

CPSC 540: Machine Learning

Exact Inference in Graphical Models

Mark Schmidt

University of British Columbia

Winter 2016

Admin

- **Assignment 3:**
 - Today is the final day to hand it in.
- **Assignment 4:**
 - Due on Tuesday.
 - Thursday is the last day to hand it in.
- **Midterm:**
 - **March 17 in class.**
 - Closed-book, two-page double-sided 'cheat sheet'.
 - Only covers topics from assignments A1-A4.
 - No requirement to pass.
 - Midterm from last year posted on Piazza.
 - Help session on March 16 from 3-5.
- **Final Project:**
 - Many of you are choosing project that are too big/hard.
 - In the project proposal, try to narrow down the scope:
 - Think of the final project as A6.
 - Main objective: show me you've learned something in this class, *and* explored a topic not covered in assignments.

Last Two Lectures: Directed and Undirected Graphical Models

- **DAG** models represent probability as ordered product of conditionals,

$$p(x) = \prod_{j=1}^d p(x_j | x_{\text{pa}(j)}),$$

and are also known as “Bayesian networks” and “belief networks”.

Last Two Lectures: Directed and Undirected Graphical Models

- **DAG** models represent probability as ordered product of conditionals,

$$p(x) = \prod_{j=1}^d p(x_j | x_{\text{pa}(j)}),$$

and are also known as “Bayesian networks” and “belief networks”.

- **UGMs** represent probability as product of non-negative potentials,

$$p(x) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \phi_c(x_c),$$

and are also known as “Markov random fields” and “Markov networks”.

- Models are useful for **density estimation** and **structured prediction**.

Markov Chains and Markov Property

- In Markov chains, we define the probability of x as

$$p(x) = p(x_1) \prod_{j=2}^d p(x_j | x_{j-1}),$$

which is a DAG model.



Markov Chains and Markov Property

- In Markov chains, we define the probability of x as

$$p(x) = p(x_1) \prod_{j=2}^d p(x_j | x_{j-1}),$$

which is a DAG model.



- From d-separation, we get the usual **Markov property**

$$p(x_j | x_{1:j-1}) = p(x_j | x_{j-1}),$$

that you're independent of the past given the last time step.

Markov Chains and Markov Property

- In Markov chains, we define the probability of x as

$$p(x) = p(x_1) \prod_{j=2}^d p(x_j | x_{j-1}),$$

which is a DAG model.



- From d-separation, we get the usual **Markov property**

$$p(x_j | x_{1:j-1}) = p(x_j | x_{j-1}),$$

that you're independent of the past given the last time step.

- A generalization of this property for general DAG models is:

$$p(x_j | x_{1:j-1}) = p(x_j | x_{\text{pa}(j)}).$$

Markov Chains and Markov Property

- For chain-structured UGMs,

$$p(x) = \frac{1}{Z} \prod_{j=2}^d \phi_{j,j-1}(x_j, x_{j=1}),$$

we don't have the usual Markov property:

- x_j might depend on future given past because of Z .



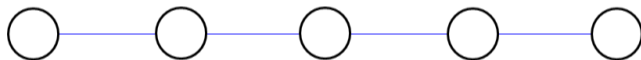
Markov Chains and Markov Property

- For chain-structured UGMs,

$$p(x) = \frac{1}{Z} \prod_{j=2}^d \phi_{j,j-1}(x_j, x_{j=1}),$$

we don't have the usual Markov property:

- x_j might depend on future given past because of Z .



- But for UGMs we have the **local Markov property**,

$$p(x_j | x_{\{1:d\} \setminus j}) = p(x_j | x_{\text{nei}(j)}),$$

where $\text{nei}(j)$ are the neighbours in the graph (**Markov blanket**).

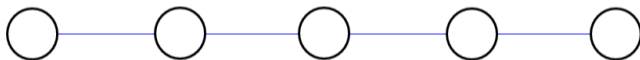
Markov Chains and Markov Property

- For chain-structured UGMs,

$$p(x) = \frac{1}{Z} \prod_{j=2}^d \phi_{j,j-1}(x_j, x_{j-1}),$$

we don't have the usual Markov property:

- x_j might depend on future given past because of Z .



- But for UGMs we have the **local Markov property**,

$$p(x_j | x_{\{1:d\} \setminus j}) = p(x_j | x_{\text{nei}(j)}),$$

where $\text{nei}(j)$ are the neighbours in the graph (**Markov blanket**).

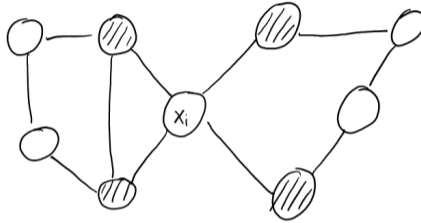
- For chain-structured UGMs, we thus have

$$p(x_j | x_{\{1:d\} \setminus j}) = p(x_j | x_{j-1}, x_{j+1}),$$

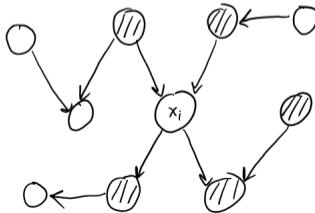
that you're independent of the past/future given last/next time.

Markov Blanket

- **Markov blanket** in UGMs is all neighbours in the graphs:



- Markov blanket in DAGs is all parents, children, and **co-parents**:



Decoding, Inference, and Sampling in UGMs

- Last time we introduced 3 common tasks we want to do with UGMs:

Decoding, Inference, and Sampling in UGMs

- Last time we introduced 3 common tasks we want to do with UGMs:
 - ① **Decoding**: Compute the optimal configuration,

$$\operatorname{argmax}_x \{p(x_1, x_2, \dots, x_n)\}.$$

Decoding, Inference, and Sampling in UGMs

- Last time we introduced 3 common tasks we want to do with UGMs:

- 1 **Decoding**: Compute the optimal configuration,

$$\operatorname{argmax}_x \{p(x_1, x_2, \dots, x_n)\}.$$

- 2 **Inference**: Compute partition function and univariate marginals,

$$Z = \sum_x \prod_{c \in C} \phi_c(x_c), \quad p(x_j = s) = \sum_{x | x_j = s} p(x).$$

Decoding, Inference, and Sampling in UGMs

- Last time we introduced 3 common tasks we want to do with UGMs:

- 1 **Decoding**: Compute the optimal configuration,

$$\operatorname{argmax}_x \{p(x_1, x_2, \dots, x_n)\}.$$

- 2 **Inference**: Compute partition function and univariate marginals,

$$Z = \sum_x \prod_{c \in C} \phi_c(x_c), \quad p(x_j = s) = \sum_{x | x_j = s} p(x).$$

- 3 **Sampling**: Generate x according from the distribution:

$$x \sim p(x).$$

- All 3 are NP-hard in discrete UGMs.
- Even computing $p(x)$ is NP-hard if we don't have Z .

Decoding, Inference, and Sampling in DAGs

- How hard are these operations in discrete DAG models?

Decoding, Inference, and Sampling in DAGs

- How hard are these operations in discrete DAG models?
 - ① **Decoding**: NP-hard (need to account for states of all variables).

Decoding, Inference, and Sampling in DAGs

- How hard are these operations in discrete DAG models?
 - ① **Decoding**: NP-hard (need to account for states of all variables).
 - ② **Inference**: Easy in the unconditional case.
 - $Z = 1$ since distribution is already normalized.

Decoding, Inference, and Sampling in DAGs

- How hard are these operations in discrete DAG models?
 - ① **Decoding**: NP-hard (need to account for states of all variables).
 - ② **Inference**: Easy in the unconditional case.
 - $Z = 1$ since distribution is already normalized.
 - $p(x_j = s)$ defined recursively via **Chapman-Kolmogorov** equations:

$$p(x_j = s) = \sum_{x_{\text{pa}(j)}} p(x_j = s, x_{\text{pa}(j)}) = \sum_{x_{\text{pa}(j)}} \underbrace{p(x_j = s | x_{\text{pa}(j)})}_{\text{given}} p(x_{\text{pa}(j)}),$$

Decoding, Inference, and Sampling in DAGs

- How hard are these operations in discrete DAG models?
 - ① **Decoding**: NP-hard (need to account for states of all variables).
 - ② **Inference**: Easy in the unconditional case.
 - $Z = 1$ since distribution is already normalized.
 - $p(x_j = s)$ defined recursively via **Chapman-Kolmogorov** equations:

$$p(x_j = s) = \sum_{x_{\text{pa}(j)}} p(x_j = s, x_{\text{pa}(j)}) = \sum_{x_{\text{pa}(j)}} \underbrace{p(x_j = s | x_{\text{pa}(j)})}_{\text{given}} p(x_{\text{pa}(j)}),$$

and by independence of parents from unobserved children we have

$$p(x_{\text{pa}(j)}) = \prod_{k \in \text{pa}(j)} p(x_k | x_{\text{pa}(k)}),$$

which is a product of marginals for $k < j$ and conditionals that are given:
(Sequentially compute $p(x_j = s)$ for each s from $j = 1$ to d .)

Decoding, Inference, and Sampling in DAGs

- How hard are these operations in discrete DAG models?
 - ① **Decoding**: NP-hard (need to account for states of all variables).
 - ② **Inference**: Easy in the unconditional case.
 - $Z = 1$ since distribution is already normalized.
 - $p(x_j = s)$ defined recursively via **Chapman-Kolmogorov** equations:

$$p(x_j = s) = \sum_{x_{\text{pa}(j)}} p(x_j = s, x_{\text{pa}(j)}) = \sum_{x_{\text{pa}(j)}} \underbrace{p(x_j = s | x_{\text{pa}(j)})}_{\text{given}} p(x_{\text{pa}(j)}),$$

and by independence of parents from unobserved children we have

$$p(x_{\text{pa}(j)}) = \prod_{k \in \text{pa}(j)} p(x_k | x_{\text{pa}(k)}),$$

which is a product of marginals for $k < j$ and conditionals that are given:
(Sequentially compute $p(x_j = s)$ for each s from $j = 1$ to d .)

- ③ **Sampling**: Easy in the unconditional case.
 - **Ancestral sampling**: If we want to sample from $p(x_1, x_2) = p(x_2 | x_1)p(x_1)$,
Sample $x_1 \sim p(x_1)$, then sample $x_2 \sim p(x_2 | x_1)$.

Decoding, Inference, and Sampling in DAGs

- How hard are these operations in discrete DAG models?
 - ① **Decoding**: NP-hard (need to account for states of all variables).
 - ② **Inference**: Easy in the unconditional case.
 - $Z = 1$ since distribution is already normalized.
 - $p(x_j = s)$ defined recursively via **Chapman-Kolmogorov** equations:

$$p(x_j = s) = \sum_{x_{\text{pa}(j)}} p(x_j = s, x_{\text{pa}(j)}) = \sum_{x_{\text{pa}(j)}} \underbrace{p(x_j = s | x_{\text{pa}(j)})}_{\text{given}} p(x_{\text{pa}(j)}),$$

and by independence of parents from unobserved children we have

$$p(x_{\text{pa}(j)}) = \prod_{k \in \text{pa}(j)} p(x_k | x_{\text{pa}(k)}),$$

which is a product of marginals for $k < j$ and conditionals that are given:
(Sequentially compute $p(x_j = s)$ for each s from $j = 1$ to d .)

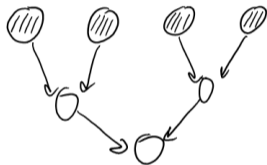
- ③ **Sampling**: Easy in the unconditional case.
 - **Ancestral sampling**: If we want to sample from $p(x_1, x_2) = p(x_2 | x_1)p(x_1)$,
Sample $x_1 \sim p(x_1)$, then sample $x_2 \sim p(x_2 | x_1)$.
 - General DAGs: sample variables in order $j = 1$ to d , conditioning on earlier samples.

Conditional Inference and Sampling in DAGs

- What about conditional inference/sampling in DAGs?
 - Could be easy or hard depending on what we condition on.

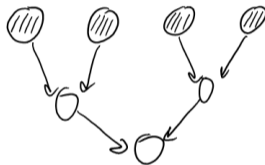
Conditional Inference and Sampling in DAGs

- What about conditional inference/sampling in DAGs?
 - Could be easy or hard depending on what we condition on.
- For example, still easy if condition on the first variables in the order:
 - Minor change to Chapman-Kolmogorov and ancestral sampling.

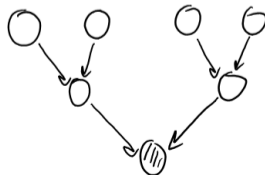


Conditional Inference and Sampling in DAGs

- What about conditional inference/sampling in DAGs?
 - Could be easy or hard depending on what we condition on.
- For example, still easy if condition on the first variables in the order:
 - Minor change to Chapman-Kolmogorov and ancestral sampling.



- NP-hard to condition on the last variables in the order:
 - Conditioning on descendent makes ancestors dependent.



Moralization: Converting DAGs to UGMs

- To address NP-hard problems, DAGs and UGMs use same methods.
- For DAGs, we typically just represent it as a UGM:

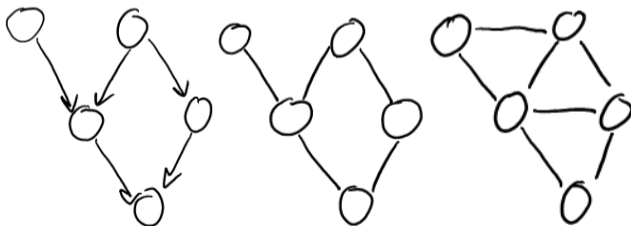
$$p(x) = \prod_{j=1}^d p(x_j | x_{\text{pa}(j)}) = \prod_{j=1}^d \phi_j(x_j, x_{\text{pa}(j)}).$$

Moralization: Converting DAGs to UGMs

- To address NP-hard problems, DAGs and UGMs use same methods.
- For DAGs, we typically just represent it as a UGM:

$$p(x) = \prod_{j=1}^d p(x_j | x_{\text{pa}(j)}) = \prod_{j=1}^d \phi_j(x_j, x_{\text{pa}(j)}).$$

- Graphically: we drop directions and “marry” parents ([moralization](#)).

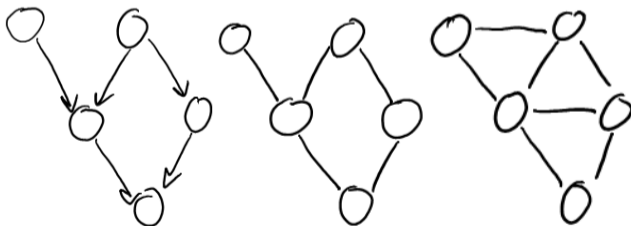


Moralization: Converting DAGs to UGMs

- To address NP-hard problems, DAGs and UGMs use same methods.
- For DAGs, we typically just represent it as a UGM:

$$p(x) = \prod_{j=1}^d p(x_j | x_{\text{pa}(j)}) = \prod_{j=1}^d \phi_j(x_j, x_{\text{pa}(j)}).$$

- Graphically: we drop directions and “marry” parents ([moralization](#)).



- This may lose some conditional independence information:
 - Models that be represented as DAGs or UGMs: “decomposable” and “triangulated”.
 - Includes chain-structured and fully-connected graphs.

Outline

- 1 DAGs vs. UGMs
- 2 Empty Graphs**
- 3 Chain-Structured Graphs
- 4 General Graphs

Inference By Enumeration

- Last time, **exact inference by table**:

Cathy	Heather	Mark	Allison	np(1)	np(2)	np(3)	np(4)	ep(1)	ep(2)	ep(3)	prodPot	Probability
right	right	right	right	1	9	1	9	2	2	2	648	0.17
wrong	right	right	right	3	9	1	9	1	2	2	972	0.26
right	wrong	right	right	1	1	1	9	1	1	2	18	0.00
wrong	wrong	right	right	3	1	1	9	2	1	2	108	0.03
right	right	wrong	right	1	9	3	9	2	1	1	486	0.13
wrong	right	wrong	right	3	9	3	9	1	1	1	729	0.19
right	wrong	wrong	right	1	1	3	9	1	2	1	54	0.01
wrong	wrong	wrong	right	3	1	3	9	2	2	1	324	0.09
right	right	right	wrong	1	9	1	1	2	2	1	36	0.01
wrong	right	right	wrong	3	9	1	1	1	2	1	54	0.01
right	wrong	right	wrong	1	1	1	1	1	1	1	1	0.00
wrong	wrong	right	wrong	3	1	1	1	2	1	1	6	0.00
right	right	wrong	wrong	1	9	3	1	2	1	2	108	0.03
wrong	right	wrong	wrong	3	9	3	1	1	1	2	162	0.04
right	wrong	wrong	wrong	1	1	3	1	1	2	2	12	0.00
wrong	wrong	wrong	wrong	3	1	3	1	2	2	2	72	0.02

- Table is too expensive** for decoding general UGMs.
 - We can't enumerate k^d possible configurations.

Inference without Edges

- To see idea behind more efficient methods, first let's consider empty graph:

$$p(x) = \frac{1}{Z} \prod_{j=1}^d \phi_j(x_j).$$

- If the x_j are binary, Z is sum of the 2^d products in the table:

$$Z = \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_d=0}^1 \prod_{j=1}^d \phi_j(x_j).$$

- If the x_j have k states, Z is the **sum of k^d** products over d variables.

Inference without Edges

- To see idea behind more efficient methods, first let's consider empty graph:

$$p(x) = \frac{1}{Z} \prod_{j=1}^d \phi_j(x_j).$$

- If the x_j are binary, Z is sum of the 2^d products in the table:

$$Z = \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_d=0}^1 \prod_{j=1}^d \phi_j(x_j).$$

- If the x_j have k states, Z is the **sum of k^d** products over d variables.
- This looks hard, but **independence lets us factorize** into product of d simple sums.
 - This trick was previously used in the EM notes.

Inference without Edges

- We can start by writing

$$Z = \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_{d-1}=0}^1 \sum_{x_d=0}^1 \left(\prod_{j=1}^{d-1} \phi_j(x_j) \right) \phi_d(x_d)$$

Inference without Edges

- We can start by writing

$$Z = \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_{d-1}=0}^1 \sum_{x_d=0}^1 \left(\prod_{j=1}^{d-1} \phi_j(x_j) \right) \phi_d(x_d)$$

- Now use $\sum_i ab_i = a \sum_i b_i$ to take terms not depending on x_d outside sum:

$$\begin{aligned} Z &= \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_{d-1}=0}^1 \prod_{j=1}^{d-1} \phi_j(x_j) \sum_{x_d=0}^1 \phi_d(x_d) \\ &= \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_{d-1}=0}^1 \prod_{j=1}^{d-1} \phi_j(x_j) \underbrace{\sum_{x_d=0}^1 \phi_d(x_d)}_{Z_d} \end{aligned}$$

Inference without Edges

- We can start by writing

$$Z = \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_{d-1}=0}^1 \sum_{x_d=0}^1 \left(\prod_{j=1}^{d-1} \phi_j(x_j) \right) \phi_d(x_d)$$

- Now use $\sum_i ab_i = a \sum_i b_i$ to take terms not depending on x_d outside sum:

$$\begin{aligned} Z &= \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_{d-1}=0}^1 \prod_{j=1}^{d-1} \phi_j(x_j) \sum_{x_d=0}^1 \phi_d(x_d) \\ &= \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_{d-1}=0}^1 \prod_{j=1}^{d-1} \phi_j(x_j) \underbrace{\sum_{x_d=0}^1 \phi_d(x_d)}_{Z_d} \end{aligned}$$

- Now take the constant Z_d outside all the sums,

$$Z = Z_d \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_{d-1}=0}^1 \prod_{j=1}^{d-1} \phi_j(x_j)$$

Inference without Edges

- So we have that

$$Z = Z_d \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_{d-1}=0}^1 \prod_{j=1}^{d-1} \phi_j(x_j).$$

Inference without Edges

- So we have that

$$Z = Z_d \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_{d-1}=0}^1 \prod_{j=1}^{d-1} \phi_j(x_j).$$

- If we repeat these steps we obtain

$$Z = Z_d Z_{d-1} \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_{d-2}=0}^1 \prod_{j=1}^{d-2} \phi_j(x_j),$$

Inference without Edges

- So we have that

$$Z = Z_d \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_{d-1}=0}^1 \prod_{j=1}^{d-1} \phi_j(x_j).$$

- If we repeat these steps we obtain

$$Z = Z_d Z_{d-1} \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_{d-2}=0}^1 \prod_{j=1}^{d-2} \phi_j(x_j),$$

and if we keep going we get

$$Z = Z_d Z_{d-1} \cdots Z_1 = \prod_{j=1}^d Z_j.$$

Inference without Edges

- Plugging in the definition of Z_j we get

$$Z = \prod_{j=1}^d \sum_{x_j=0}^1 \phi_j(x_j),$$

so for independent variables Z is a product of d two-term sums.

- If each variable has k states, it costs $O(dk)$ to compute.

Inference without Edges

- Plugging in the definition of Z_j we get

$$Z = \prod_{j=1}^d \sum_{x_j=0}^1 \phi_j(x_j),$$

so for independent variables Z is a product of d two-term sums.

- If each variable has k states, it costs $O(dk)$ to compute.
- By similar logic, we have $p(x_j) = \phi_j(x_j)/Z_j$ and can thus be computed in $O(s)$.
- We could plug this back into the UGM to get

$$\begin{aligned} p(x) &= \frac{1}{Z} \prod_{j=1}^d \phi_j(x_j) = \frac{1}{\prod_{j=1}^d Z_j} \prod_{j=1}^d \phi_j(x_j) \\ &= \prod_{j=1}^d \frac{\phi_j(x_j)}{Z_j} = \prod_{j=1}^d p(x_j), \end{aligned}$$

and this DAG representation allows ancestral sampling in $O(dk)$.

Decoding and Inference without Edges

- Since $\max_i \{ab_i\} = a \max_i \{b_i\}$ for $a \geq 0$, can use same logic for **decoding**:

$$\begin{aligned}\tilde{p}(x^*) &= \max_x p(x) \\ &= \max_{x_1} \max_{x_2} \cdots \max_{x_{d-1}} \max_{x_d} \prod_{j=1}^d \phi_j(x_j) \\ &= \max_{x_1} \max_{x_2} \cdots \max_{x_{d-1}} \prod_{j=1}^{d-1} \phi_j(x_j) \max_{x_d} \phi_d(x_d)\end{aligned}$$

Decoding and Inference without Edges

- Since $\max_i \{ab_i\} = a \max_i \{b_i\}$ for $a \geq 0$, can use same logic for **decoding**:

$$\begin{aligned}
 \tilde{p}(x^*) &= \max_x p(x) \\
 &= \max_{x_1} \max_{x_2} \cdots \max_{x_{d-1}} \max_{x_d} \prod_{j=1}^d \phi_j(x_j) \\
 &= \max_{x_1} \max_{x_2} \cdots \max_{x_{d-1}} \prod_{j=1}^{d-1} \phi_j(x_j) \max_{x_d} \phi_d(x_d) \\
 p(x^*) &= \left(\max_{x_d} \phi_d(x_d) \right) \max_{x_1} \max_{x_2} \cdots \max_{x_{d-1}} \prod_{j=1}^{d-1} \phi_j(x_j) \\
 &= \prod_{j=1}^d \max_{x_j} \phi_j(x_j),
 \end{aligned}$$

Decoding and Inference without Edges

- Since $\max_i \{ab_i\} = a \max_i \{b_i\}$ for $a \geq 0$, can use same logic for **decoding**:

$$\begin{aligned}
 \tilde{p}(x^*) &= \max_x p(x) \\
 &= \max_{x_1} \max_{x_2} \cdots \max_{x_{d-1}} \max_{x_d} \prod_{j=1}^d \phi_j(x_j) \\
 &= \max_{x_1} \max_{x_2} \cdots \max_{x_{d-1}} \prod_{j=1}^{d-1} \phi_j(x_j) \max_{x_d} \phi_d(x_d) \\
 p(x^*) &= \left(\max_{x_d} \phi_d(x_d) \right) \max_{x_1} \max_{x_2} \cdots \max_{x_{d-1}} \prod_{j=1}^{d-1} \phi_j(x_j) \\
 &= \prod_{j=1}^d \max_{x_j} \phi_j(x_j),
 \end{aligned}$$

Tedious way of showing you set x_j to maximize its own potential.

- “Generalized distributive law”: work for many “+” and “*” operations:
 - E.g., commutative semi-rings (Gaussian elimination, fast Fourier transform).

Outline

- 1 DAGs vs. UGMs
- 2 Empty Graphs
- 3 Chain-Structured Graphs**
- 4 General Graphs

Computer Science Graduate Markov Model

- Computer Science Graduate Careers Markov chain:
 - Variable x_1 can be in one of three states:

State	Probability	Description
Industry	0.60	They work for a company or own their own company.
Grad School	0.30	They are trying to get a Masters or PhD degree.
Video Games	0.10	They mostly play video games.

Computer Science Graduate Markov Model

- Computer Science Graduate Careers Markov chain:

- Variable x_1 can be in one of three states:

State	Probability	Description
Industry	0.60	They work for a company or own their own company.
Grad School	0.30	They are trying to get a Masters or PhD degree.
Video Games	0.10	They mostly play video games.

- Variable x_t only depends on x_{t-1} :

From/to	Video Games	Industry	Grad School	Video Games (with PhD)	Industry (with PhD)	Academia	Deceased
Video Games	0.08	0.90	0.01	0	0	0	0.01
Industry	0.03	0.95	0.01	0	0	0	0.01
Grad School	0.06	0.06	0.75	0.05	0.05	0.02	0.01
Video Games (with PhD)	0	0	0	0.30	0.60	0.09	0.01
Industry (with PhD)	0	0	0	0.02	0.95	0.02	0.01
Academia	0	0	0	0.01	0.01	0.97	0.01
Deceased	0	0	0	0	0	0	1

Computer Science Graduate Markov Model

- Computer Science Graduate Careers Markov chain:

- Variable x_1 can be in one of three states:

State	Probability	Description
Industry	0.60	They work for a company or own their own company.
Grad School	0.30	They are trying to get a Masters or PhD degree.
Video Games	0.10	They mostly play video games.

- Variable x_t only depends on x_{t-1} :

From/to	Video Games	Industry	Grad School	Video Games (with PhD)	Industry (with PhD)	Academia	Deceased
Video Games	0.08	0.90	0.01	0	0	0	0.01
Industry	0.03	0.95	0.01	0	0	0	0.01
Grad School	0.06	0.06	0.75	0.05	0.05	0.02	0.01
Video Games (with PhD)	0	0	0	0.30	0.60	0.09	0.01
Industry (with PhD)	0	0	0	0.02	0.95	0.02	0.01
Academia	0	0	0	0.01	0.01	0.97	0.01
Deceased	0	0	0	0	0	0	1

- So the probability of a sequence is

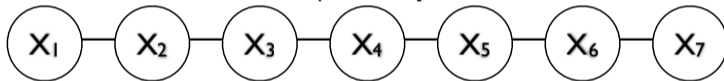
$$\begin{aligned}
 p(x_1, x_2, \dots, x_n) &= p(x_1)p(x_2|x_1)p(x_3|x_2, x_1) \dots p(x_n|x_{n-1}, x_{n-2}, \dots, x_1) \\
 &= p(x_1)p(x_2|x_1)p(x_3|x_2) \dots p(x_n|x_{n-1}).
 \end{aligned}$$

Markov Chain Models

- This is a special case of a UGM

$$p(x_1, x_2, \dots, x_n) = \phi_1(x_1) \prod_{i=2}^n \phi(x_i, x_{i-1}),$$

with a chain-structured dependency:

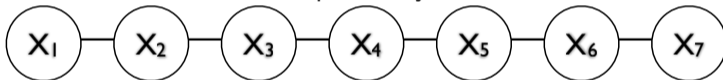


Markov Chain Models

- This is a special case of a UGM

$$p(x_1, x_2, \dots, x_n) = \phi_1(x_1) \prod_{i=2}^n \phi(x_i, x_{i-1}),$$

with a chain-structured dependency:



- **Homogeneous chain**: edge potentials are constant across time.
- Markov chains are ubiquitous in sequence/time-series models:

9 Applications

- 9.1 Physics
- 9.2 Chemistry
- 9.3 Testing
- 9.4 Speech Recognition
- 9.5 Information sciences
- 9.6 Queueing theory
- 9.7 Internet applications
- 9.8 Statistics
- 9.9 Economics and finance
- 9.10 Social sciences
- 9.11 Mathematical biology
- 9.12 Genetics
- 9.13 Games
- 9.14 Music
- 9.15 Baseball
- 9.16 Markov text generators

General Chain-Structured UGM

- The general class of chain-structured UGMs is

$$p(x_1, x_2, \dots, x_n) \propto \prod_{i=1}^n \phi_i(x_i) \prod_{i=2}^n \phi_{i,i-1}(x_i, x_{i-1}),$$

(x_t could depend on future things that might happen)

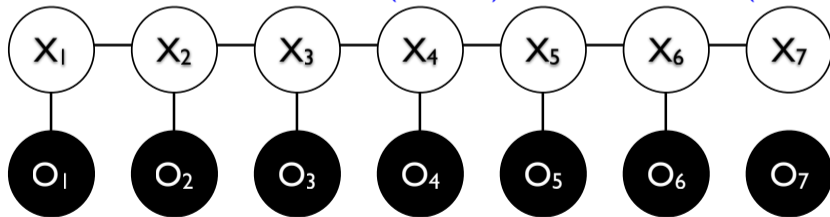
General Chain-Structured UGM

- The general class of chain-structured UGMs is

$$p(x_1, x_2, \dots, x_n) \propto \prod_{i=1}^n \phi_i(x_i) \prod_{i=2}^n \phi_{i,i-1}(x_i, x_{i-1}),$$

(x_t could depend on future things that might happen)

- Includes **hidden Markov models (discrete)** and **Kalman filters (Gaussian)**:



- O_i are observations (included in ϕ_i) and x_j are hidden states you want.
- Probably the most widely-used time-series models.

Applications of HMMs and Kalman Filters

Applications [edit]

HMMs can be applied in many fields where the goal is to recover a data sequence that is not immediately observable (but other data that depend on the sequence are).

Applications include:

- . Single Molecule Kinetic analysis^[16]
- . Cryptanalysis
- . Speech recognition
- . Speech synthesis
- . Part-of-speech tagging
- . Document Separation in scanning solutions
- . Machine translation
- . Partial discharge
- . Gene prediction
- . Alignment of bio-sequences
- . Time Series Analysis
- . Activity recognition
- . Protein folding^[17]
- . Metamorphic Virus Detection^[18]
- . DNA Motif Discovery^[19]

Applications of HMMs and Kalman Filters

Applications [\[edit\]](#)

HMMs can be applied in many fields where the goal is to recover a data sequence that is not immediately observable (but other data that depend on the sequence are).

Applications include:

- . Single Molecule Kinetic analysis^[16]
- . Cryptanalysis
- . Speech recognition
- . Speech synthesis
- . Part-of-speech tagging
- . Document Separation in scanning solutions
- . Machine translation
- . Partial discharge
- . Gene prediction
- . Alignment of bio-sequences
- . Time Series Analysis
- . Activity recognition
- . Protein folding^[17]
- . Metamorphic Virus Detection^[18]
- . DNA Motif Discovery^[19]

Applications [\[edit\]](#)

- | | | |
|--|---|--|
| . Attitude and Heading Reference Systems | . Economics, in particular macroeconomics, time series analysis, and econometrics ^[42] | . Simultaneous localization and mapping |
| . Autopilot | . Inertial guidance system | . Speech enhancement |
| . Battery state of charge (SoC) estimation ^{[39][40]} | . Orbit Determination | . Visual odometry |
| . Brain-computer interface | . Power system state estimation | . Weather forecasting |
| . Chaotic signals | . Radar tracker | . Navigation system |
| . Tracking and Vertex Fitting of charged particles in Particle Detectors ^[41] | . Satellite navigation systems | . 3D modeling |
| . Tracking of objects in computer vision | . Seismology ^[43] | . Structural health monitoring |
| . Dynamic positioning | . Sensorless control of AC motor variable-frequency drives | . Human sensorimotor processing ^[44] |

Decoding in Chain-Structured Models

- **Table is too expensive** for Markov chain models:
 - We can't enumerate k^d possible configurations.

Decoding in Chain-Structured Models

- **Table is too expensive** for Markov chain models:
 - We can't enumerate k^d possible configurations.
- But **variables are not independent**:
 - We can't use our nice argument for empty graphs.

Decoding in Chain-Structured Models

- **Table is too expensive** for Markov chain models:
 - We can't enumerate k^d possible configurations.
- But **variables are not independent**:
 - We can't use our nice argument for empty graphs.
- But decoding in chains is not NP-hard:
 - **Conditional independence** structure yields efficient algorithms (Viterbi decoding).

Decoding in Chain-Structured Models

- For Markov chains we have

$$\begin{aligned} p(x^*) &= \max_x p(x_1) \prod_{j=2}^d p(x_j | x_{j-1}) \\ &= \max_{x_d} \max_{x_{d-1}} \dots \max_{x_2} \max_{x_1} p(x_1) \prod_{j=2}^d p(x_j | x_{j-1}), \end{aligned}$$

Decoding in Chain-Structured Models

- For Markov chains we have

$$\begin{aligned}
 p(x^*) &= \max_x p(x_1) \prod_{j=2}^d p(x_j | x_{j-1}) \\
 &= \max_{x_d} \max_{x_{d-1}} \dots \max_{x_2} \max_{x_1} p(x_1) \prod_{j=2}^d p(x_j | x_{j-1}),
 \end{aligned}$$

and again using that $\max_i ab_i = a \max_i b_i$ we get

$$p(x^*) = \max_{x_d} \max_{x_{d-1}} \dots \max_{x_2} \prod_{j=3}^d p(x_j | x_{j-1}) \underbrace{\max_{x_1} p(x_1) p(x_2 | x_1)}_{V(2, x_2)}.$$

- Not as nice as before: inner-most max is not a constant:
 - It depends on x_2 so we can't take it outside sum over x_2 .

Decoding in Chain-Structured Models

- Let's just store the k values of $\{V(2, 1), V(2, 2), \dots, V(2, k)\}$ and keep going,

$$\begin{aligned}
 p(x^*) &= \max_{x_d} \max_{x_{d-1}} \dots \max_{x_3} \max_{x_2} \prod_{j=3}^d p(x_j | x_{j-1}) V(2, x_2) \\
 &= \max_{x_d} \max_{x_{d-1}} \dots \max_{x_3} \prod_{j=4}^d p(x_j | x_{j-1}) \underbrace{\max_{x_2} p(x_3 | x_2) V(2, x_2)}_{V(3, x_3)}.
 \end{aligned}$$

Decoding in Chain-Structured Models

- Let's just store the k values of $\{V(2, 1), V(2, 2), \dots, V(2, k)\}$ and keep going,

$$\begin{aligned}
 p(x^*) &= \max_{x_d} \max_{x_{d-1}} \dots \max_{x_3} \max_{x_2} \prod_{j=3}^d p(x_j | x_{j-1}) V(2, x_2) \\
 &= \max_{x_d} \max_{x_{d-1}} \dots \max_{x_3} \prod_{j=4}^d p(x_j | x_{j-1}) \underbrace{\max_{x_2} p(x_3 | x_2) V(2, x_2)}_{V(3, x_3)}.
 \end{aligned}$$

- Key idea: given k values of $V(2, x_2)$, we can compute all $V(3, x_3)$ in $O(k^2)$.

Decoding in Chain-Structured Models

- Let's just store the k values of $\{V(2, 1), V(2, 2), \dots, V(2, k)\}$ and keep going,

$$\begin{aligned}
 p(x^*) &= \max_{x_d} \max_{x_{d-1}} \dots \max_{x_3} \max_{x_2} \prod_{j=3}^d p(x_j | x_{j-1}) V(2, x_2) \\
 &= \max_{x_d} \max_{x_{d-1}} \dots \max_{x_3} \prod_{j=4}^d p(x_j | x_{j-1}) \underbrace{\max_{x_2} p(x_3 | x_2) V(2, x_2)}_{V(3, x_3)}.
 \end{aligned}$$

- Key idea: given k values of $V(2, x_2)$, we can compute all $V(3, x_3)$ in $O(k^2)$.
- If we keep going

$$\begin{aligned}
 p(x^*) &= \max_{x_d} \max_{x_{d-1}} \dots \max_{x_4} \prod_{j=5}^d p(x_j | x_{j-1}) \underbrace{\max_{x_3} p(x_4 | x_3) V(3, x_3)}_{V(4, x_4)} \\
 &= \max_{x_d} V(d, x_d).
 \end{aligned}$$

Decoding in Chain-Structured Models

- The V functions summarize everything you need to know about the past.
- Given k values of $V(j-1, x_{j-1})$ can compute all k values of $V(j, x_{j+1})$ in $O(k^2)$.

Decoding in Chain-Structured Models

- The V functions summarize everything you need to know about the past.
- Given k values of $V(j-1, x_{j-1})$ can compute all k values of $V(j, x_{j+1})$ in $O(k^2)$.
- Doing this $d-1$ times gives a cost of $O(dk^2)$ to find maximum value.
- If we store the argmax values as we go, get decoding by backtracking.

Decoding in Chain-Structured Models

- The V functions summarize everything you need to know about the past.
- Given k values of $V(j-1, x_{j-1})$ can compute all k values of $V(j, x_{j+1})$ in $O(k^2)$.
- Doing this $d-1$ times gives a cost of $O(dk^2)$ to find maximum value.
- If we store the argmax values as we go, get decoding by backtracking.
- A special case of dynamic programming:
 - 1 Optimal solution is defined through recursive calls,

$$V(j, x_{j+1}) = \max_{x_j} p(x_{j+1}|x_j)V(j, x_j).$$

- 2 Limited number of possible recursive calls:
 - d values of first argument, k values of second.

Decoding in Chain-Structured Models

- The V functions **summarize everything you need to know about the past**.
- Given k values of $V(j-1, x_{j-1})$ can compute all k values of $V(j, x_{j+1})$ in $O(k^2)$.
- Doing this $d-1$ times gives a cost of $O(dk^2)$ to find maximum value.
- If we store the argmax values as we go, get decoding by **backtracking**.
- A special case of **dynamic programming**:
 - 1 Optimal solution is defined through recursive calls,

$$V(j, x_{j+1}) = \max_{x_j} p(x_{j+1}|x_j)V(j, x_j).$$

- 2 Limited number of possible recursive calls:

- d values of first argument, k values of second.

so we can solve the problem by **storing answers to recursive calls**.

Decoding in Chain-Structured Models

- **Viterbi decoding** algorithm for general chain-structured UGMs:
 - Forward phase:

$$V_{1,s} = \phi_1(s), \quad V_{i,s} = \max_{s'} \{ \phi_i(s) \phi_{i,i-1}(s, s') V_{i-1,s'} \},$$

- Backward phase: backtrack through argmax values.
- Solves the decoding problem in $O(dk^2)$ instead of $O(dk^n)$.

Decoding in Chain-Structured Models

- **Viterbi decoding** algorithm for general chain-structured UGMs:
 - Forward phase:

$$V_{1,s} = \phi_1(s), \quad V_{i,s} = \max_{s'} \{ \phi_i(s) \phi_{i,i-1}(s, s') V_{i-1,s'} \},$$

- Backward phase: backtrack through argmax values.
 - Solves the decoding problem in $O(dk^2)$ instead of $O(dk^n)$.
- For the CS grad student Markov model with $n = 60$:
 - Optimal decoding is 'industry' for each year.

Decoding in Chain-Structured Models

- **Viterbi decoding** algorithm for general chain-structured UGMs:

- Forward phase:

$$V_{1,s} = \phi_1(s), \quad V_{i,s} = \max_{s'} \{ \phi_i(s) \phi_{i,i-1}(s, s') V_{i-1,s'} \},$$

- Backward phase: backtrack through argmax values.
- Solves the decoding problem in $O(dk^2)$ instead of $O(dk^n)$.
- For the CS grad student Markov model with $n = 60$:
 - Optimal decoding is 'industry' for each year.
 - Optimal decoding might not look like 'typical' state.
 - Optimal decoding would be different with inhomogeneous chain.
 - Optimal decoding would be different if we changed n .

Inference in Chain-Structured Models

- We can do inference following the same logic:
 - $V_{i,s}$ will **sum** over variable rather than maximize over them.

Inference in Chain-Structured Models

- We can do inference following the same logic:
 - $V_{i,s}$ will **sum** over variable rather than maximize over them.
- **Forward-backward algorithm** for general case:
 - Forward phase (sums up paths from the beginning):

$$V_{1,s} = \phi_1(s), \quad V_{i,s} = \sum_{s'} \phi_i(s) \phi_{i,i-1}(s, s') V_{i-1,s'}, \quad Z = \sum_s V_{n,s}.$$

Inference in Chain-Structured Models

- We can do inference following the same logic:
 - $V_{i,s}$ will **sum** over variable rather than maximize over them.
- **Forward-backward algorithm** for general case:
 - Forward phase (sums up paths from the beginning):

$$V_{1,s} = \phi_1(s), \quad V_{i,s} = \sum_{s'} \phi_i(s) \phi_{i,i-1}(s, s') V_{i-1,s'}, \quad Z = \sum_s V_{n,s}.$$

- Backward phase: (sums up paths to the end):

$$B_{n,s} = 1, \quad B_{i,s} = \sum_{s'} \phi_{i+1}(s') \phi_{i+1,i}(s', s) B_{i+1,s'}.$$

Inference in Chain-Structured Models

- We can do inference following the same logic:
 - $V_{i,s}$ will **sum** over variable rather than maximize over them.
- **Forward-backward algorithm** for general case:
 - Forward phase (sums up paths from the beginning):

$$V_{1,s} = \phi_1(s), \quad V_{i,s} = \sum_{s'} \phi_i(s) \phi_{i,i-1}(s, s') V_{i-1,s'}, \quad Z = \sum_s V_{n,s}.$$

- Backward phase: (sums up paths to the end):

$$B_{n,s} = 1, \quad B_{i,s} = \sum_{s'} \phi_{i+1}(s') \phi_{i+1,i}(s', s) B_{i+1,s'}.$$

- Marginals are given by $p(x_i = s) \propto V_{i,s} B_{i,s}$.

Sampling in Chain-Structured Models

- Sampling in Markov chains by ancestral sampling:
 - Sample time 1 based on $p(x_1)$.
 - Sample time t based on time $t - 1$ using $p(x_t|x_{t-1})$.
 - Simulates the process forward from the beginning.

Sampling in Chain-Structured Models

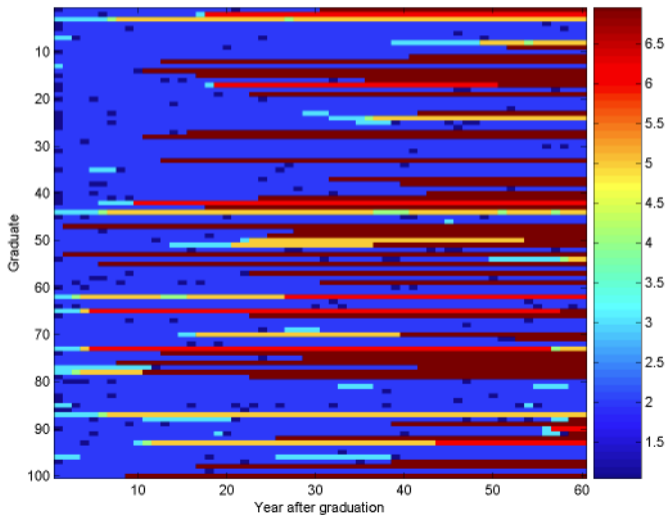
- Sampling in Markov chains by ancestral sampling:
 - Sample time 1 based on $p(x_1)$.
 - Sample time t based on time $t - 1$ using $p(x_t|x_{t-1})$.
 - Simulates the process forward from the beginning.
- **Forward-filter backward-sample algorithm** for general case:
 - Forward phase (same as before):

$$V_{1,s} = \phi_1(s), \quad V_{i,s} = \sum_{s'} \phi_i(s) \phi_{i,i-1}(s, s') V_{i-1,s'}.$$

- Backward phase: sample x_n now that we have $p(x_n)$, then sample time $(t - 1)$ based on $V_{t-1,s}$ and x_t .
- Simulates the process backwards from the end.

Samples in CS Grad Markov Chain

Samples are more informative about what the model looks like:



Outline

- 1 DAGs vs. UGMs
- 2 Empty Graphs
- 3 Chain-Structured Graphs
- 4 General Graphs**

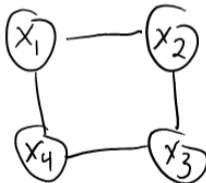
Decoding, Inference, and Sampling in General Graphs

- What if we want to go beyond chains? Can't we apply same logic?
 - Yes, but there is going to be a problem...

Decoding, Inference, and Sampling in General Graphs

- What if we want to go beyond chains? Can't we apply same logic?
 - Yes, but there is going to be a problem...
- Consider a simple 4-node grid-structure UGM:

$$p(x) \propto \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\phi_{34}(x_3, x_4)\phi_{14}(x_1, x_4).$$



Variable Elimination in General Graphs

- We have that Z is defined by

$$\begin{aligned}
 Z &= \sum_{x_4} \sum_{x_3} \sum_{x_2} \sum_{x_1} \phi_{12}(x_1, x_2) \phi_{23}(x_2, x_3) \phi_{34}(x_3, x_4) \phi_{14}(x_1, x_4) \\
 &= \sum_{x_4} \sum_{x_3} \sum_{x_2} \phi_{23}(x_2, x_3) \phi_{34}(x_3, x_4) \underbrace{\sum_{x_1} \phi_{12}(x_1, x_2) \phi_{14}(x_1, x_4)}_{V_{24}(x_2, x_4)},
 \end{aligned}$$

so now x_j our V_{24} function has k^2 values instead of k .

Variable Elimination in General Graphs

- We have that Z is defined by

$$\begin{aligned} Z &= \sum_{x_4} \sum_{x_3} \sum_{x_2} \sum_{x_1} \phi_{12}(x_1, x_2) \phi_{23}(x_2, x_3) \phi_{34}(x_3, x_4) \phi_{14}(x_1, x_4) \\ &= \sum_{x_4} \sum_{x_3} \sum_{x_2} \phi_{23}(x_2, x_3) \phi_{34}(x_3, x_4) \underbrace{\sum_{x_1} \phi_{12}(x_1, x_2) \phi_{14}(x_1, x_4)}_{V_{24}(x_2, x_4)}, \end{aligned}$$

so now x_j our V_{24} function has k^2 values instead of k .

- Continuing, we get

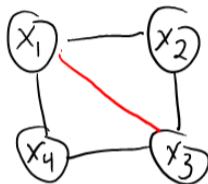
$$Z = \sum_{x_4} \sum_{x_3} \phi_{34}(x_3, x_4) \underbrace{\sum_{x_2} \phi_{23}(x_2, x_3) V_{24}(x_2, x_4)}_{V_{34}(x_3, x_4)},$$

and so on. The total cost will now be $O(dk^3)$.

- This strategy is called **variable elimination**.

Variable Elimination in General Graphs

- If we add the edge (1, 3),



we get

$$\begin{aligned}
 Z &= \sum_{x_4} \sum_{x_3} \sum_{x_2} \sum_{x_1} \phi_{12}(x_1, x_2) \phi_{13}(x_1, x_3) \phi_{23}(x_2, x_3) \phi_{34}(x_3, x_4) \phi_{14}(x_1, x_4) \\
 &= \sum_{x_4} \sum_{x_3} \sum_{x_2} \phi_{23}(x_2, x_3) \phi_{34}(x_3, x_4) \underbrace{\sum_{x_1} \phi_{12}(x_1, x_2) \phi_{13}(x_1, x_3) \phi_{14}(x_1, x_4)}_{V_{234}(x_2, x_3, x_4)},
 \end{aligned}$$

so now we have a V_{234} function with k^3 possible values.

- Same $O(dk^4)$ cost of exhaustive enumeration.

Variable Elimination in General Graphs

- The cost also changes if we change the **order of the sums**.

Variable Elimination in General Graphs

- The cost also changes if we change the **order of the sums**.
- Consider chain-structured graph with sums in a different order:

$$\begin{aligned}
 Z &= \sum_{x_5} \sum_{x_4} \sum_{x_3} \sum_{x_2} \sum_{x_1} \prod_{j=2}^d \phi(x_j, x_{j-1}) \\
 &= \sum_{x_5} \sum_{x_3} \sum_{x_2} \sum_{x_4} \sum_{x_1} \prod_{j=2}^d \phi(x_j, x_{j-1}) \\
 &= \sum_{x_5} \sum_{x_3} \sum_{x_2} \sum_{x_4} \prod_{j=3}^d \phi(x_j, x_{j-1}) \underbrace{\sum_{x_1} \phi(x_2, x_1)}_{V_2(x_2)} \\
 &= \sum_{x_5} \sum_{x_3} \sum_{x_2} \phi(x_3, x_2) \underbrace{\sum_{x_4} \phi(x_4, x_3) \phi(x_5, x_4)}_{V_{235}(x_2, x_3, x_5)} V_2(x_2).
 \end{aligned}$$

- So even though we have a chain, we have a V with k^3 values instead of k .

Variable Elimination and Treewidth

- So cost of variable elimination depends on
 - 1 Graph structure.
 - 2 Variable order.

Variable Elimination and Treewidth

- So cost of variable elimination depends on
 - 1 Graph structure.
 - 2 Variable order.
- Cost of variable elimination for best ordering is given by:
 - $O(dk^{\omega+1})$, where ω is the **treewidth** of the graph.

Variable Elimination and Treewidth

- So cost of variable elimination depends on
 - ① Graph structure.
 - ② Variable order.
- Cost of variable elimination for best ordering is given by:
 - $O(dk^{\omega+1})$, where ω is the **treewidth** of the graph.
- Treewidth ω is minimum over triangulations of size of largest clique.
 - For chains, $\omega = 1$ (many orderings achieve this).

Variable Elimination and Treewidth

- So cost of variable elimination depends on
 - 1 Graph structure.
 - 2 Variable order.
- Cost of variable elimination for best ordering is given by:
 - $O(dk^{\omega+1})$, where ω is the **treewidth** of the graph.
- Treewidth ω is minimum over triangulations of size of largest clique.
 - For chains, $\omega = 1$ (many orderings achieve this).
 - In the worst case, $\omega = (d - 1)$ so there is no gain.

Variable Elimination and Treewidth

- So cost of variable elimination depends on
 - 1 Graph structure.
 - 2 Variable order.
- Cost of variable elimination for best ordering is given by:
 - $O(dk^{\omega+1})$, where ω is the **treewidth** of the graph.
- Treewidth ω is minimum over triangulations of size of largest clique.
 - For chains, $\omega = 1$ (many orderings achieve this).
 - In the worst case, $\omega = (d - 1)$ so there is no gain.
 - Computing ω and optimal ordering is NP-hard.
 - But various heuristic ordering methods exist.

Variable Elimination and Treewidth

- So cost of variable elimination depends on
 - 1 Graph structure.
 - 2 Variable order.
- Cost of variable elimination for best ordering is given by:
 - $O(dk^{\omega+1})$, where ω is the **treewidth** of the graph.
- Treewidth ω is minimum over triangulations of size of largest clique.
 - For chains, $\omega = 1$ (many orderings achieve this).
 - In the worst case, $\omega = (d - 1)$ so there is no gain.
 - Computing ω and optimal ordering is NP-hard.
 - But various heuristic ordering methods exist.
 - For trees, $\omega = 1$.

Variable Elimination and Treewidth

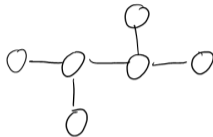
- So cost of variable elimination depends on
 - 1 Graph structure.
 - 2 Variable order.
- Cost of variable elimination for best ordering is given by:
 - $O(dk^{\omega+1})$, where ω is the **treewidth** of the graph.
- Treewidth ω is minimum over triangulations of size of largest clique.
 - For chains, $\omega = 1$ (many orderings achieve this).
 - In the worst case, $\omega = (d - 1)$ so there is no gain.
 - Computing ω and optimal ordering is NP-hard.
 - But various heuristic ordering methods exist.
 - For trees, $\omega = 1$.
 - If just you have a big loop, $\omega = 2$.

Variable Elimination and Treewidth

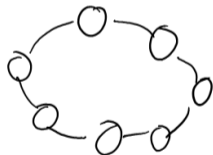
- So cost of variable elimination depends on
 - 1 Graph structure.
 - 2 Variable order.
- Cost of variable elimination for best ordering is given by:
 - $O(dk^{\omega+1})$, where ω is the **treewidth** of the graph.
- Treewidth ω is minimum over triangulations of size of largest clique.
 - For chains, $\omega = 1$ (many orderings achieve this).
 - In the worst case, $\omega = (d - 1)$ so there is no gain.
 - Computing ω and optimal ordering is NP-hard.
 - But various heuristic ordering methods exist.
 - For trees, $\omega = 1$.
 - If just you have a big loop, $\omega = 2$.
 - For a d_1 by d_2 grid, $\omega = \min\{d_1, d_2\}$.

Variable Elimination and Treewidth

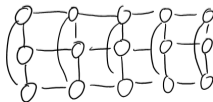
- Trees have $\omega = 1$, decoding/inference/sampling costs $O(dk^2)$.



- A loop has $\omega = 2$, cost is $O(dk^3)$.



- A time-series with 3 variables has $\omega = 3$, cost is $O(dk^4)$.



Summary

- **Markov blanket** is set of nodes that make x_j independent of all others.

Summary

- **Markov blanket** is set of nodes that make x_j independent of all others.
- **Moralization of DAGs** to do decoding/inference/sampling as a UGM.

Summary

- **Markov blanket** is set of nodes that make x_j independent of all others.
- **Moralization of DAGs** to do decoding/inference/sampling as a UGM.
- **Decoding/inference/sampling** with different graph structures:
 - Factorizing sum for independent distributions.
 - **Viterbi decoding** and **forward-backward** for chains.
 - **Variable elimination** for general graphs.

Summary

- **Markov blanket** is set of nodes that make x_j independent of all others.
- **Moralization of DAGs** to do decoding/inference/sampling as a UGM.
- **Decoding/inference/sampling** with different graph structures:
 - Factorizing sum for independent distributions.
 - **Viterbi decoding** and **forward-backward** for chains.
 - **Variable elimination** for general graphs.
- I will be gone for the next 3 lectures:
 - Michael Gelbart will introduce deep learning and Bayesian stats.
 - Then we'll have the midterm.
 - Then I'll cover advanced topics in graphical models, deep learning, and Bayesian stats.