CS 263: Counting and Sampling

Nima Anari

slides for

Introduction
Logistics

Course staff:

Nima Anari (Instructor)

Victor Lecomte (Course Assistant)
Logistics

Course staff:

Nima Anari
(Instructor)

Victor Lecomte
(Course Assistant)

You:

~39 undergrad + masters + Ph.D.
Logistics

Course staff:

Nima Anari  
(Instructor)

Victor Lecomte  
(Course Assistant)

You:

~39 undergrad + masters + Ph.D.

https://cs263.stanford.edu
Logistics

Course staff:

Nima Anari
(Instructor)

Victor Lecomte
(Course Assistant)

https://cs263.stanford.edu

Lectures: Monday, Wednesday 3:00 pm - 4:20 pm (Hewlett 102)
- Recorded and on Canvas
- Plans to make edited recordings public later ...

You:

~39 undergrad + masters + Ph.D.
Logistics

Course staff:

Nima Anari  
(Instructor)

Victor Lecomte  
(Course Assistant)

You:

~39 undergrad + masters + Ph.D.

https://cs263.stanford.edu

Lectures: Monday, Wednesday 3:00 pm - 4:20 pm (Hewlett 102)
- Recorded and on Canvas
- Plans to make edited recordings public later ...

Homework: 4 sets (20% each)
Logistics

Course staff:

Nima Anari  (Instructor)
Victor Lecomte  (Course Assistant)

You:

~39 undergrad + masters + Ph.D.

https://cs263.stanford.edu

Lectures: Monday, Wednesday 3:00 pm - 4:20 pm (Hewlett 102)
- Recorded and on Canvas
- Plans to make edited recordings public later …

Homework: 4 sets (20% each)

Final report: 20% of grade

- Groups of 1 or 2
- Survey (of ≥ 3 papers) or research (new progress) on topics related to the course
Logistics

Course staff:

Nima Anari (Instructor)
Victor Lecomte (Course Assistant)

You:

~39 undergrad + masters + Ph.D.

https://cs263.stanford.edu

Lectures: Monday, Wednesday 3:00 pm - 4:20 pm (Hewlett 102)
  - Recorded and on Canvas
  - Plans to make edited recordings public later ...

Homework: 4 sets (20% each)

Final report: 20% of grade
  - Groups of 1 or 2
  - Survey (of $\geq 3$ papers) or research (new progress) on topics related to the course

Office hours: Starting next week
What is “Counting and Sampling”?

Bit of Complexity Theory
- The class \#P
- Parsimonious reductions

Approximation
- Counting: FPTAS/FPRAS
- Sampling: FPAUS
- Equivalence

First Algorithm: DNFs
What is “Counting and Sampling”? 

Bit of Complexity Theory 
- The class #P 
- Parsimonious reductions

Approximation 
- Counting: FPTAS/FPRAS 
- Sampling: FPAUS 
- Equivalence

First Algorithm: DNFs
Distribution $\mu$ on large $\Omega$ is usually finite but exp. large

Sampling: efficiently producing sample $\omega \sim \mu$.

Counting: efficiently computing $P_{\mu}[\text{event}]$ for events of interest.

Example: #SAT $\phi = (x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_2 \lor x_4) \land \cdots$

$\Omega$ is $\{0, 1\}^n$.

$\mu$ is uniform over satisfying assignments.

Why is it called counting? Because $P[\text{$x_1 = 1$}] = \#\text{sat assignments of } \phi$ with $x_1 = 1 = \#\text{sat assignments of } \phi \land x_1$.

The numerator and denominator are counts. In fact, numerator is $\#\text{sat assignments to } \phi'$.

This is called "self-reducibility" which will come back to this later.
Distribution $\mu$ on large $\Omega$

- **Sampling**: efficiently producing sample $\omega \sim \mu$. 

usually finite but exp. large

Counting: efficiently computing $P[\mu[\text{event}]]$ for events of interest.

Example: $\#\text{SAT} \phi = (x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_2 \lor x_4) \land \cdots$

$\Omega$ is $\{0, 1\}^n$.

$\mu$ is uniform over satisfying assignments.

Why is it called counting? Because $P[\mu[\text{x}_1 = 1]] = \#\text{sat assignments of } \phi$ with $\text{x}_1 = 1$.

The numerator and denominator are counts. In fact, numerator is $\#\text{sat assignments to } \phi' = \phi \land \text{x}_1$.

This is called "self-reducibility" will come back to this later.
Distribution $\mu$ on large $\Omega$

- **Sampling**: efficiently producing sample $\omega \sim \mu$.
- **Counting**: efficiently computing $P_\mu[\text{event}]$ for events of interest.

- Usually finite but exp. large
Distribution $\mu$ on large $\Omega$

- **Sampling**: efficiently producing sample $\omega \sim \mu$.
- **Counting**: efficiently computing $\mathbb{P}_\mu[\text{event}]$ for events of interest.

**Example: #SAT**

$$\phi = (x_1 \lor \overline{x}_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_4) \land \cdots$$

- $\Omega$ is $\{0, 1\}^n$.
- $\mu$ is uniform over *satisfying* assignments.
Distribution $\mu$ on large $\Omega$

- **Sampling**: efficiently producing sample $\omega \sim \mu$.
- **Counting**: efficiently computing $P_{\mu}[\text{event}]$ for events of interest.

**Example: #SAT**

$\phi = (x_1 \lor \overline{x}_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_4) \land \cdots$

- $\Omega$ is $\{0, 1\}^n$.
- $\mu$ is uniform over satisfying assignments.

Why is it called counting? Because $P_{\mu}[x_1 = 1] = \#\text{sat assignments of } \phi$ with $x_1 = 1$. The numerator and denominator are counts. In fact, numerator is $\#\text{sat assignments to } \phi' = \phi \land \overline{x}_1$. This is called "self-reducibility" and will come back to this later.
usually finite but exp. large

Distribution $\mu$ on large $\Omega$

- **Sampling**: efficiently producing sample $\omega \sim \mu$.
- **Counting**: efficiently computing $\mathbb{P}_\mu[\text{event}]$ for events of interest.

Example: #SAT

$\phi = (x_1 \lor \overline{x}_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_4) \land \cdots$

- $\Omega$ is $\{0, 1\}^n$.
- $\mu$ is uniform over satisfying assignments.

Why is it called **counting**?

- Because $\mathbb{P}[x_1 = 1] = \frac{\#\text{sat assignments of } \phi \text{ with } x_1 = 1}{\#\text{sat assignments of } \phi}$

This is called "self-reducibility" which we will come back to later.
Distribution \( \mu \) on large \( \Omega \)

- **Sampling**: efficiently producing sample \( \omega \sim \mu \).
- **Counting**: efficiently computing \( \mathbb{P}_\mu [\text{event}] \) for events of interest.

**Example: #SAT**

\[
\phi = (x_1 \lor x_2 \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4) \land \cdots
\]

- \( \Omega \) is \( \{0, 1\}^n \).
- \( \mu \) is uniform over satisfying assignments.

- Why is it called *counting*?
- Because \( \mathbb{P}[x_1 = 1] = \frac{\#\text{sat assignments of } \phi \text{ with } x_1 = 1}{\#\text{sat assignments of } \phi} \)

The numerator and denominator are *counts*. 

- In fact, numerator is \( \#\text{sat assignments to } \phi' = \phi \land x_1 \).
- This is called *self-reducibility* — will come back to this later.
Distribution $\mu$ on large $\Omega$

- **Sampling**: efficiently producing sample $\omega \sim \mu$.
- **Counting**: efficiently computing $P_{\mu}[\text{event}]$ for events of interest.

**Example: #SAT**

$\phi = (x_1 \lor \bar{x}_2 \lor x_3) \land (\bar{x}_1 \lor x_2 \land x_4) \land \cdots$

- $\Omega$ is $\{0, 1\}^n$.
- $\mu$ is uniform over satisfying assignments.

- Why is it called counting?
- Because $P[x_1 = 1] = \frac{\#\text{sat assignments of } \phi \text{ with } x_1 = 1}{\#\text{sat assignments of } \phi}$

- The numerator and denominator are counts.

- In fact, numerator is #sat assignments to $\phi' = \phi \land x_1$.

This is called “self-reducibility”. will come back to this later
Formalism

w.r.t. an easy background measure on $\Omega$, usually counting/uniform on finite $\Omega$

Suppose $\mu$ is an unnormalized density:

$\mu : \Omega \to \mathbb{R}_{\geq 0}$
Suppose $\mu$ is an unnormalized density w.r.t. an easy background measure on $\Omega$, usually counting/uniform on finite $\Omega$.

\[ \mu : \Omega \to \mathbb{R}_{\geq 0} \]

**Definition: sampling**

Produce $\omega \in \Omega$ with

\[ P[\omega] \propto \mu(\omega). \]
Suppose $\mu$ is an unnormalized density w.r.t. an easy background measure on $\Omega$, usually counting/uniform on finite $\Omega$:

$$\mu : \Omega \to \mathbb{R}_{\geq 0}$$

### Definition: sampling
Produce $\omega \in \Omega$ with

$$\mathbb{P}[\omega] \propto \mu(\omega).$$

### Definition: counting
Compute the normalizing factor

$$\sum_{\omega} \mu(\omega).$$
Formalism

w.r.t. an easy background measure on $\Omega$, usually counting/uniform on finite $\Omega$

Suppose $\mu$ is an unnormalized density:

$$\mu : \Omega \rightarrow \mathbb{R}_{\geq 0}$$

**Definition: sampling**

Produce $\omega \in \Omega$ with

$$\mathbb{P}[\omega] \propto \mu(\omega).$$

**Definition: counting**

Compute the normalizing factor

$$\sum_{\omega} \mu(\omega).$$

Standard assumption: $\mu$ is easy to compute for any desired point $\omega \in \Omega$. 
Example: SAT

\[ \phi = (x_1 \lor \overline{x_2} \lor x_3) \land (x_1 \lor x_2 \lor x_4) \land \cdots \]

1. \( \Omega = \{0, 1\}^n \) assignments
2. \( \mu(x) = 1[x \text{ satisfies } \phi] \)
Example: SAT

\[ \phi = (x_1 \lor \overline{x}_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_4) \land \cdots \]

\[ \Omega = \{0, 1\}^n \]

\[ \mu(x) = 1[x \text{ satisfies } \phi] \]

Example: spin systems

\[ \Omega = \{+, -\}^V \]

\[ \mu(x) = \prod_{u \sim v} \phi(x_u, x_v) \]

graph G = (V, E)
Example: SAT

\[ \phi = (x_1 \lor \overline{x_2} \lor x_3) \land (x_1 \lor x_2 \lor x_4) \land \cdots \]

\[ \Omega = \{0, 1\}^n \]

\[ \mu(x) = 1[x \text{ satisfies } \phi] \]

Example: generative AI models

\[ \Omega = \{\text{images}\} \]

\[ \Omega = \{\text{text}\} \]

We don’t know \( \mu \). We learn something about it from data. What to learn is often guided by a sampling algorithm.

Example: spin systems

\[ \Omega = \{+,-\}^V \]

\[ \mu(x) = \prod_{u \sim v} \phi(x_u, x_v) \]

Graph \( G = (V, E) \)

Score-based models: \( \nabla \log \mu \)

\[ \frac{\mu(x + \Delta x)}{\mu(x)} \approx \exp(\nabla \log \mu \cdot \Delta x) \]
What is “Counting and Sampling”?

Bit of Complexity Theory
- The class $\#P$
- Parsimonious reductions

Approximation
- Counting: FPTAS/FPRAS
- Sampling: FPAUS
- Equivalence

First Algorithm: DNFs
What is “Counting and Sampling”?

Bit of Complexity Theory
- The class \( \#P \)
- Parsimonious reductions

Approximation
- Counting: FPTAS/FPRAS
- Sampling: FPAUS
- Equivalence

First Algorithm: DNFs
Poly-time nondet. Turing machine $M$

$M : (x, y) \mapsto \{\text{Accept, Reject}\}$

Example: SAT

$M_{\text{SAT}} : (\text{formula } \phi, \text{ assignment } x) \mapsto$

\[
\begin{cases}
\text{Accept} & \text{if } x \text{ satisfies } \phi, \\
\text{Reject} & \text{otherwise.}
\end{cases}
\]
Poly-time nondet. Turing machine $M$

$M : (x, y) \mapsto \{ \text{Accept}, \text{Reject} \}$

Example: SAT

$M_{\text{SAT}} : (\text{formula } \phi, \text{assignment } x) \mapsto$

$$\begin{cases} 
\text{Accept} & \text{if } x \text{ satisfies } \phi, \\
\text{Reject} & \text{otherwise}. 
\end{cases}$$

NP consists of all functions $x \mapsto 1[\exists y : M(x, y) = \text{Accept}].$
Poly-time nondet. Turing machine $M$

\[ M : (x, y) \mapsto \{ \text{Accept, Reject} \} \]

- **NP** consists of all functions
  \[ x \mapsto 1[\exists y : M(x, y) = \text{Accept}] . \]

- **#P** consists of all functions
  \[ x \mapsto |\{ y \mid M(x, y) = \text{Accept} \}|. \]

**Example: SAT**

$M_{\text{SAT}} : (\text{formula } \phi, \text{assignment } x) \mapsto$

\[ \begin{cases} \text{Accept} & \text{if } x \text{ satisfies } \phi, \\ \text{Reject} & \text{otherwise.} \end{cases} \]
Poly-time nondet. Turing machine $M$

$M : (x, y) \mapsto \{\text{Accept, Reject}\}$

Example: SAT

$M_{\text{SAT}} : (\text{formula } \phi, \text{assignment } x) \mapsto$

\[
\begin{cases} 
\text{Accept} & \text{if } x \text{ satisfies } \phi, \\
\text{Reject} & \text{otherwise}.
\end{cases}
\]

NP consists of all functions

$x \mapsto 1[\exists y : M(x, y) = \text{Accept}].$

#P consists of all functions

$x \mapsto |\{y \mid M(x, y) = \text{Accept}\}|.$

Every NP problem has a #P variant.

#SAT

#3-Colorings

#Ind. Sets

#2-SAT

#Matchings

#Trees
Poly-time nondet. Turing machine $M$

$M : (x, y) \mapsto \{\text{Accept}, \text{Reject}\}$

**Example: SAT**

$M_{\text{SAT}} : (\text{formula } \phi, \text{assignment } x) \mapsto$

\[
\begin{cases}
\text{Accept} & \text{if } x \text{ satisfies } \phi, \\
\text{Reject} & \text{otherwise.}
\end{cases}
\]

- NP consists of all functions
  \[x \mapsto 1[\exists y : M(x, y) = \text{Accept}].\]
- \#P consists of all functions
  \[x \mapsto |\{y \mid M(x, y) = \text{Accept}\}|.\]
- Every NP problem has a \#P variant.
- \#P-complete: Every other \#P problem poly-time reduces to it.

\#SAT
\#3-Colorings
\#Ind. Sets
\#2-SAT
\#Matchings
\#Trees
Poly-time nondet. Turing machine $M$

$M: (x, y) \mapsto \{\text{Accept, Reject}\}$

Example: SAT

$M_{SAT}: (\text{formula } \phi, \text{assignment } x) \mapsto$

\[
\begin{cases} 
\text{Accept} & \text{if } x \text{ satisfies } \phi, \\
\text{Reject} & \text{otherwise.}
\end{cases}
\]

- NP consists of all functions
\[x \mapsto 1[\exists y : M(x, y) = \text{Accept}] .\]
- #P consists of all functions
\[x \mapsto |\{y \mid M(x, y) = \text{Accept}\}|.\]
- Every NP problem has a #P variant.
- #P-complete: Every other #P problem poly-time reduces to it.
- Harder than NP-complete!

#SAT  #2-SAT  
#3-Colorings  #Matchings  
#Ind. Sets   #Trees
Poly-time nondet. Turing machine $M$

$M : (x, y) \mapsto \{\text{Accept, Reject}\}$

Example: SAT

$M_{\text{SAT}} : (\text{formula } \phi, \text{assignment } x) \mapsto$

\[
\begin{cases} 
\text{Accept} & \text{if } x \text{ satisfies } \phi, \\
\text{Reject} & \text{otherwise.}
\end{cases}
\]

NP consists of all functions

$x \mapsto 1[\exists y : M(x, y) = \text{Accept}].$

#P consists of all functions

$x \mapsto |\{y \mid M(x, y) = \text{Accept}\}|.$

Every NP problem has a #P variant.

#P-complete: Every other #P problem poly-time reduces to it.

Harder than NP-complete!

- #SAT
- #3-Colorings
- #Ind. Sets
- #2-SAT
- #Matchings
- #Trees
Poly-time nondet. Turing machine $M$

$M : (x, y) \mapsto \{\text{Accept, Reject}\}$

Example: SAT

$M_{\text{SAT}} : (\text{formula } \phi, \text{ assignment } x) \mapsto$

- Accept if $x$ satisfies $\phi$,
- Reject otherwise.

NP consists of all functions

$x \mapsto 1[\exists y : M(x, y) = \text{Accept}].$

#P consists of all functions

$x \mapsto |\{y | M(x, y) = \text{Accept}\}|.$

Every NP problem has a #P variant.

#P-complete: Every other #P problem poly-time reduces to it.

Harder than NP-complete!

- #SAT
- #3-Colorings
- #Ind. Sets

#2-SAT
#Matchings
#Trees
Poly-time nondet. Turing machine $M$

$M : (x, y) \mapsto \{\text{Accept, Reject}\}$

Example: SAT

\[ M_{\text{SAT}} : (\text{formula } \phi, \text{assignment } x) \mapsto \begin{cases} 
\text{Accept} & \text{if } x \text{ satisfies } \phi, \\
\text{Reject} & \text{otherwise}. 
\end{cases} \]

- **NP** consists of all functions
  \[ x \mapsto 1[\exists y : M(x, y) = \text{Accept}]. \]

- **#P** consists of all functions
  \[ x \mapsto |\{y | M(x, y) = \text{Accept}\}|. \]

- Every NP problem has a #P variant.

- **#P-complete**: Every other #P problem poly-time reduces to it.

- Harder than NP-complete!
  - #SAT
  - #3-Colorings
  - #Ind. Sets
  - #2-SAT
  - #Matchings
  - #Trees

- #P-complete: Every other #P problem poly-time reduces to it.
- Harder than NP-complete!
Poly-time nondet. Turing machine $M$

$M : (x, y) \mapsto \{\text{Accept, Reject}\}$

Example: SAT

$M_{\text{SAT}} : (\text{formula } \phi, \text{assignment } x) \mapsto$

\[
\begin{cases}
\text{Accept} & \text{if } x \text{ satisfies } \phi, \\
\text{Reject} & \text{otherwise.}
\end{cases}
\]

- **NP** consists of all functions

\[x \mapsto 1[\exists y : M(x, y) = \text{Accept}]\]

- **#P** consists of all functions

\[x \mapsto |\{y \mid M(x, y) = \text{Accept}\}|\]

- Every NP problem has a #P variant.

- **#P-complete**: Every other #P problem poly-time reduces to it.

- Harder than NP-complete!

  - #SAT
  - #2-SAT
  - #3-Colorings
  - #Ind. Sets
  - #Matchings
  - #Trees
Poly-time nondet. Turing machine $M$

$M : (x, y) \mapsto \{\text{Accept, Reject}\}$

Example: SAT

$M_{\text{SAT}} : (\text{formula } \phi, \text{ assignment } x) \mapsto$

\[
\begin{cases}
\text{Accept} & \text{if } x \text{ satisfies } \phi, \\
\text{Reject} & \text{otherwise}.
\end{cases}
\]

- **NP** consists of all functions
  \[x \mapsto 1[\exists y : M(x, y) = \text{Accept}]\].

- **#P** consists of all functions
  \[x \mapsto |\{y | M(x, y) = \text{Accept}\}|\].

- Every NP problem has a #P variant.
- **#P-complete**: Every other #P problem poly-time reduces to it.
- Harder than NP-complete!

- #SAT
- #3-Colorings
- #Ind. Sets
- #2-SAT
- #Matchings
- #Trees

#P-complete: Every other #P problem poly-time reduces to it.
Poly-time nondet. Turing machine $M$

$M : (x, y) \mapsto \{\text{Accept, Reject}\}$

Example: SAT

$M_{\text{SAT}} : (\text{formula } \phi, \text{assignment } x) \mapsto$

$$\begin{cases} \text{Accept} & \text{if } x \text{ satisfies } \phi, \\ \text{Reject} & \text{otherwise.} \end{cases}$$

- NP consists of all functions
  $$x \mapsto 1[\exists y : M(x, y) = \text{Accept}].$$

- #P consists of all functions
  $$x \mapsto |\{y \mid M(x, y) = \text{Accept}\}|.$$

- Every NP problem has a #P variant.

- #P-complete: Every other #P problem poly-time reduces to it.

- Harder than NP-complete!
  - 😞 #SAT
  - 😞 #2-SAT
  - 😞 #3-Colorings
  - 😞 #Matchings
  - 😞 #Ind. Sets
  - 😁 #Trees

#P-complete: Every other #P problem poly-time reduces to it.

Harder than NP-complete!
All NP problems reduce to SAT [Cook-Levin].

\[ (x_1 \lor \overline{x_2} \lor \cdots) \land \cdots \]
All NP problems reduce to SAT [Cook-Levin].

This reduction is parsimonious. There is a one-to-one correspondence:
accepting paths ↔ sat assignments

m-to-n is also called parsimonious
All NP problems reduce to SAT [Cook-Levin].

This reduction is *parsimonious*. There is a one-to-one correspondence: accepting paths ↔ sat assignments.

Thus #SAT is #P-complete.
All NP problems reduce to SAT [Cook-Levin].

This reduction is **parsimonious**. There is a one-to-one correspondence:

- accepting paths ↔ sat assignments

Thus **#SAT** is **#P**-complete.

In fact, all the natural NP-complete problems we know admit parsimonious reductions: #3-Colorings, #Hamiltonian Cycles, …
All NP problems reduce to SAT [Cook-Levin].

This reduction is **parsimonious**. There is a one-to-one correspondence:

- accepting paths ↔ sat assignments

Thus **#SAT** is **#P-complete**.

In fact, all the natural NP-complete problems we know admit parsimonious reductions: #3-Colorings, #Hamiltonian Cycles, …

**Open problem**: Do all NP-complete problems have a **#P-complete** variant?
#P-complete problems are really hard. At least as hard as NP.
#P-complete problems are really hard. At least as hard as NP.

Much harder: $\text{PH} \subseteq \text{P}^\#P$ [Toda’91]. 😞

poly hierarchy: $x \mapsto 1[\exists y \forall z \exists \cdots M(x, y, z, \ldots)]$
#P-complete problems are really hard. At least as hard as NP.

Much harder: $\text{PH} \subseteq \text{P}^\#P$ [Toda’91]. 😞

poly hierarchy: $x \mapsto 1[\exists y \forall z \exists \cdots M(x, y, z, \ldots)]$

Even P can yield #P-complete!
#P-complete problems are really hard. At least as hard as NP.

Much harder: $\text{PH} \subseteq \text{P}^\#P$ [Toda'91].

Poly hierarchy: $x \mapsto 1[\exists y \forall z \exists \cdots M(x, y, z, \ldots)]$

Even P can yield #P-complete!

**Example: #DNF**

Count sat assignments to DNF:

$$(x_1 \land \overline{x_2} \land x_3) \lor (\cdots) \lor \cdots$$

Proof of hardness: $\#\text{DNF} = 2^n - \#\text{CNF}$. 
- #P-complete problems are really hard. At least as hard as NP.
- Much harder: $\text{PH} \subseteq \text{P}^\#\text{P}$ [Toda’91].
  
  poly hierarchy: $x \mapsto 1[\exists y \forall z \exists \cdots M(x, y, z, \ldots)]$

- Even P can yield #P-complete!

**Example: #DNF**

Count sat assignments to DNF:

$$(x_1 \land \overline{x_2} \land x_3) \lor (\cdots) \lor \cdots$$

Proof of hardness: $\text{#DNF} = 2^n - \text{#CNF}$.  

**Example: bipartite perfect matching**

Counting perfect matchings in bipartite graphs is #P-complete. [Valiant’79]
#P-complete problems are really hard. At least as hard as NP.

Much harder: $\text{PH} \subseteq \text{P}^{\#P}$ [Toda’91].

poly hierarchy: $x \mapsto 1[\exists y \forall z \exists \cdots M(x, y, z, \ldots)]$

Even P can yield #P-complete!

**Example: #DNF**

Count sat assignments to DNF:

$$(x_1 \land \overline{x_2} \land x_3) \lor (\cdots) \lor \cdots$$

Proof of hardness: $\#\text{DNF} = 2^n - \#\text{CNF}$.

**Example: bipartite perfect matching**

Counting perfect matchings in bipartite graphs is #P-complete. [Valiant’79]

Reductions are not parsimonious.
- #P-complete problems are really hard. At least as hard as NP.
- Much harder: $\text{PH} \subseteq \text{P}^\#P$ [Toda’91].
  
  poly hierarchy: $x \mapsto 1 [\exists y \forall z \exists \ldots M(x, y, z, \ldots)]$

- Even P can yield #P-complete!

**Example: #DNF**

Count sat assignments to DNF:

$$(x_1 \land \overline{x_2} \land x_3) \lor (\ldots) \lor \ldots$$

Proof of hardness: $\#\text{DNF} = 2^n - \#\text{CNF}$.  

**Example: bipartite perfect matching**

Counting perfect matchings in bipartite graphs is #P-complete. [Valiant’79]

- Reductions are not parsimonious.
- Observation: efficient counting known for only a handful of gems: spanning trees, planar perf. matchings, Eulerian circuits, \ldots
All hope is lost?
What is “Counting and Sampling”?  

**Bit of Complexity Theory**  
- The class \( \mathbb{#P} \)  
- Parsimonious reductions  

**Approximation**  
- Counting: FPTAS/FPRAS  
- Sampling: FPAUS  
- Equivalence  

**First Algorithm: DNFs**
What is “Counting and Sampling”?

Bit of Complexity Theory
- The class \(\#P\)
- Parsimonious reductions

Approximation
- Counting: FPTAS/FPRAS
- Sampling: FPAUS
- Equivalence

First Algorithm: DNFs
Approximation to the rescue

Approx. counting: output $Z$ with

$$Z \leq \text{count} \leq (1 + \epsilon)Z.$$
Approximation to the rescue

- **Approx. counting:** output $Z$ with
  
  $$Z \leq \text{count} \leq (1 + \epsilon)Z.$$  

- **Fully poly-time approx. scheme:**
  above with runtime $\text{poly}(n, 1/\epsilon)$.

### Exercise

$2/3$ can be replaced by $1 - \delta$ with runtime $\text{poly}(n, 1/\epsilon, \log(1/\delta))$.  

Why all $\epsilon$? Why not $100$-approx?

Approx. counting is all-or-nothing.

**Example:** #SAT

Suppose $A$ is $f(n)$-approx. alg. Give $\phi(1) \land \phi(2) \land \cdots \land \phi(t)$ with $\phi(i)$ being disjoint copies of $\phi$.  

$t \sqrt{\text{output}} \approx \#\text{SAT}(\phi)$.

Approx. ratio is $t \sqrt{f(nt)}$. Even for $f(n) = 2^n^{0.99}$, enough to set $t = \text{poly}(n, 1/\epsilon)$ to get $t \sqrt{f(nt)} \leq 1 + \epsilon$.  

FPTAS
Approximation to the rescue

Approx. counting: output $Z$ with

$$Z \leq \text{count} \leq (1 + \epsilon)Z.$$  

FPTAS

Fully poly-time approx. scheme:
above with runtime $\text{poly}(n, 1/\epsilon)$.

FPRAS, input size

Fully poly rand. approx. scheme:
above but with randomness and
2/3 chance of success.
Approximation to the rescue

- **Approx. counting:** output $Z$ with
  $$Z \leq \text{count} \leq (1 + \epsilon)Z.$$  
  **FPTAS**

- **Fully poly-time approx. scheme:**
  above with runtime $\text{poly}(n, 1/\epsilon)$.
  **FPRAS**

- **Fully poly rand. approx. scheme:**
  above but with randomness and $2/3$ chance of success.

- **Exercise:** $2/3$ can be replaced by $1 - \delta$ with runtime
  $\text{poly}(n, 1/\epsilon, \log(1/\delta))$. 

Why all $\epsilon$? Why not $100$-approx?
Approx. counting is all-or-nothing.

Example: \#SAT
Suppose $A$ is $f(n)$-approx. alg. Give
$$\phi(1) \land \phi(2) \land \cdots \land \phi(t)$$
with $\phi(i)$ being disjoint copies of $\phi$.

$t \sqrt{\text{output}} \approx \#\text{SAT}(\phi)$.

Approx. ratio is $t \sqrt{f(nt)}$. Even for
$f(n) = 2^{0.99n}$, enough to set $t = \text{poly}(n, 1/\epsilon)$
to get $t \sqrt{f(nt)} \leq 1 + \epsilon$.
Approximation to the rescue

- Approx. counting: output $Z$ with $Z \leq \text{count} \leq (1 + \epsilon)Z$.

- Fully poly-time approx. scheme: above with runtime $\text{poly}(n, 1/\epsilon)$.

- Fully poly rand. approx. scheme: above but with randomness and $2/3$ chance of success.

- Exercise: $2/3$ can be replaced by $1 - \delta$ with runtime $\text{poly}(n, 1/\epsilon, \log(1/\delta))$.

Why all $\epsilon$? Why not $100$-approx?

Approx. ratio is $t \sqrt{f(nt)}$. Even for $f(n) = 2^{0.99n}$, enough to set $t = \text{poly}(n, 1/\epsilon)$ to get $t \sqrt{f(nt)} \leq 1 + \epsilon$.
Approximation to the rescue

- **Approx. counting**: output $Z$ with
  \[ Z \leq \text{count} \leq (1 + \epsilon)Z. \]

- **Fully poly-time approx. scheme**:
  above with runtime $\text{poly}(n, 1/\epsilon)$.

- **Fully poly rand. approx. scheme**:
  above but with randomness and $2/3$ chance of success.

- **Exercise**: $2/3$ can be replaced by $1 - \delta$ with runtime $\text{poly}(n, 1/\epsilon, \log(1/\delta))$.

- Why all $\epsilon$? Why not 100-approx?
- Approx. counting is all-or-nothing.
Approximation to the rescue

- **Approx. counting:** output $Z$ with
  \[ Z \leq \text{count} \leq (1 + \epsilon)Z. \]

  - Fully poly-time approx. scheme:
    above with runtime $\text{poly}(n, 1/\epsilon)$.

  - Fully poly rand. approx. scheme:
    above but with randomness and 2/3 chance of success.

  - Exercise: 2/3 can be replaced by $1 - \delta$ with runtime $\text{poly}(n, 1/\epsilon, \log(1/\delta))$.

  - Why all $\epsilon$? Why not 100-approx?
  
  - Approx. counting is all-or-nothing.

**Example: \#SAT**

Suppose $A$ is $f(n)$-approx. alg. Give

\[ \phi^{(1)} \land \phi^{(2)} \land \ldots \land \phi^{(t)} \]

with $\phi^{(i)}$ being disjoint copies of $\phi$.

\[
\sqrt[\text{t}]\text{output} \approx \#\text{SAT}(\phi).
\]

Approx. ratio is $\sqrt[\text{t}]f(nt)$. Even for $f(n) = 2^{n^{0.99}}$, enough to set $t = \text{poly}(n, 1/\epsilon)$ to get $\sqrt[\text{t}]f(nt) \leq 1 + \epsilon$. 
For any “tensorizable” problem, nothing between $1 + \varepsilon$ and exponential.
Approximation is all-or-nothing

- For any “tensorizable” problem, nothing between $1 + \epsilon$ and exponential.
- For “self-reducible” probs, $\text{poly}(n)$-approx gives an FPRAS. [Jerrum-Sinclair].

[will see later in the course]
Notion of approximation: For dists $\nu$, $\mu$ on $\Omega$ we use total variation:

$$d_{TV}(\nu, \mu) = \max \left\{ \mathbb{P}_{\nu}[E] - \mathbb{P}_{\mu}[E] \mid E \subseteq \Omega \right\}$$

$$= \frac{1}{2} \sum_{\omega \in \Omega} |\mu(\omega) - \nu(\omega)|.$$
Approximate sampling

- **Notion of approximation**: For dists \( \nu, \mu \) on \( \Omega \) we use total variation:

\[
d_{TV}(\nu, \mu) = \max\{P_{\nu}[E] - P_{\mu}[E] \mid E \subseteq \Omega\} = \frac{1}{2} \sum_{\omega \in \Omega} |\mu(\omega) - \nu(\omega)|.
\]

- **Fully poly approx. unif. sampler**: Output distribution has \( d_{TV} \leq \delta \) and runtime is \( \text{poly}(n, \log(1/\delta)) \).
Approximate sampling

- **Notion of approximation:** For dists $\nu, \mu$ on $\Omega$ we use **total variation**:

  \[ d_{TV}(\nu, \mu) = \max \{ \mathbb{P}_\nu[E] - \mathbb{P}_\mu[E] \mid E \subseteq \Omega \} = \frac{1}{2} \sum_{\omega \in \Omega} |\mu(\omega) - \nu(\omega)|. \]

- **Fully poly approx. unif. sampler:**
  - output distribution has $d_{TV} \leq \delta$
  - and runtime is $\text{poly}(n, \log(1/\delta))$.

- **Note:** the log dependence on $\delta$ is similarly “all-or-nothing”.

---

Theorem [Jerrum-Valiant-Vazirani]

For “self-reducible” problems:

- approx counting $\equiv$ approx sampling

---

<table>
<thead>
<tr>
<th>Exact Counting</th>
<th>Approx Counting (FPRAS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Sampling</td>
<td>Approx Sampling (FPAUS)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FPTAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrows are poly-time reductions</td>
</tr>
</tbody>
</table>
Approximate sampling

- **Notion of approximation:** For dists $\nu, \mu$ on $\Omega$ we use total variation:

  \[ d_{TV}(\nu, \mu) = \max\{P_{\nu}[E] - P_{\mu}[E] \mid E \subseteq \Omega\} = \frac{1}{2} \sum_{\omega \in \Omega} |\mu(\omega) - \nu(\omega)|. \]

- **Fully poly approx. unif. sampler:** Output distribution has $d_{TV} \leq \delta$ and runtime is $\text{poly}(n, \log(1/\delta))$.

- **Note:** The log dependence on $\delta$ is similarly “all-or-nothing”.

---

**Theorem [Jerrum-Valiant-Vazirani]**

For “self-reducible” problems:

approx counting $\equiv$ approx sampling

\[ \text{(FPRAS)} \]

Exact Counting $\longrightarrow$ Approx Counting

Exact Sampling $\longrightarrow$ Approx Sampling

\[ \text{if FPTAS} \]

arrows are poly-time reductions

\[ \text{(FPAUS)} \]
Counting via Markov chains

Basis of Markov Chain
Monte Carlo: Approx Sampler → Approx Counter.
Counting via Markov chains

- Basis of Markov Chain Monte Carlo: Approx Sampler → Approx Counter.
- A good portion of this course will be on sampling via Markov chains.

\[ x_0 \rightarrow x_1 \rightarrow \cdots \rightarrow x_t \]

hope this is close to \( \mu \)
What is “Counting and Sampling”? 

Bit of Complexity Theory 
- The class $\#P$
- Parsimonious reductions

Approximation
- Counting: FPTAS/FPRAS
- Sampling: FPAUS
- Equivalence

First Algorithm: DNFs
What is “Counting and Sampling”? 

Bit of Complexity Theory

- The class $\#P$
- Parsimonious reductions

Approximation

- Counting: FPTAS/FPRAS
- Sampling: FPAUS
- Equivalence

First Algorithm: DNFs
Given DNF formula

\[ \phi = (x_1 \land \overline{x_2} \land x_3) \lor \cdots, \]

can we approx sample/count satisfying assignments?
Given DNF formula

\[ \phi = (x_1 \land \overline{x_2} \land x_3) \lor \cdots , \]

can we approx sample/count satisfying assignments?

**Naïve attempt**

```
while not accepted do
    sample \( x \in \{0, 1\}^n \) u.a.r.
    if \( x \) sats \( \phi \) then
        accept and return \( x \)
```

This is an instance of rejection sampling.

Rejection sampling
We have access to sampler for \( \nu \), but want samples \( \propto \mu \):

```
while not accepted do
    sample \( x \sim \nu \) accept w.p.
```

\( c \) small enough that prob is always \( \leq 1 \)

\( \mu(\cdot)/\nu(\cdot) \)

Output is always \( \sim \) normalized \( \mu \)
Can take a long time
If \( \mu \) is normalized, the best \( c \) is

\( \min \{ \nu(\cdot)/\mu(\cdot) \} \), and it takes \( \approx \max \{ \mu(\cdot)/\nu(\cdot) \} \) iterations.

For \( \phi = (x_1 \land \overline{x_2} \land x_3) \lor \cdots \) it takes \( 2^n \) tries on average.
Given DNF formula

\[ \phi = (x_1 \land \overline{x_2} \land x_3) \lor \cdots, \]

can we approx sample/count satisfying assignments?

**Naïve attempt**

```plaintext
while not accepted do
    sample \( x \in \{0, 1\}^n \) u.a.r.
    if \( x \) sats \( \phi \) then
        accept and return \( x \)
```

This is an instance of rejection sampling.
Given DNF formula
\[ \phi = (x_1 \land \overline{x_2} \land x_3) \lor \cdots, \]
can we approx sample/count satisfying assignments?

**Naïve attempt**

\[
\text{while not accepted do}
\begin{align*}
\text{sample } x & \in \{0, 1\}^n \text{ u.a.r.} \\
\text{if } x \text{ sats } \phi & \text{ then} \\
& \text{accept and return } x
\end{align*}
\]

This is an instance of rejection sampling.

**Rejection sampling**

We have access to sampler for \( \nu \), but want samples \( \propto \mu \):

\[
\text{while not accepted do}
\begin{align*}
\text{sample } x & \sim \nu \\
\text{accept w.p. } c \mu(x) / \nu(x)
\end{align*}
\]

small enough that prob is always \( \leq 1 \)

If \( \mu \) is normalized, the best \( c \) is
\[ \min \left\{ \nu(x) / \mu(x) \right\}, \]
and it takes \( \approx \max \left\{ \mu(x) / \nu(x) \right\} \) iterations.

For \( \phi = (x_1 \land x_2 \land \cdots \land x_n) \) it takes \( 2^n \) tries on average.
Given DNF formula

\[ \phi = (x_1 \land \overline{x_2} \land x_3) \lor \cdots , \]

can we approx sample/count satisfying assignments?

**Naïve attempt**

```
while not accepted do
  sample \( x \in \{0, 1\}^n \) u.a.r.
  if \( x \) sats \( \phi \) then
    accept and return \( x \)
```

This is an instance of rejection sampling.

**Rejection sampling**

We have access to sampler for \( \nu \), but want samples \( \propto \mu \):

```
while not accepted do
  sample \( x \sim \nu \)
  accept w.p. \( \frac{\mu(x)}{\nu(x)} \)
```

small enough that prob is always \( \leq 1 \)

Output is always \( \sim \) normalized \( \mu \)

Can take a long time

If \( \mu \) is normalized, the best \( c \) is

\[ \min \{ \frac{\nu(x)}{\mu(x)} \} \]

and it takes

\[ \approx \max \{ \frac{\mu(x)}{\nu(x)} \} \] iterations.

For \( \phi = (x_1 \land \overline{x_2} \land x_3) \lor \cdots \) it takes \( 2^n \) tries on average.
Given DNF formula

\[ \phi = (x_1 \land \overline{x_2} \land x_3) \lor \cdots, \]

can we approx sample/count satisfying assignments?

### Naïve attempt

```plaintext
while not accepted do
    sample \( x \in \{0, 1\}^n \) u.a.r.
    if \( x \) sats \( \phi \) then
        accept and return \( x \)
```

- This is an instance of rejection sampling.

### Rejection sampling

We have access to sampler for \( \nu \), but want samples \( \propto \mu \):

```plaintext
while not accepted do
    sample \( x \sim \nu \)
    accept w.p. \[ \frac{c \mu(x)}{\nu(x)} \]
```

small enough that prob is always \( \leq 1 \)

- Output is always \( \sim \) normalized \( \mu \)
- Can take a long time 😞
Given DNF formula

\[ \phi = (x_1 \land \overline{x_2} \land x_3) \lor \cdots, \]

can we approx sample/count satisfying assignments?

**Naïve attempt**

```
while not accepted do
    sample \( x \in \{0, 1\}^n \) u.a.r.
    if \( x \) sats \( \phi \) then
        accept and return \( x \)
```

This is an instance of rejection sampling.

**Rejection sampling**

We have access to sampler for \( \nu \), but want samples \( \propto \mu \):

```
while not accepted do
    sample \( x \sim \nu \)
    accept w.p. \( c \mu(x)/\nu(x) \)
```

small enough that prob is always \( \leq 1 \)

- Output is always \( \sim \) normalized \( \mu \)
- Can take a long time
- If \( \mu \) is normalized, the best \( c \) is
  \( \min\{\nu(x)/\mu(x)\} \), and it takes
  \( \approx \max\{\mu(x)/\nu(x)\} \) iterations.
Given DNF formula

\[ \phi = (x_1 \land \overline{x_2} \land x_3) \lor \cdots , \]

can we approx sample/count satisfying assignments?

**Naïve attempt**

**while not accepted do**
- sample \( x \in \{0, 1\}^n \) u.a.r.
- if \( x \) sats \( \phi \) then
  - accept and return \( x \)

▷ This is an instance of rejection sampling.

**Rejection sampling**

We have access to sampler for \( \nu \), but want samples \( \propto \mu \):

**while not accepted do**
- sample \( x \sim \nu \)
- accept w.p. \( c \mu(x)/\nu(x) \)

small enough that prob is always \( \leq 1 \)

▷ Output is always \( \sim \) normalized \( \mu \)
▷ Can take a long time 😞
▷ If \( \mu \) is normalized, the best \( c \) is \( \min \{ \nu(x)/\mu(x) \} \), and it takes \( \simeq \max \{ \mu(x)/\nu(x) \} \) iterations.
▷ For \( \phi = (x_1 \land x_2 \land \cdots \land x_n) \) it takes \( 2^n \) tries on average. 😞
A better envelope [Karp-Luby]

\[ \phi = C_1 \lor C_2 \lor \cdots \lor C_m \]
Let $A_i = \{\text{sat assignments of } C_i\}$. 

\[ \phi = C_1 \lor C_2 \lor \cdots \lor C_m \]
A better envelope [Karp-Luby]

\[ \phi = C_1 \lor C_2 \lor \cdots \lor C_m \]

- Let \( A_i = \{ \text{sat assignments of } C_i \} \).
- Want to sample from \( A_1 \cup \cdots \cup A_m \).

nexxt lecture: turning this into approx counting.
A better envelope [Karp-Luby]

\[ \phi = C_1 \lor C_2 \lor \cdots \lor C_m \]

- Let \( A_i = \{\text{sat assignments of } C_i\} \).
- Want to sample from \( A_1 \cup \cdots \cup A_m \).
- **Idea:** sample from \( A_1 \biguplus \cdots \biguplus A_m \) and rejection sample it into \( A_1 \cup \cdots \cup A_m \).

\[
\begin{align*}
\text{while not accepted} \quad & \text{do} \\
\text{sample } x & \in A_1 \biguplus \cdots \biguplus A_m \text{ u.a.r.} \\
\text{if } x & \text{ is sampled from } A_i \text{ and } x \not\in A_j \text{ for all } j < i \quad & \text{then accept and return } x
\end{align*}
\]

Chance of acceptance \( \geq \frac{1}{m} \).

On average \( \approx m \) iterations suffice.

Next lecture: turning this into approx counting.
A better envelope [Karp-Luby]

\[ \phi = C_1 \lor C_2 \lor \cdots \lor C_m \]

- Let \( A_i = \{\text{sat assignments of } C_i\} \).
- Want to sample from \( A_1 \cup \cdots \cup A_m \).
- Idea: sample from \( A_1 \coprod \cdots \coprod A_m \) and rejection sample it into \( A_1 \cup \cdots \cup A_m \).

**while not accepted do**
  - sample \( x \in A_1 \coprod \cdots \coprod A_m \) u.a.r.
  - if \( x \) is sampled from \( A_i \) and \( x \notin A_j \) for all \( j < i \) then
    - accept and return \( x \)

Chance of acceptance \( \geq \frac{1}{m} \).
On average \( \approx m \) iterations suffice.

Next lecture: turning this into approx counting.
A better envelope [Karp-Luby]

\[ \phi = C_1 \lor C_2 \lor \cdots \lor C_m \]

- Let \( A_i = \{ \text{sat assignments of } C_i \} \).
- Want to sample from \( A_1 \cup \cdots \cup A_m \).
- Idea: sample from \( A_1 \uplus \cdots \uplus A_m \) and rejection sample it into \( A_1 \cup \cdots \cup A_m \).

while not accepted do
  sample \( x \in A_1 \uplus \cdots \uplus A_m \) u.a.r.
  if \( x \) is sampled from \( A_i \) and \( x \notin A_j \) for all \( j < i \) then
    accept and return \( x \)

\( \text{Chance of acceptance } \geq \frac{1}{m} \).
A better envelope \([\text{Karp-Luby}]\)

\[
\phi = C_1 \lor C_2 \lor \cdots \lor C_m
\]

- Let \(A_i = \text{\{sat assignments of } C_i \text{\}}\).
- Want to sample from \(A_1 \cup \cdots \cup A_m\).
- Idea: sample from \(A_1 \bigcup \cdots \bigcup A_m\) and rejection sample it into \(A_1 \cup \cdots \cup A_m\).

\[
\text{while not accepted do }
\]

- sample \(x \in A_1 \bigcup \cdots \bigcup A_m\) \text{ u.a.r.}
- if \(x\) is sampled from \(A_i\) and
  - \(x \notin A_j\) for all \(j < i\)
  - accept and return \(x\)

- Chance of acceptance \(\geq 1/m\).
- On average \(\approx m\) iterations suffice.

Next lecture: turning this into approx counting.
A better envelope [Karp-Luby]

\[ \phi = C_1 \lor C_2 \lor \cdots \lor C_m \]

- Let \( A_i = \{\text{sat assignments of } C_i\} \).
- Want to sample from \( A_1 \cup \cdots \cup A_m \).
- Idea: sample from \( A_1 \sqcup \cdots \sqcup A_m \) and rejection sample it into \( A_1 \cup \cdots \cup A_m \).

**while** not accepted **do**

- sample \( x \in A_1 \sqcup \cdots \sqcup A_m \) u.a.r.
- if \( x \) is sampled from \( A_i \) and \( x \notin A_j \) for all \( j < i \) then
  - accept and return \( x \)

- Chance of acceptance \( \geq \frac{1}{m} \).
- On average \( \sim m \) iterations suffice.
- Next lecture: turning this into approx counting.