10-607 Computational Foundations for Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

APPLICATION: Variable Elimination

Matt Gormley
Lecture 11
Nov. 28, 2018

Reminders

- Homework C: Data Structures
  - Out: Mon, Nov. 26
  - Due: Mon, Dec. 3 at 11:59pm
- Quiz B: Computation; Programming & Efficiency
  - Wed, Dec. 5, in-class
  - Covers Lectures 7 – 12

APPLICATION: EXACT INFERENCE IN GRAPHICAL MODELS

EXACT INFERENCE

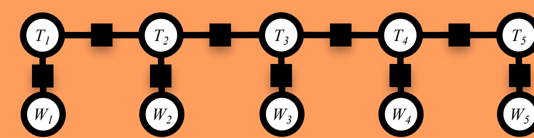
Exact Inference

1. Data

$$\mathcal{D} = \{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$$

2. Model

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$



3. Objective

$$\ell(\theta; \mathcal{D}) = \sum_{n=1}^{N} \log p(\boldsymbol{x}^{(n)} \mid \boldsymbol{\theta})$$

5. Inference

1. Marginal Inference

$$p(\boldsymbol{x}_C) = \sum_{\boldsymbol{x}' : \boldsymbol{x}'_C = \boldsymbol{x}_C} p(\boldsymbol{x}' \mid \boldsymbol{\theta})$$

2. Partition Function

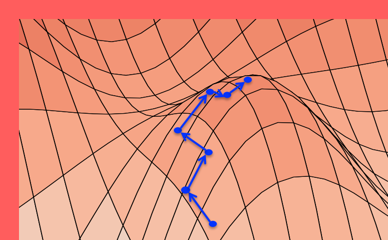
$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$

3. MAP Inference

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmax}} \; p(\boldsymbol{x} \mid \boldsymbol{\theta})$$

4. Learning

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \, \ell(\boldsymbol{\theta}; \mathcal{D})$$



Complexity Classes

- An algorithm runs in **polynomial time** if its runtime is a polynomial function of the input size (e.g. $O(n^k)$ for some fixed constant k)
- The **class P** consists of all problems that can be solved in polynomial time

- A problem for which the answer is binary (e.g. yes/no) is called a **decision problem**
- The **class NP** contains all decision problems where ‘yes’ answers can be verified (proved) in polynomial time
- A problem is **NP-Hard** if given an O(1) oracle to solve it, every problem in NP can be solved in polynomial time (e.g. by reduction)
- A problem is **NP-Complete** if it belongs to both the classes NP and NP-Hard

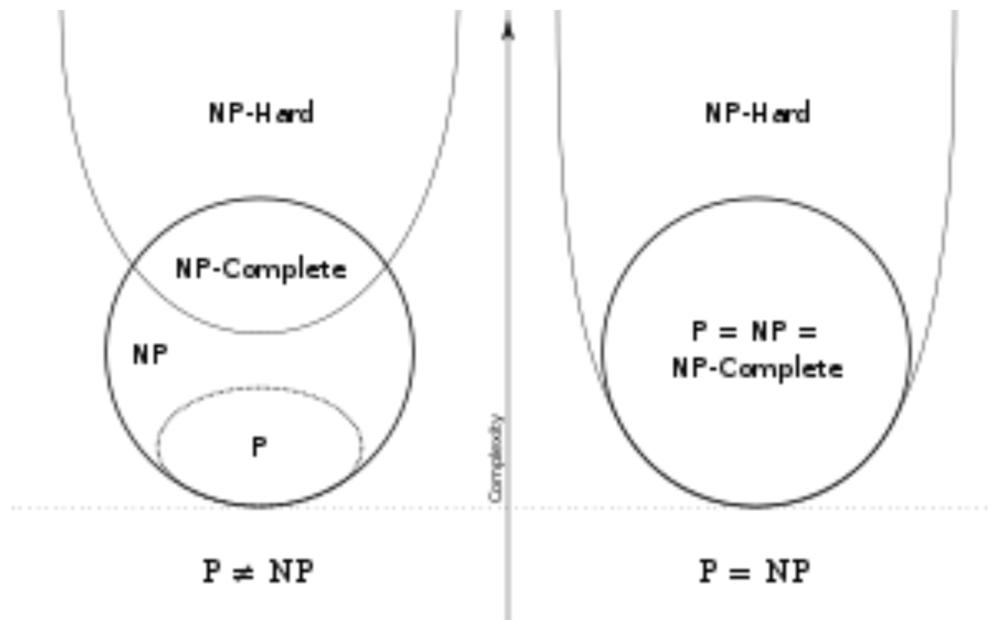


Figure from https://en.wikipedia.org/wiki/NP-completeness

5. Inference

Three Tasks:

1. Marginal Inference **(#P-Hard)**
Compute marginals of variables and cliques

$$p(x_i) = \sum_{\boldsymbol{x}' : x'_i = x_i} p(\boldsymbol{x}' \mid \boldsymbol{\theta}) \qquad \bigg| \qquad p(\boldsymbol{x}_C) = \sum_{\boldsymbol{x}' : \boldsymbol{x}'_C = \boldsymbol{x}_C} p(\boldsymbol{x}' \mid \boldsymbol{\theta})$$

2. Partition Function **(#P-Hard)**
Compute the normalization constant

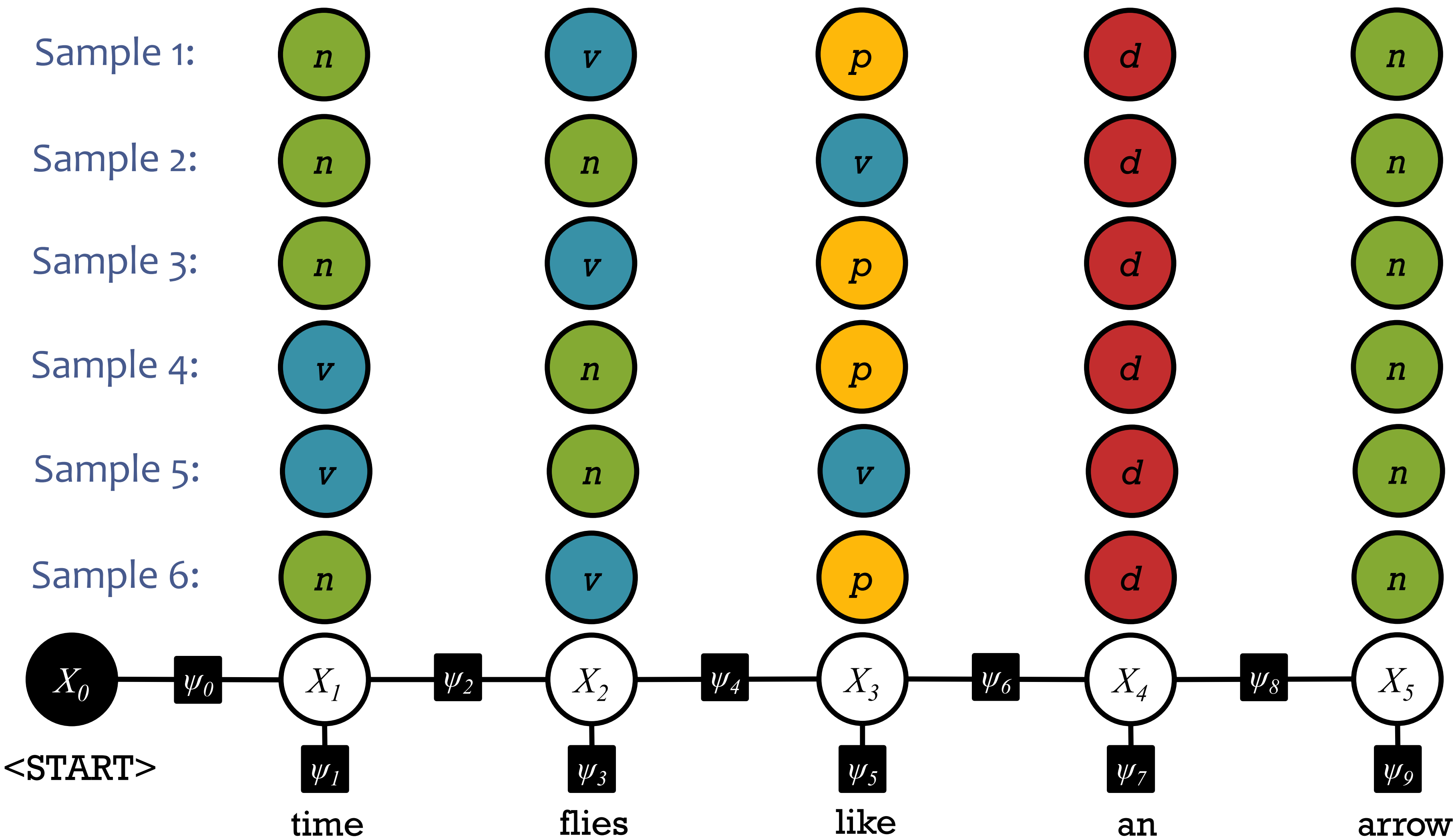
$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$

3. MAP Inference **(NP-Hard)**
Compute variable assignment with highest probability

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmax}} \; p(\boldsymbol{x} \mid \boldsymbol{\theta})$$

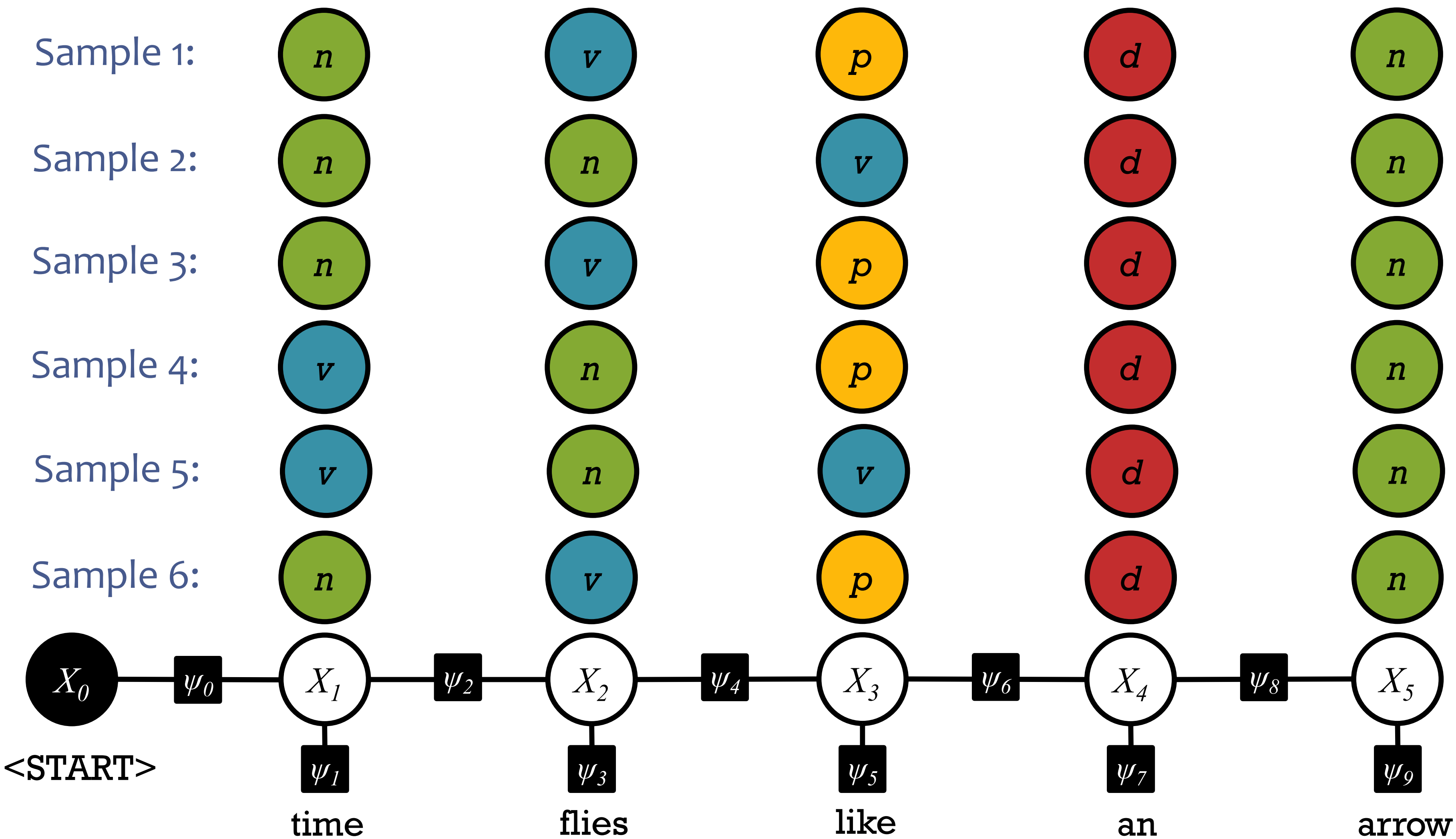
Marginals by Sampling on Factor Graph

Suppose we took many samples from the distribution over taggings: $p(\boldsymbol{x}) = \frac{1}{Z} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})$

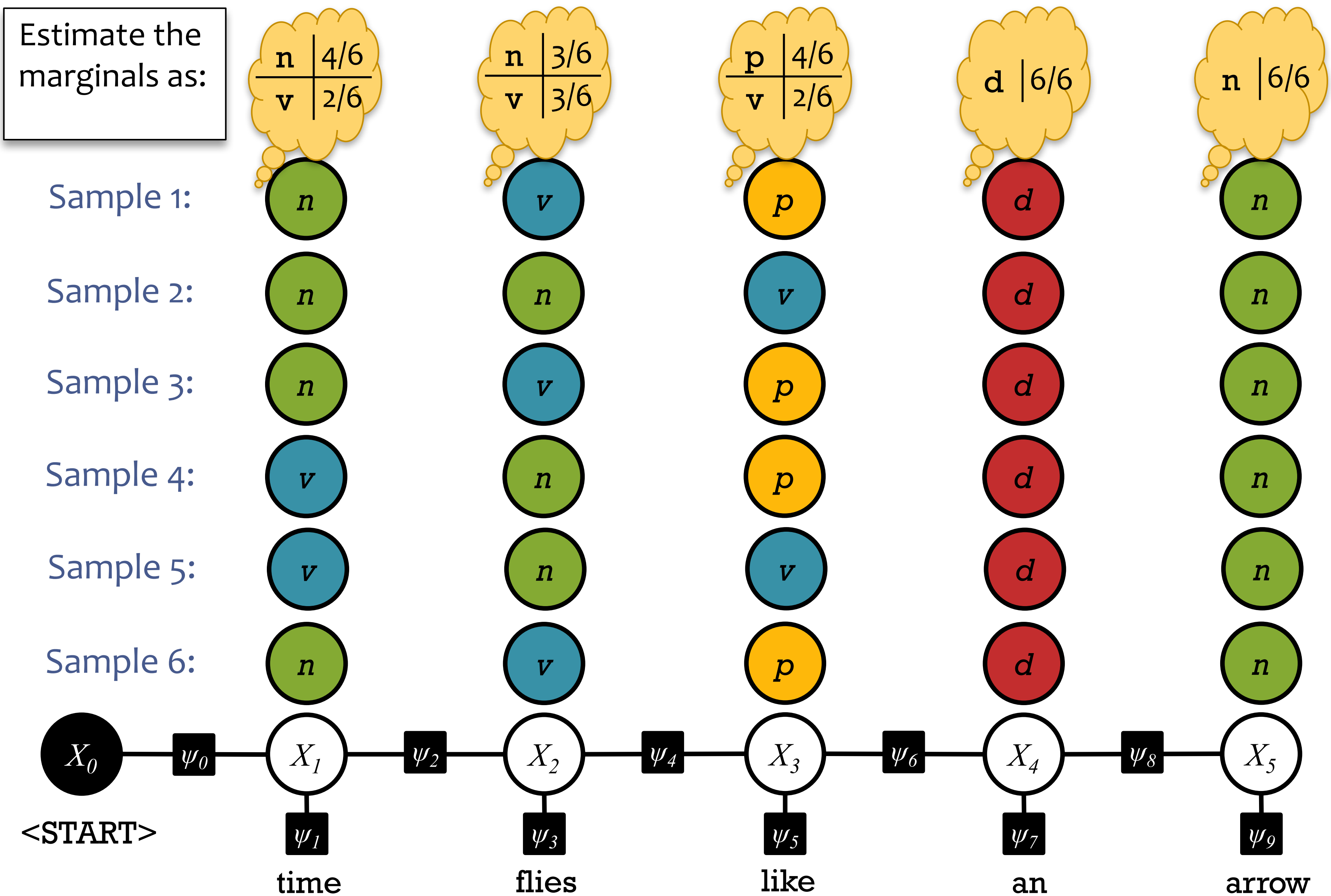


Marginals by Sampling on Factor Graph

The marginal $p(X_i = x_i)$ gives the probability that variable X_i takes value x_i in a random sample



Marginals by Sampling on Factor Graph



Simple and general exact inference for graphical models

VARIABLE ELIMINATION

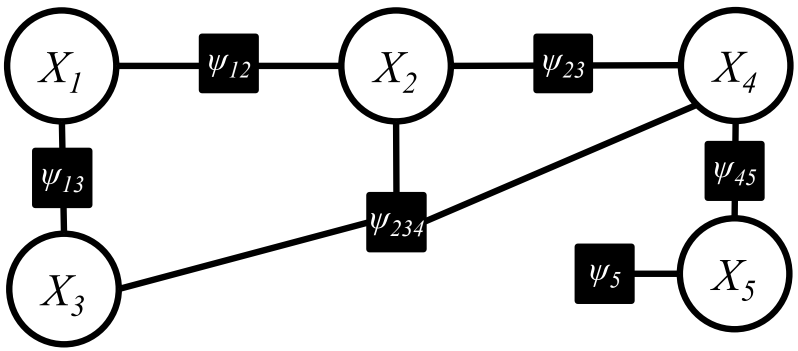
Brute Force (Naïve) Inference

For all i, suppose the **range** of X_i is $\{0, 1, 2\}$.

Let $k=3$ denote the **size of the range.**

The distribution **factorizes** as:

$$p(\boldsymbol{x}) = \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)$$
$$\psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5)$$



Naively, we compute the **partition function** as:

$$Z = \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} p(\boldsymbol{x})$$

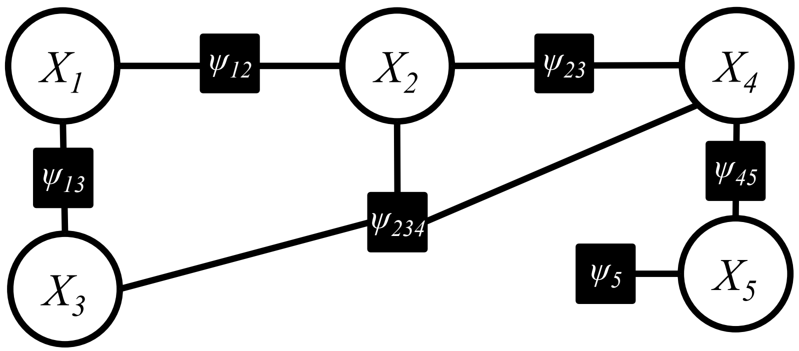
Brute Force (Naïve) Inference

For all i, suppose the **range** of X_i is $\{0, 1, 2\}$.

Let $k=3$ denote the **size of the range**.

The distribution **factorizes** as:

$$p(\boldsymbol{x}) = \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5)$$



Naively, we compute the **partition function** as:

$$Z = \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} p(\boldsymbol{x})$$

$p(\mathbf{x})$ can be represented as a joint probability table with 3^5 entries:

x_1	x_2	x_3	x_4	x_5	$p(\mathbf{x})$
0	0	0	0	0	0.019517693
0	0	0	0	1	0.017090249
0	0	0	0	2	0.014885825
0	0	0	1	0	0.024117638
0	0	0	1	1	0.000925849
0	0	0	1	2	0.028112576
0	0	0	2	0	0.028050205
0	0	0	2	1	0.004812689
0	0	0	2	2	0.007987737
0	0	1	0	0	0.028433687
0	0	1	0	1	0.037073469
0	0	1	0	2	0.013558227
0	0	1	1	0	0.019479016
0	0	1	1	1	0.012312901
0	0	1	1	2	0.023439775
0	0	1	2	0	0.038206131
0	0	1	2	1	0.038996005
0	0	1	2	2	0.041458783
0	0	2	0	0	0.044616806
0	0	2	0	1	0.020846989
0	0	2	0	2	0.03006475
0	0	2	1	0	0.048436964
0	0	2	1	1	0.02854376
0	0	2	1	2	0.029191506
0	0	2	2	0	0.031531118
0	0	2	2	1	0.005132392
0	0	2	2	2	0.032027091
…	…	…	…	…	…

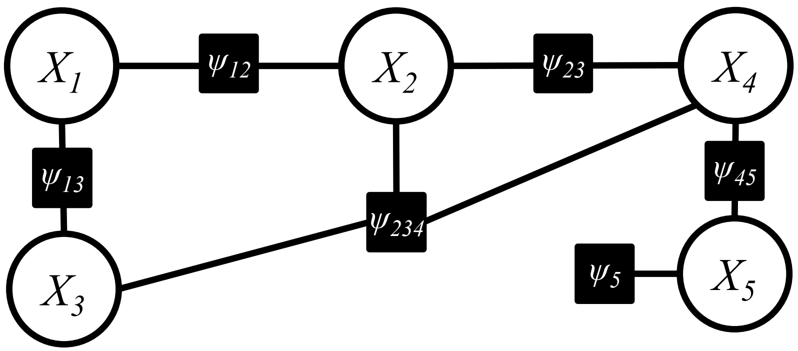
Brute Force (Naïve) Inference

For all i, suppose the **range** of X_i is $\{0, 1, 2\}$.

Let $k=3$ denote the **size of the range.**

The distribution **factorizes** as:

$$p(\boldsymbol{x}) = \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)$$
$$\psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5)$$



Naively, we compute the **partition function** as:

$$Z = \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} p(\boldsymbol{x})$$

$p(\mathbf{x})$ can be represented as a joint probability table with 3^5 entries:

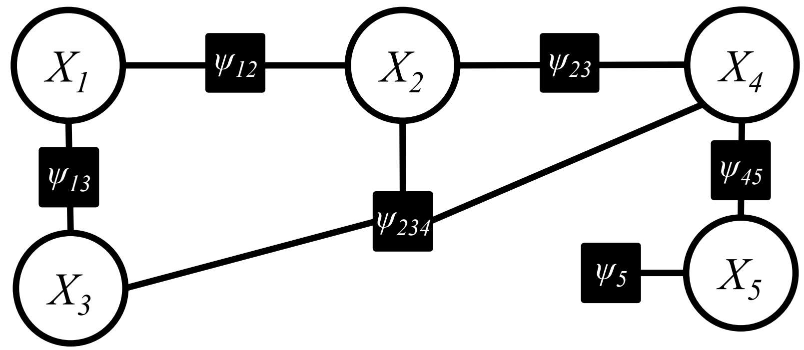
x_1	x_2	x_3	x_4	x_5	$p(\mathbf{x})$
0	0	0	0	0	0.019517693
0	0	0	0	1	0.017090249
0	0	0	0	2	0.014885825
0	0	0	1	0	0.024117638
0	0	0	1	1	0.000925849
0	0	0	1	2	0.028112576
0	0	0	2	0	0.028050205
0	0	0	2	1	0.004812689
0	0	0	2	2	0.007987737
0	0	1	0	0	0.028433687
0	0	1	0	1	0.037073469
0	0	1	0	2	0.013558227
0	0	1	1	0	0.019479016
0	0	1	1	1	0.012312901
0	0	1	1	2	0.023439775
0	0	1	2	0	0.038206131
0	0	1	2	1	0.038996005
0	0	1	2	2	0.041458783
0	0	2	0	0	0.044616806
0	0	2	0	1	0.020846989
0	0	2	0	2	0.03006475
0	0	2	1	0	0.048436964
0	0	2	1	1	0.02854376

Naïve computation of Z requires 3^5 additions.

Can we do better?

The Variable Elimination Algorithm

Instead, capitalize on the factorization of *p(**x**)*.



$$Z = \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} \psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\psi_{45}(x_4,x_5)\psi_5(x_5)$$

$$= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4} \psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4) \sum_{x_5} \psi_{45}(x_4,x_5)\psi_5(x_5)$$

Only 3^2 additions are needed to marginalize out x_5.

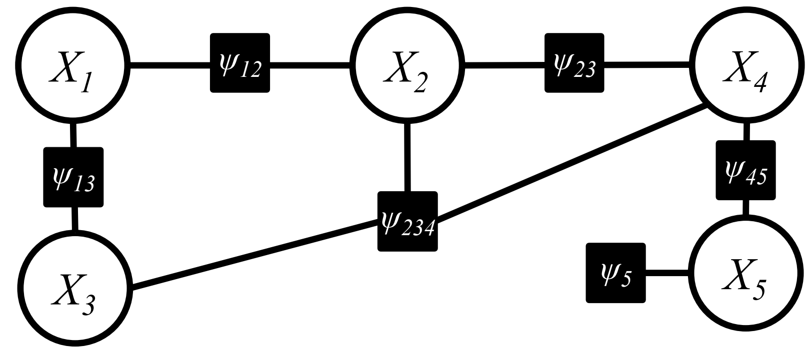
We denote the **marginal's table** by $m_5(x_4)$.

This "factor" is a much smaller table with 3^2 entries:

x_4	x_5	$p(\mathbf{x})$
0	0	0.019517693
0	1	0.017090249
0	2	0.014885825
1	0	0.024117638
1	1	0.000925849
1	2	0.028112576
2	0	0.028050205
2	1	0.004812689
2	2	0.007987737

The Variable Elimination Algorithm

Instead, capitalize on the factorization of *p*(***x***).

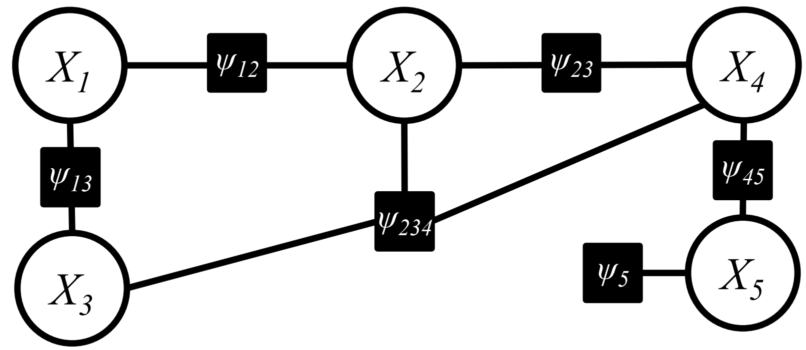


$$
\begin{aligned}
Z &= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5}\psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\psi_{45}(x_4,x_5)\psi_5(x_5)\\
&= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4}\psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\sum_{x_5}\psi_{45}(x_4,x_5)\psi_5(x_5)\\
&= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4}\psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)m_5(x_4)
\end{aligned}
$$

$$
m_5(x_4) \triangleq \sum_{x_5}\psi_{45}(x_4,x_5)\psi_5(x_5)
$$

The Variable Elimination Algorithm

Instead, capitalize on the factorization of *p*(***x***).



$$Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \psi_{45}(x_4, x_5) \psi_5(x_5)$$

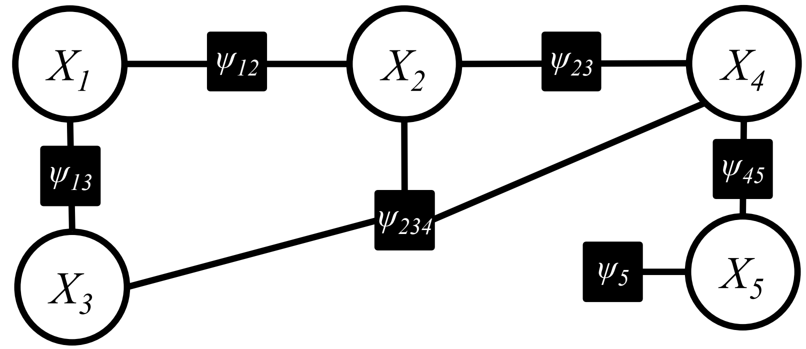
$$= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5)$$

$$= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) m_5(x_4)$$

This "factor" is still a 3^4 table so apply the same trick again.

$$m_5(x_4) \triangleq \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5)$$

The Variable Elimination Algorithm



Instead, capitalize on the factorization of $p(\mathbf{x})$.

$$Z = \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\sum_{x_5}\psi_{45}(x_4,x_5)\psi_5(x_5)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)m_5(x_4)$$

3^2 additions

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)m_4(x_2,x_3)$$

3^3 additions

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)m_3(x_1,x_2)$$

3^3 additions

$$= \sum_{x_1} m_2(x_1)$$

3^2 additions

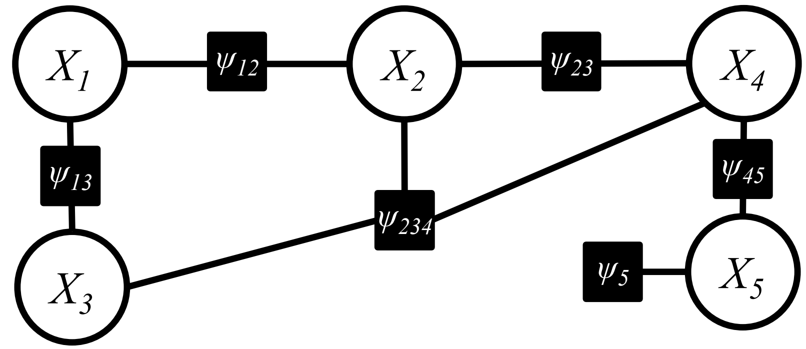
3 additions

Naïve solution requires $3^5=243$ additions.

Variable elimination only requires $3+3^2+3^3+3^3+3^2 = 75$ additions.

The Variable Elimination Algorithm

The same trick can be used to compute **marginal probabilities.** Just choose the variable elimination order such that the query variables are last.



$$p(x_1) = \frac{1}{Z} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4)\psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5)\psi_5(x_5)$$

$$= \frac{1}{Z} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4)\psi_{234}(x_2, x_3, x_4) m_5(x_4)$$

$$= \frac{1}{Z} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) m_4(x_2, x_3)$$

$$= \frac{1}{Z} \sum_{x_2} \psi_{12}(x_1, x_2) m_3(x_1, x_2)$$

$$= \frac{1}{Z} m_2(x_1)$$

3^2 additions

3^3 additions

3^3 additions

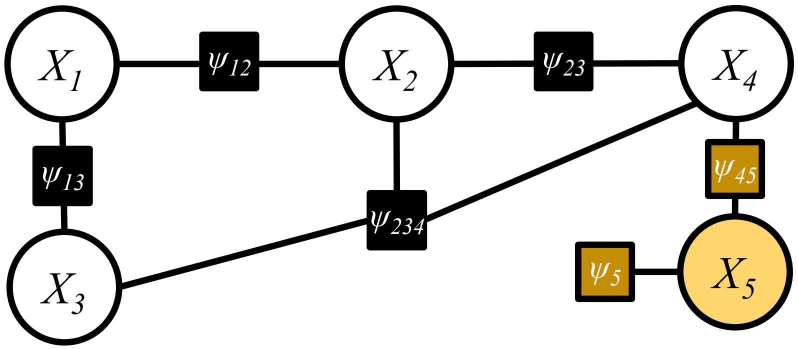
3^2 additions

3 different values on LHS

For directed graphs, Z = 1.

For undirected graphs, if we compute each (unnormalized) value on the LHS, we can sum them to get Z.

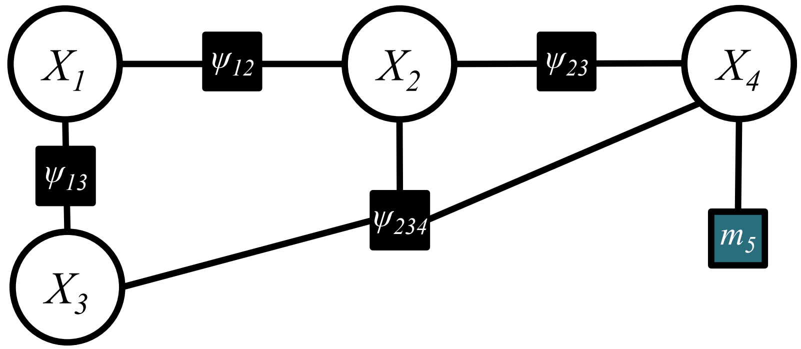
The Variable Elimination Algorithm



$$
\begin{aligned}
Z &= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5) \\
&= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) m_5(x_4) \\
&= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) m_4(x_2, x_3)
\end{aligned}
$$

In a factor graph, variable **elimination** corresponds to replacement of a subgraph with a factor.

The Variable Elimination Algorithm



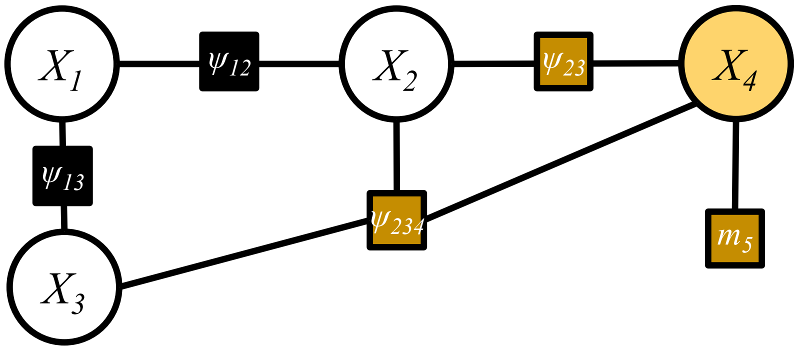
$$Z = \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\sum_{x_5}\psi_{45}(x_4,x_5)\psi_5(x_5)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)m_5(x_4)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)m_4(x_2,x_3)$$

In a factor graph, variable **elimination** corresponds to replacement of a subgraph with a factor.

The Variable Elimination Algorithm



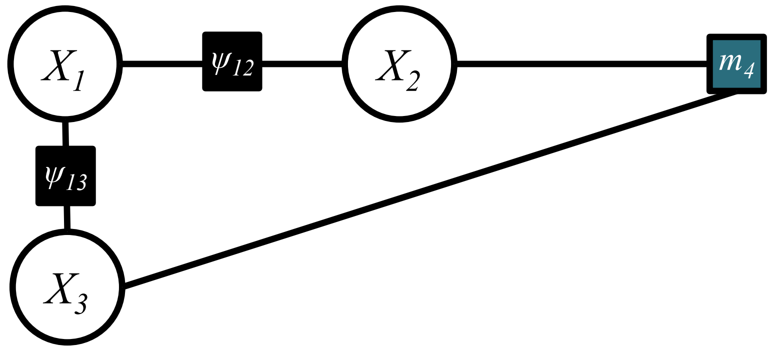
$$Z = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5)$$

$$= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) m_5(x_4)$$

$$= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) m_4(x_2, x_3)$$

In a factor graph, variable **elimination** corresponds to replacement of a subgraph with a factor.

The Variable Elimination Algorithm



$$Z = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5)$$

$$= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) m_5(x_4)$$

$$= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) m_4(x_2, x_3)$$

In a factor graph, variable **elimination** corresponds to replacement of a subgraph with a factor.

Variable Elimination for Marginal Inference

Algorithm 1: Variable Elimination for Marginal Inference

Input: the factor graph and the query variable

Output: the marginal distribution for the query variable

a. Run a breadth-first-search starting at the query variable to obtain an ordering of the variable nodes
b. Reverse that ordering
c. Eliminate each variable in the reversed ordering using Algorithm 2

Algorithm 2: Eliminate One Variable

Input: the variable to be eliminated

Output: new factor graph with the variable marginalized out

a. Find the input variable and its neighboring factors -- call this set the eliminated set
b. Replace the eliminated set with a new factor
    a. The neighbors of the new factor should be all the neighbors of all the factors in the eliminated set
    b. The new factor should assign a score to each possible assignment of its neighboring variables
    c. Said score should be identical to the product of the factors it is replacing, summing over the eliminated variable

Variable Elimination for Marginal Inference

Algorithm 3: Variable Elimination for the Partition Function

Input: the factor graph

Output: the partition function

a. Run a breadth-first-search starting at an arbitrary variable to obtain an ordering of the variable nodes
b. Eliminate each variable in the ordering using Algorithm 2

Algorithm 2: Eliminate One Variable

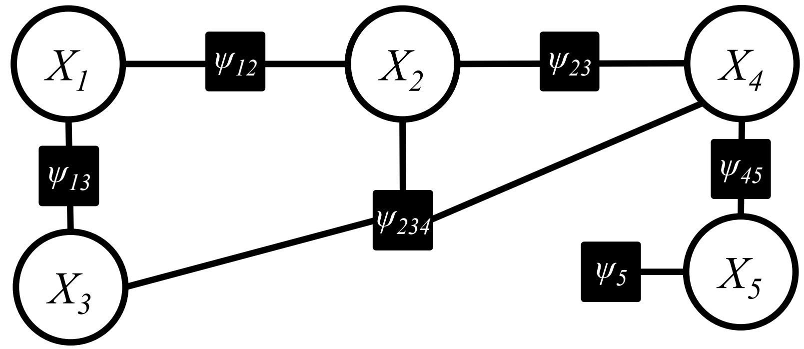
Input: the variable to be eliminated

Output: new factor graph with the variable marginalized out

a. Find the input variable and its neighboring factors -- call this set the eliminated set
b. Replace the eliminated set with a new factor
    a. The neighbors of the new factor should be all the neighbors of all the factors in the eliminated set
    b. The new factor should assign a score to each possible assignment of its neighboring variables
    c. Said score should be identical to the product of the factors it is replacing, summing over the eliminated variable

Variable Elimination Complexity

Instead, capitalize on the factorization of $p(\mathbf{x})$.



In-Class Exercise: *Fill in the blank*

Brute force, naïve, inference is O(____)

Variable elimination is O(____)

where n = # of variables
k = max # values a variable can take
r = # variables participating in largest "intermediate" table

PROFILING FOR EFFICIENCY

Software Profiling

CPU Profiler:

- Intermediate Goal: Analyze the CPU usage of a program at a fine-grained level
  (e.g. time spent within each function)
- End Goal: To make the program more CPU efficient by optimizing most time consuming parts of program

Memory Profiler:

- Intermediate Goal: Analyze the memory consumption of a program
  (e.g. how much space does a particular type of object use on the heap)
- End Goal: To make the program more memory by utilizing different data structures or data storage techniques to reduce memory load

Software Profiling

Deterministic CPU Profiler

- Augments the code with **additional bookkeeping** calls
- Provides **exact number** of times each function is called, and **exact amount** of time spent in each function
- Comes at the cost of much **slower runtime**

Statistical CPU Profiler

- Leaves the code nearly unchanged, and instead **takes samples** (hundreds or more) of the stacktrace
- Provides the **proportion of samples** that landed in each function and **estimates** the total time spent in each function
- Typically yields **little to no slowdown** of the code

Line Profiler

- Same as above for each type, but counts the **number of times each line is executed** and provides the amount of **time spent on each line**
- Increases complexity of the profiler, but provides **much more detailed analysis**

Python Profilers

Name	Type	Level of Detail	Output	Notes
cProfile	deterministic	function-level	console	built into Python standard library; C-based implementation
profile	deterministic	function-level	console	same as cProfile, but implemented in pure Python
line_profiler	deterministic + statistical	line-level	console	C-based implementation
pprofile	deterministic + statistical	line-level	console	pure Python implementation (few users)
PyFlame by Uber	deterministic + statistical	line-level	flame graph	Linux only as of 2018
Plop by Dropbox	deterministic + statistical	line-level	bubble plot	(few users)

cProfile Output

```
$ python -m cProfile -s cumtime lwn2pocket.py
         72270 function calls (70640 primitive calls) in 4.481 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.004    0.004    4.481    4.481 lwn2pocket.py:2(<module>)
        1    0.001    0.001    4.296    4.296 lwn2pocket.py:51(main)
        3    0.000    0.000    4.286    1.429 api.py:17(request)
        3    0.000    0.000    4.268    1.423 sessions.py:386(request)
      4/3    0.000    0.000    3.816    1.272 sessions.py:539(send)
        4    0.000    0.000    2.965    0.741 adapters.py:323(send)
        4    0.000    0.000    2.962    0.740 connectionpool.py:421(urlopen)
        4    0.000    0.000    2.961    0.740 connectionpool.py:317(_make_request)
        2    0.000    0.000    2.675    1.338 api.py:98(post)
       30    0.000    0.000    1.621    0.054 ssl.py:727(recv)
       30    0.000    0.000    1.621    0.054 ssl.py:610(read)
       30    1.621    0.054    1.621    0.054 {method 'read' of '_ssl._SSLSocket' objects}
        1    0.000    0.000    1.611    1.611 api.py:58(get)
        4    0.000    0.000    1.572    0.393 httplib.py:1095(getresponse)
        4    0.000    0.000    1.572    0.393 httplib.py:446(begin)
       60    0.000    0.000    1.571    0.026 socket.py:410(readline)
        4    0.000    0.000    1.571    0.393 httplib.py:407(_read_status)
        1    0.000    0.000    1.462    1.462 pocket.py:44(wrapped)
        1    0.000    0.000    1.462    1.462 pocket.py:152(make_request)
        1    0.000    0.000    1.462    1.462 pocket.py:139(_make_request)
        1    0.000    0.000    1.459    1.459 pocket.py:134(_post_request)
[…]
```

Figure from https://julien.danjou.info/guide-to-python-profiling-cprofile-concrete-case-carbonara/

line_profiler Output

```
Pystone(1.1) time for 50000 passes = 2.48
This machine benchmarks at 20161.3 pystones/second
Wrote profile results to pystone.py.lprof
Timer unit: 1e-06 s

File: pystone.py
Function: Proc2 at line 149
Total time: 0.606656 s

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   149                                           @profile
   150                                           def Proc2(IntParIO):
   151     50000        82003      1.6     13.5      IntLoc = IntParIO + 10
   152     50000        63162      1.3     10.4      while 1:
   153     50000        69065      1.4     11.4          if Char1Glob == 'A':
   154     50000        66354      1.3     10.9              IntLoc = IntLoc - 1
   155     50000        67263      1.3     11.1              IntParIO = IntLoc - IntGlob
   156     50000        65494      1.3     10.8              EnumLoc = Ident1
   157     50000        68001      1.4     11.2          if EnumLoc == Ident1:
   158     50000        63739      1.3     10.5              break
   159     50000        61575      1.2     10.1      return IntParIO
```

Figure from https://github.com/rkern/line_profiler

PyFlame Output

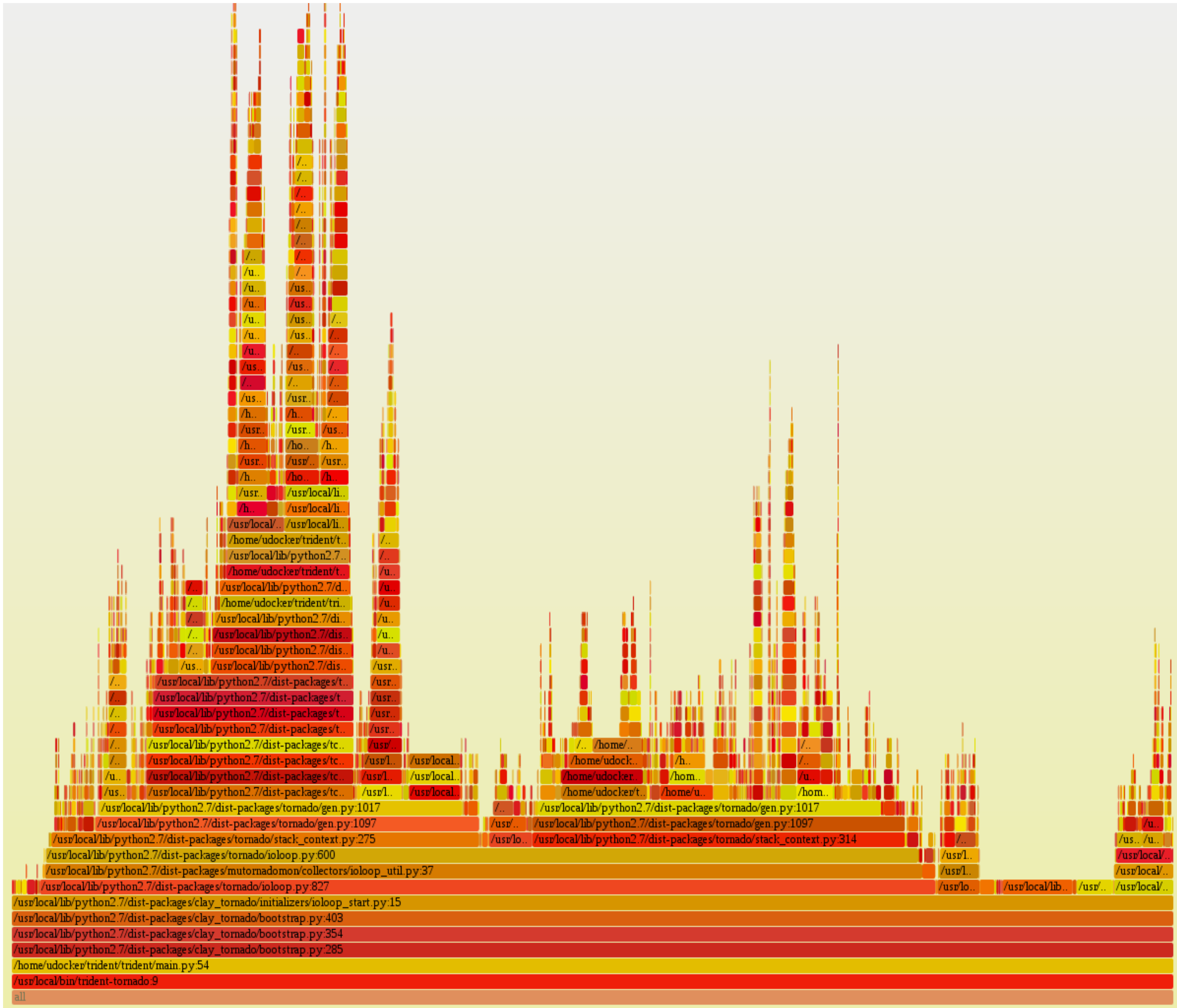


Figure from https://github.com/uber/pyflame

Plop Output

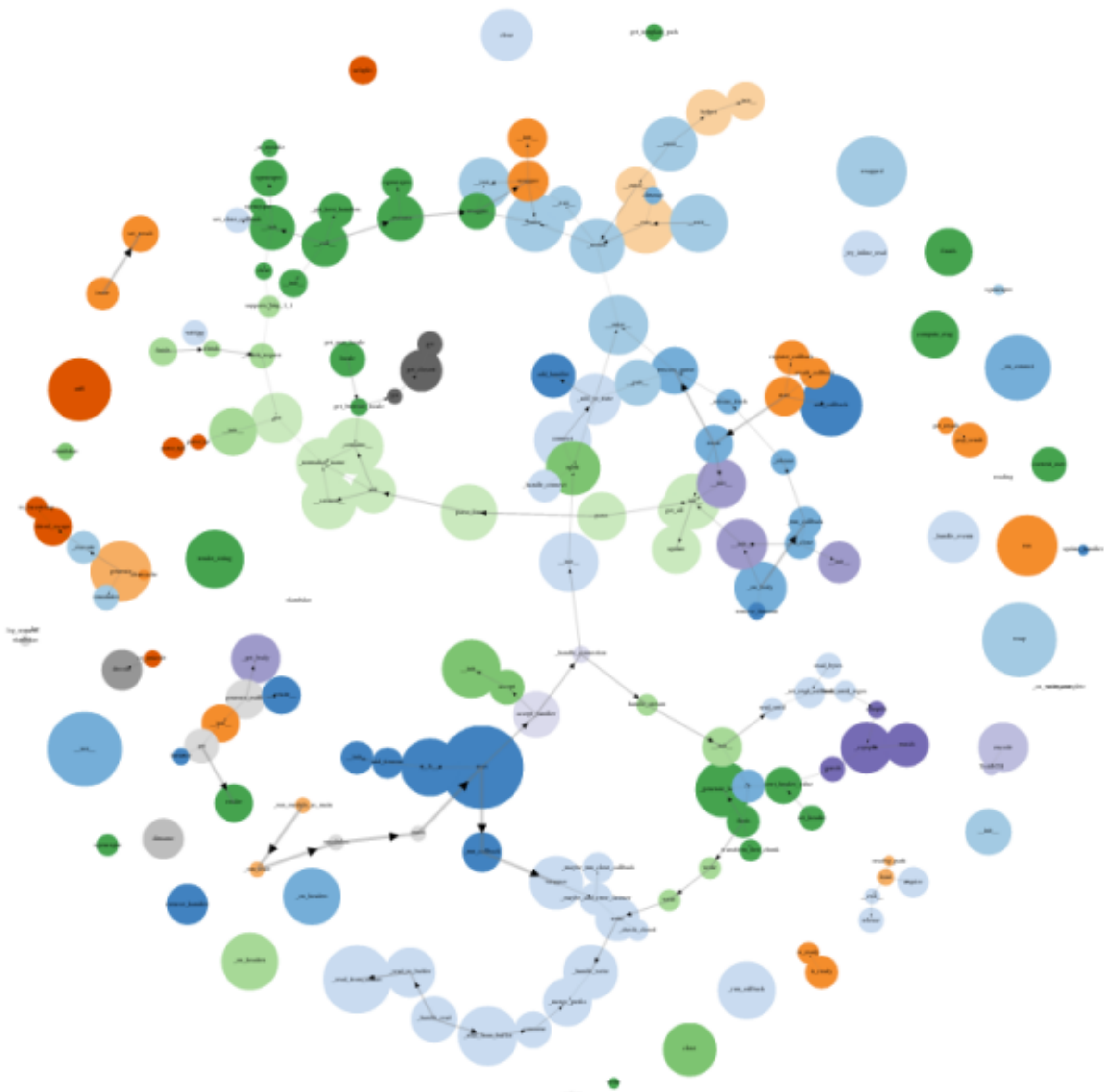


Figure from https://blogs.dropbox.com/tech/2012/07/plop-low-overhead-profiling-for-python/