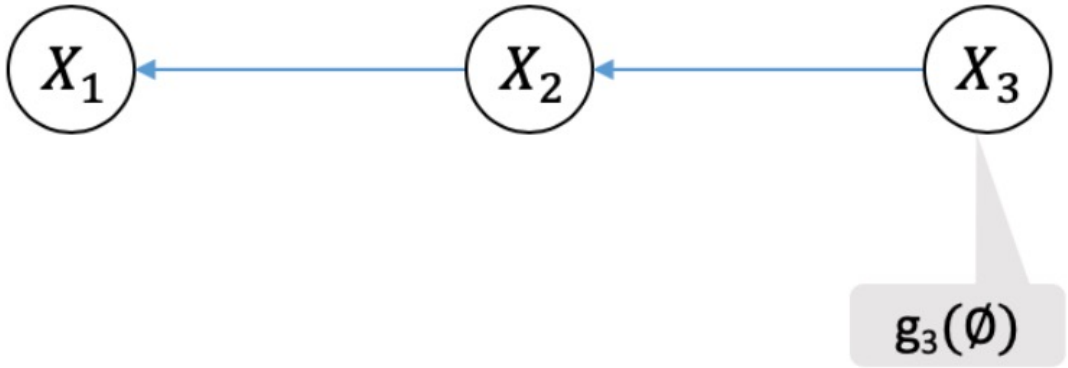
The incoming $\tau(X_2)$ is now an ordinary factor in the next local product.

A graphical elimination game

And finally, we eliminate X_3:

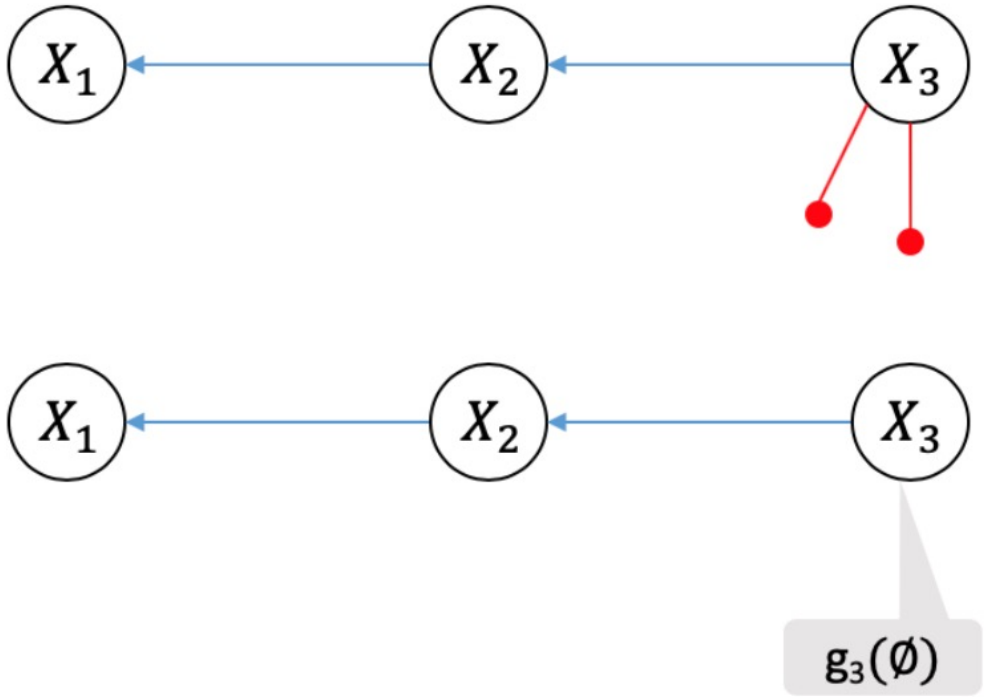


$$\phi(X_3) = \tau(X_3)\phi_6(X_3)$$



Eliminating X_3 leaves the final choice and completes the dependency chain.

consume pass a factor save a dependency



MULTIPLY THE ACTIVE FACTORS

$$\phi(X_3) = \tau(X_3)\phi_6(X_3)$$

PASS THE REDUCED FACTOR

$$\text{best score} = \max_{X_3} \phi(X_3)$$

SAVE THE ELIMINATED VARIABLE

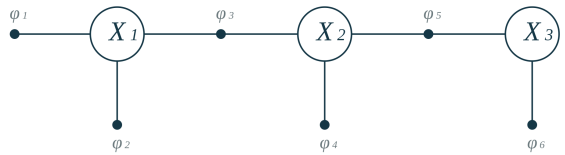
$$g_3(\emptyset) = \arg\max_{X_3} \phi(X_3)$$

No variables remain in the factor graph. The saved tables still describe the assignment.

Each local elimination saves a table and returns a smaller factor graph.

1 REMOVE THE FACTORS CONTAINING X_j

$$\mathcal{F}_j = \{\phi_i \in \Phi_{j:n} : X_j \in \mathcal{X}_i\}$$



2 FORM THEIR PRODUCT

$$\phi(X_j, X_{j+1}) \leftarrow \prod_{\phi_i \in \mathcal{F}_j} \phi_i(\mathcal{X}_i)$$

3 MAXIMIZE AND SAVE THE CHOICE

$$\tau(X_{j+1}) = \max_{X_j} \phi, \qquad g_j(X_{j+1}) = \arg\max_{X_j} \phi$$

4 PUT τ BACK; RETURN g_j AND THE REDUCED GRAPH

$$\Phi_{j+1:n} = (\Phi_{j:n} \setminus \mathcal{F}_j) \cup \{\tau\}$$

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 $j = 1 \ldots n$:
  - $g_j(X_{j+1}), \Phi_{j+1:n} \leftarrow \text{CreateLookupTable}(\Phi_{j:n}, X_j)$
  - return $g_1(X_2)g_2(X_3)\ldots g_n(\emptyset)$

`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 the lookup table $g_j(X_{j+1})$ and reduced graph $\Phi_{j+1:n}$

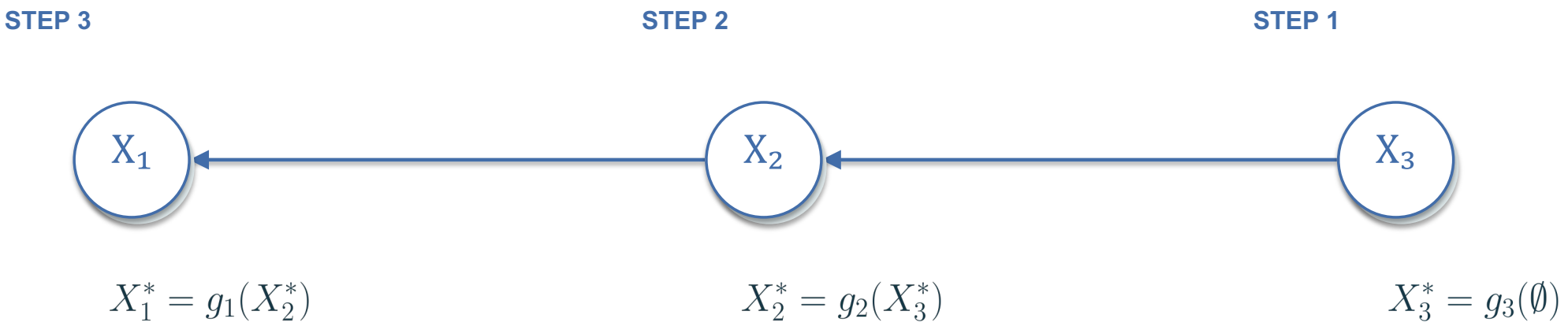
Back-substitution

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

Back-substitution follows the arrows from the last eliminated variable.



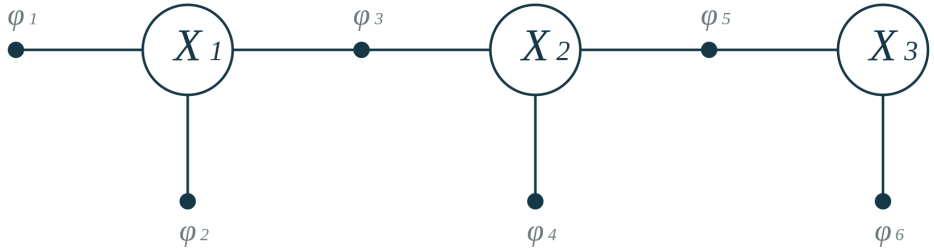
Look up X_1 after X_2 is known.

Look up X_2 after X_3 is known.

Choose X_3 first.

$$X^* = \left(g_1(g_2(g_3(\emptyset))),\ g_2(g_3(\emptyset)),\ g_3(\emptyset)\right)$$

GTSAM's optimizer returns the same maximizing assignment.




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

Variable	value
X1	Hallway
X2	Dining Room
X3	Kitchen

WHAT THE RESULT MEANS

One jointly compatible state sequence—not an independent best room at each time.

An assignment gives a value to every state variable in the graph.

Sum-product for HMMs

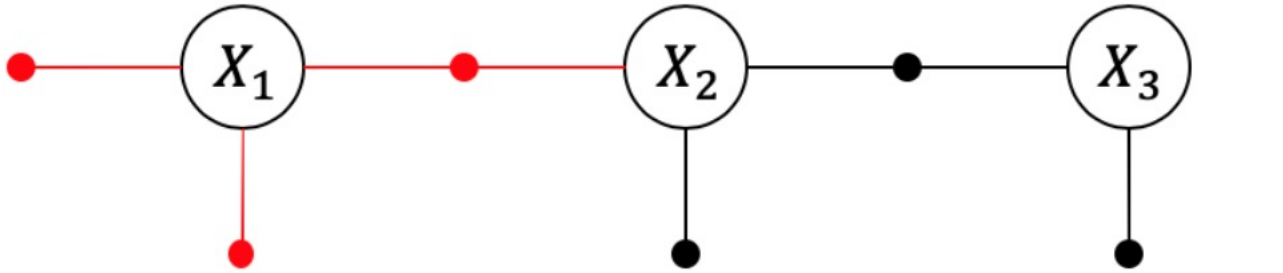
- What if we want *P(X|z,a)* rather than argmax *P(X|z,a)* ?
- Similar dynamic programming idea:

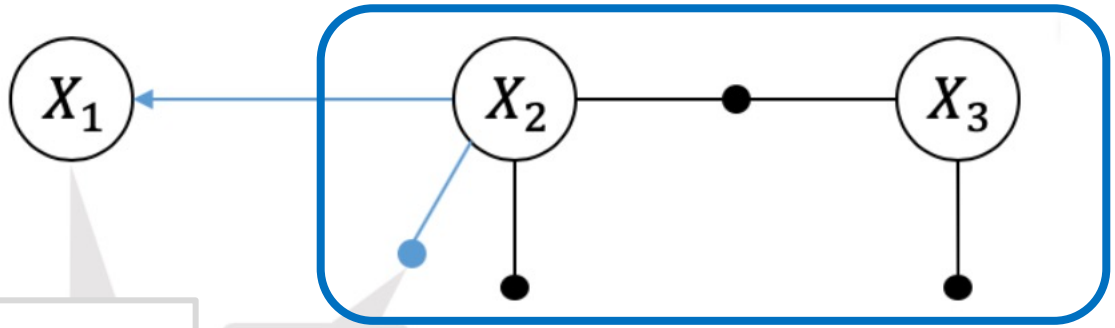
$$
\begin{aligned}
P(\mathcal{X}|\mathcal{Z}) &\propto \prod \phi_i(\mathcal{X}_i) \\
&\propto \phi_1(X_1)\phi_2(X_1)\phi_3(X_1,X_2)\phi_4(X_2)\phi_5(X_2,X_3)\phi_6(X_3) \\
&\propto \{\phi_1(X_1)\phi_2(X_1)\phi_3(X_1,X_2)\}\ \phi_4(X_2)\phi_5(X_2,X_3)\phi_6(X_3) \\
&\propto \{P(X_1|X_2,\mathcal{Z})\tau(X_2)\}\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

Sum-product looks the same graphically:

But eliminating now involves a sum and a conditional

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



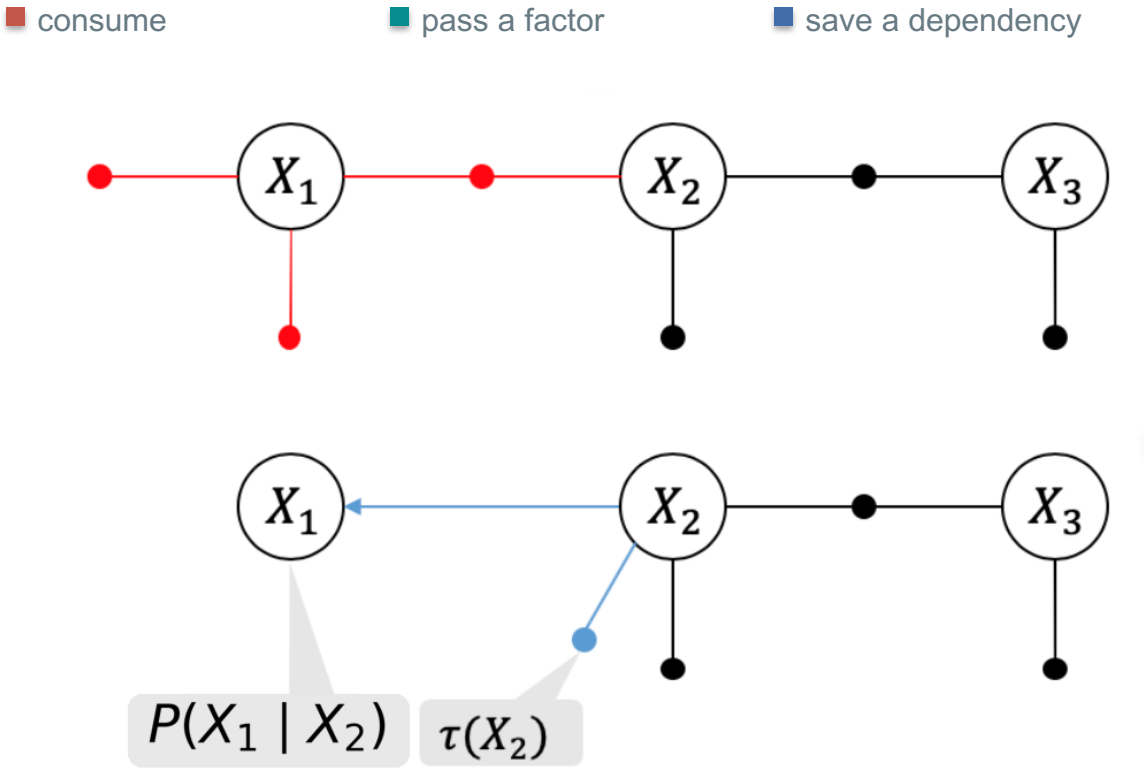


$$P(X_1|X_2) = \frac{\phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2)}{\tau(X_2)}$$

$\tau(X_2)$

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

Summing out X_1 creates a new factor and a conditional for the eliminated state.



MULTIPLY THE ACTIVE FACTORS

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

PASS THE REDUCED FACTOR

$$\tau(X_2) = \sum_{X_1} \phi(X_1, X_2)$$

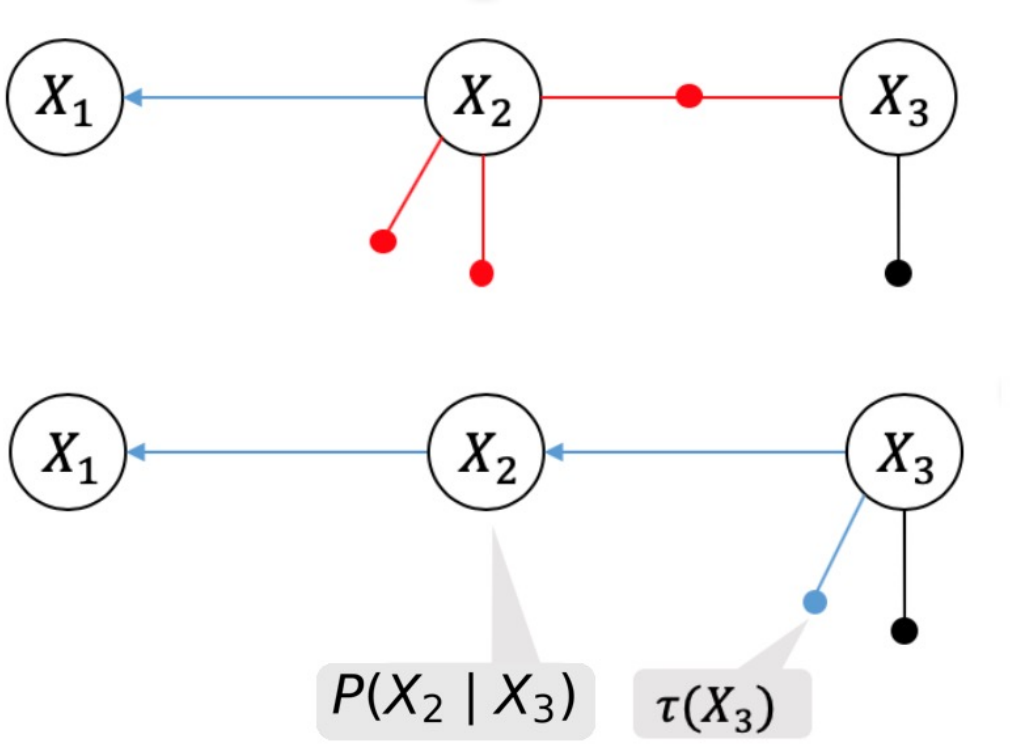
SAVE THE ELIMINATED VARIABLE

$$P(X_1 \mid X_2) = \frac{\phi(X_1, X_2)}{\tau(X_2)}$$

Divide by the sum to retain a conditional distribution for every supported X_2.

Summing out X_2 adds another conditional to the emerging Bayes net.

consume pass a factor save a dependency



MULTIPLY THE ACTIVE FACTORS

$$\phi(X_2, X_3) = \tau(X_2)\phi_4(X_2) \cdot \phi_5(X_2, X_3)$$

PASS THE REDUCED FACTOR

$$\tau(X_3) = \sum_{X_2} \phi(X_2, X_3)$$

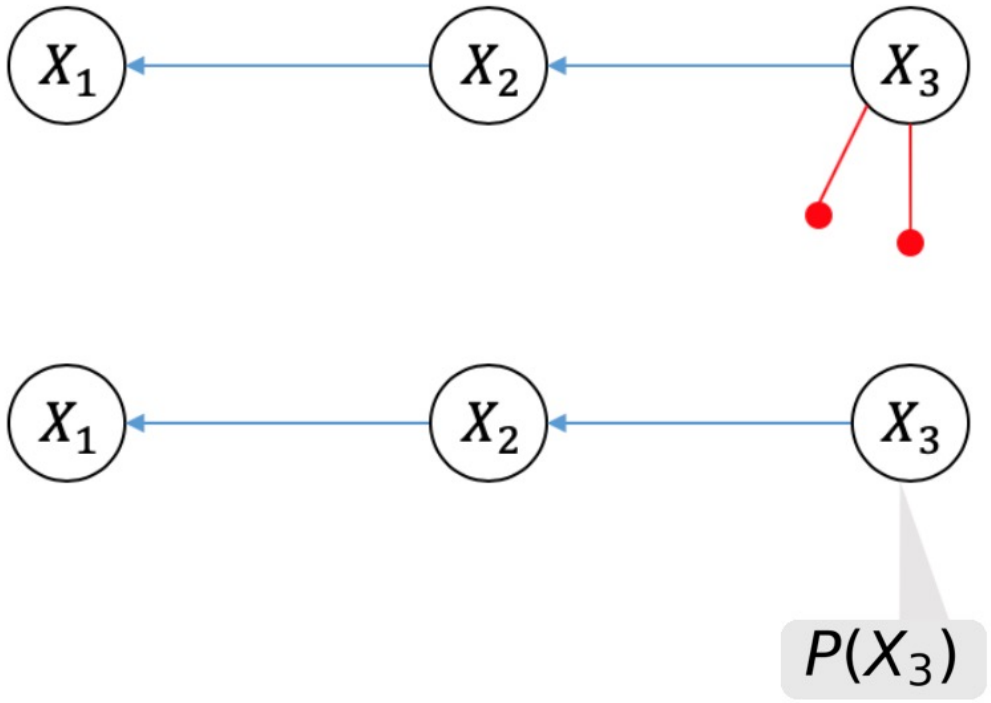
SAVE THE ELIMINATED VARIABLE

$$P(X_2 \mid X_3) = \frac{\phi(X_2, X_3)}{\tau(X_3)}$$

Same graph operation, different saved object: a conditional, not a maximizing lookup.

Normalizing the last factor completes the posterior Bayes net.

consume pass a factor save a dependency



MULTIPLY THE ACTIVE FACTORS

$$\phi(X_3) = \tau(X_3)\phi_6(X_3)$$

PASS THE REDUCED FACTOR

$$Z = \sum_{X_3} \phi(X_3)$$

SAVE THE ELIMINATED VARIABLE

$$P(X_3) = \frac{\phi(X_3)}{Z}$$

The last factor becomes the root distribution. All evidence is implicit in these conditionals.

The outer loop eliminates variables in the order we choose.

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

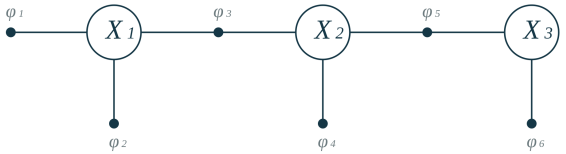
For j = 1, …, n:

Here we use the chain order $X_1, X_2, \ldots, X_n$.

$$\left(g_j(X_{j+1}), \Phi_{j+1:n}\right) \leftarrow \text{CreateLookupTable}(\Phi_{j:n}, X_j)$$

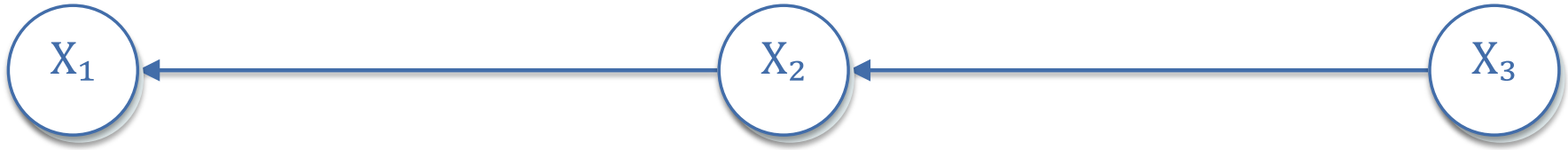
THEN RECOVER THE ASSIGNMENT

Start at the last eliminated variable.
Use each saved table only after its argument is known.



At the final step there is no next variable; the last table has the empty argument ∅.

The posterior's arrows record inference dependencies—not backwards robot motion.



$P(X_1 \mid X_2, \mathcal{Z})$ $P(X_2 \mid X_3, \mathcal{Z})$ $P(X_3 \mid \mathcal{Z})$

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

Eliminate left to right. Recover or sample right to left, following the saved dependencies.

Comparing max and sum-product:

Almost identical, replace max with sum:

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

- for $j = 1 \ldots n$:
  - $g_j(X_{j+1}), \Phi_{j+1:n} \leftarrow \text{CreateLookupTable}(\Phi_{j:n}, X_j)$
  - return $g_1(X_2)g_2(X_3)\ldots g_n(\emptyset)$

`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 the lookup table $g_j(X_{j+1})$ and reduced graph $\Phi_{j+1:n}$

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

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

`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 $\Phi_{j+1:n}$
- return the conditional $P(X_j|X_{j+1})$ and reduced graph $\Phi_{j+1:n}$

- Spot the differences ☺

The graph and product are the same; what we save makes the algorithms different.

MAX-PRODUCT

$$\tau = \max_{X_j} \phi$$

$$g_j = \arg\max_{X_j} \phi$$

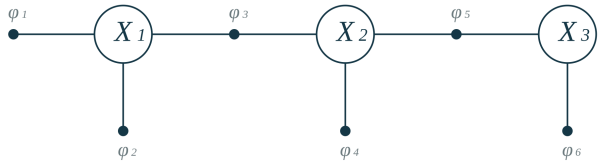
Output: a best assignment. Recover it by back-substitution.

SUM-PRODUCT

$$\tau = \sum_{X_j} \phi$$

$$P(X_j \mid X_{j+1}) = \frac{\phi}{\tau}$$

Output: a full posterior. Use it to query or sample.



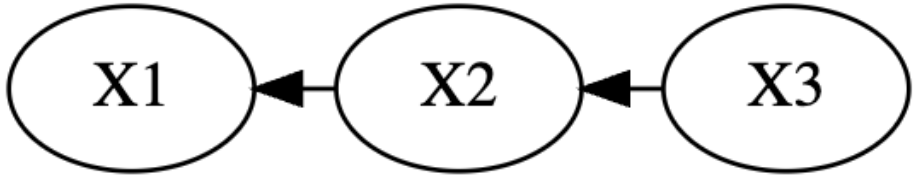
In GTSAM: *optimize* returns an assignment; *sumProduct* returns a distribution.

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

Variable	value
X1	Hallway
X2	Dining Room
X3	Kitchen

ONE STATE SEQUENCE

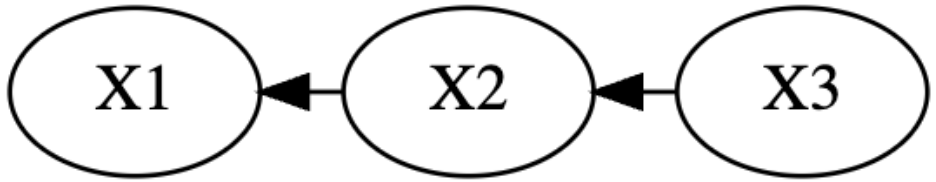
```
posterior = graph.sumProduct()
show(posterior, hints={"X": 1})
```



A POSTERIOR OVER STATE SEQUENCES

A posterior lets us sample alternative histories—not just one best path.

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




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

✓ 0.1s

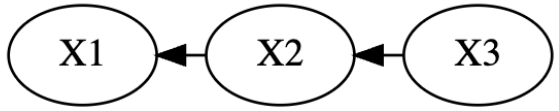
Variable	value
X1	Hallway
X2	Dining Room
X3	Kitchen

SAMPLE IN DEPENDENCY ORDER

Sample X_3 from the root. Then sample X_2 given X_3, and X_1 given X_2.

Sampling many histories reveals uncertainty at each point in time.

```
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):
        key = X[k][0]
        room_index = sample[key]
        counts[k-1][room_index] += 1 # base 0!
```

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

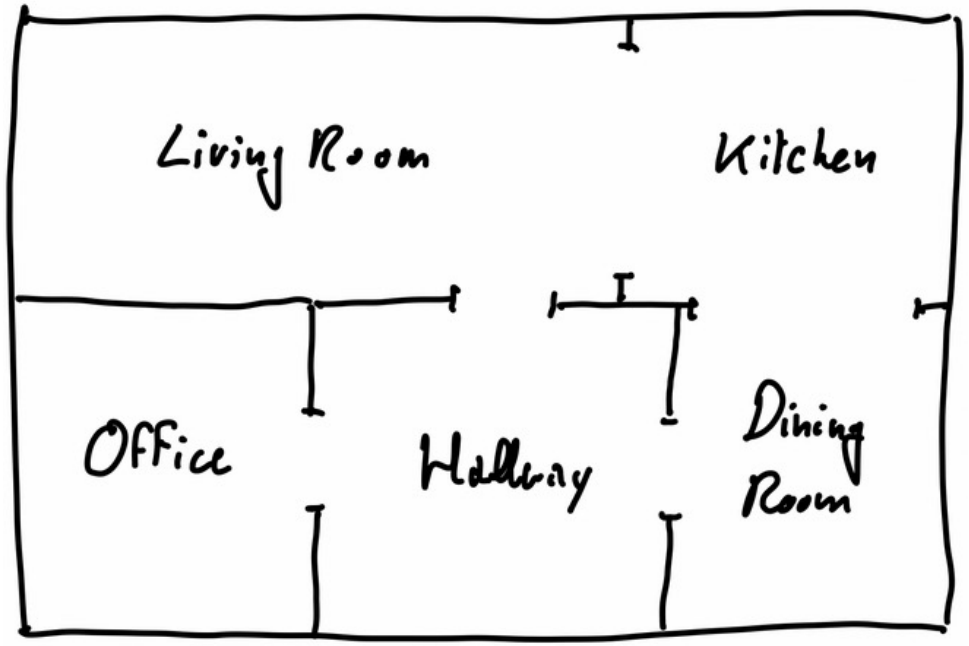
✓ 0.7s

	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

Each sample is a joint history; the count table summarizes empirical room marginals.

Inference gives us beliefs; rewards give us a reason to act.

FULL POSTERIOR INFERENCE



REWARD / COST FRAMEWORK

Next lecture changes one assumption: for an MDP, the current room is known.