CS 263: Counting and Sampling

Nima Anari

Stanford University

slides for

Introduction

Logistics

- Course staff:

Nima Anari
(Instructor)

Victor Lecomte
(Course Assistant)

Logistics

- Course staff:

Nima Anari
(Instructor)

Victor Lecomte
(Course Assistant)

- You:

$\underbrace{\text{~39 undergrad + masters + Ph.D.}}$

Logistics

- Course staff:

Nima Anari
(Instructor)

Victor Lecomte
(Course Assistant)

- You:

$\sim$39 undergrad + masters + Ph.D.

https://cs263.stanford.edu

Logistics

- Course staff:

Nima Anari
(Instructor)

Victor Lecomte
(Course Assistant)

- You:

$\overbrace{\text{~39 undergrad + masters + Ph.D.}}$

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 . . .

Logistics

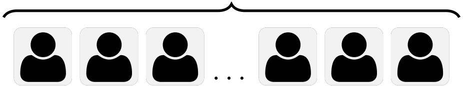
- Course staff:

Nima Anari
(Instructor)

Victor Lecomte
(Course Assistant)

- You:

$\sim$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:

$\sim$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 $\geqslant 3$ papers) or research (new progress) on topics related to the course

Logistics

- Course staff:

Nima Anari (Instructor) | Victor Lecomte (Course Assistant)

- You:

$\underbrace{\text{~39 undergrad + masters + Ph.D.}}$

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 $\geqslant 3$ papers) or research (new progress) on topics related to the course
- Office hours: Starting next week

What is “Counting and Sampling”?

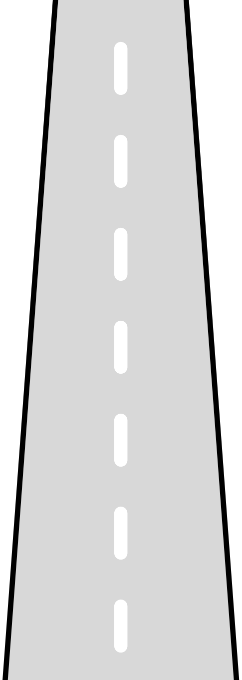
Bit of Complexity Theory

- The class #P
- Parsimonious reductions

Approximation

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

First Algorithm: DNFs



What is “Counting and Sampling”?

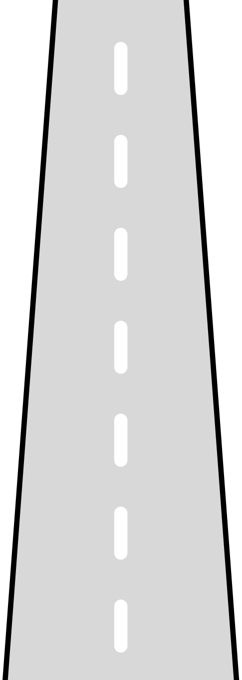
Bit of Complexity Theory

- The class #P
- Parsimonious reductions

Approximation

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

First Algorithm: DNFs



usually finite but exp. large

Distribution μ on large Ω

usually finite but exp. large

Distribution μ on large Ω

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

usually finite but exp. large

Distribution μ on large Ω

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

usually finite but exp. large

Distribution μ on large Ω

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

Example: #SAT

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge \cdots$$

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

usually finite but exp. large

Distribution μ on large Ω

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

Example: #SAT

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge \cdots$$

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

- Why is it called counting?

usually finite but exp. large

Distribution μ on large Ω

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

Example: #SAT

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge \cdots$$

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

- Why is it called counting?
- Because $\mathbb{P}[x_1 = 1] =$

$$\frac{\text{\#sat assignments of } \phi \text{ with } x_1 = 1}{\text{\#sat assignments of } \phi}$$

usually finite but exp. large

Distribution μ on large Ω

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

Example: #SAT

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge \cdots$$

- Ω is $\{0, 1\}^n$.
- μ 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.

usually finite but exp. large

Distribution μ on large Ω

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

Example: #SAT

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge \cdots$$

- Ω is $\{0,1\}^n$.
- μ 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 #sat assignments to

$$\phi' = \phi \wedge x_1.$$

This is called "self-reducibility".

will come back to this later

Formalism

w.r.t. an easy background measure on Ω, usually counting/uniform on finite Ω

Suppose μ is an unnormalized density:

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

Formalism

w.r.t. an easy background measure on Ω, usually counting/uniform on finite Ω

Suppose μ is an unnormalized density:

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

Definition: sampling

Produce $\omega \in \Omega$ with

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

Formalism

w.r.t. an easy background measure on Ω, usually counting/uniform on finite Ω

Suppose μ is an unnormalized density:

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

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 Ω, usually counting/uniform on finite Ω

Suppose μ is an unnormalized density:

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

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: μ is easy to compute for any desired point $\omega \in \Omega$.

Example: SAT

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge \cdots$$

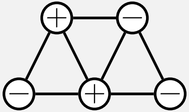
- $\Omega = \{0,1\}^n$ ← assignments
- $\mu(x) = \mathbb{1}[x \text{ satisfies } \phi]$

Example: SAT

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge \cdots$$

- $\Omega = \{0,1\}^n$ ⟵ assignments
- $\mu(x) = \mathbb{1}[x \text{ satisfies } \phi]$

Example: spin systems



graph $G = (V, E)$

- $\Omega = \{+,-\}^V$ ⟵ could be larger
- $\mu(x) = \prod_{u \sim v} \phi(x_u, x_v)$

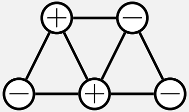
local interaction

Example: SAT

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge \cdots$$

- $\Omega = \{0, 1\}^n \longleftarrow$ assignments
- $\mu(x) = \mathbb{1}[x \text{ satisfies } \phi]$

Example: spin systems



graph $G = (V, E)$

- $\Omega = \{+, -\}^V \longleftarrow$ could be larger
- $\mu(x) = \prod_{u \sim v} \phi(x_u, x_v)$

local interaction

Example: generative AI models

- $\Omega = \{$, , , $\ldots\} \longleftarrow$ images
- $\Omega = \{\text{good job, slay, sus}, \ldots\} \longleftarrow$ text

We don't know μ. We learn something about it from data. What to learn is often guided by a sampling algorithm.

- Score-based models: $\nabla \log \mu$

nearby points

$x \qquad x + \Delta x$

$$\frac{\mu(x + \Delta x)}{\mu(x)} \simeq \exp(\nabla \log \mu \cdot \Delta x).$$

What is “Counting and Sampling”?

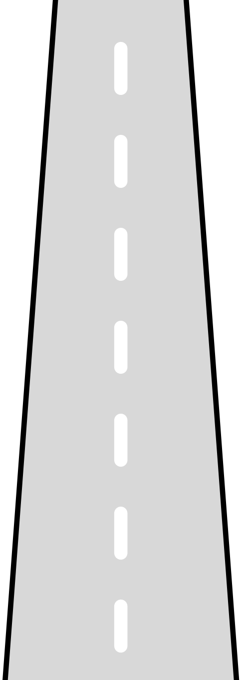
Bit of Complexity Theory

- The class #P
- Parsimonious reductions

Approximation

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

First Algorithm: DNFs



What is “Counting and Sampling”?

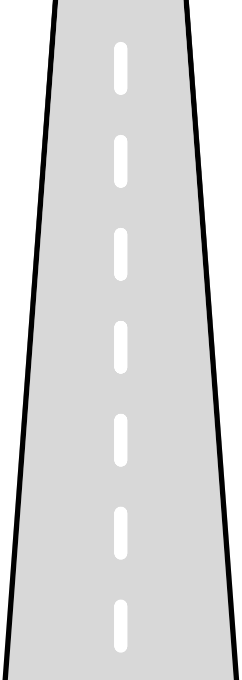
Bit of Complexity Theory

- The class #P
- Parsimonious reductions

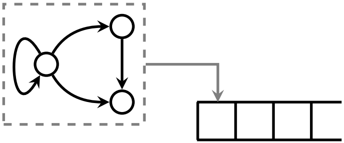
Approximation

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

First Algorithm: DNFs



Poly-time nondet. Turing machine M



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

input

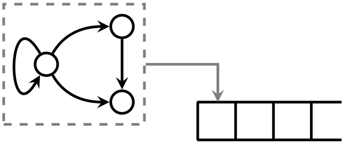
witness/nondet. choices

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



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

input witness/nondet. choices

Example: SAT

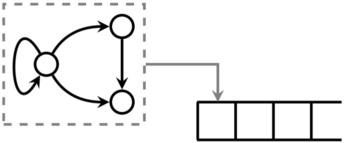
$M_{\text{SAT}} : (\text{formula } \phi, \text{assignment } x) \mapsto$

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

- NP consists of all functions

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

Poly-time nondet. Turing machine M



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

input witness/nondet. choices

Example: SAT

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

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

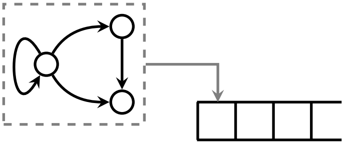
- NP consists of all functions

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

- #P consists of all functions

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

Poly-time nondet. Turing machine M



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

input

witness/nondet. choices

Example: SAT

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

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

- NP consists of all functions

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

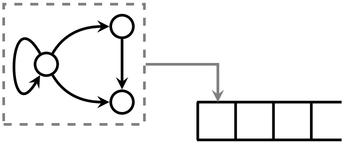
- #P consists of all functions

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

- Every NP problem has a #P variant.

not unique

Poly-time nondet. Turing machine M



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

input witness/nondet. choices

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

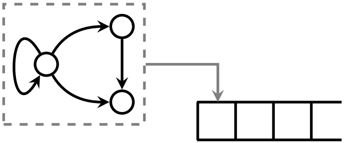
$$x \mapsto \mathbb{1}[\exists y : M(x, y) = \text{Accept}] \,.$$

- #P consists of all functions

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

- Every NP problem has a #P variant.
  not unique
- #P-complete: Every other #P problem poly-time reduces to it.

Poly-time nondet. Turing machine M



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

input

witness/nondet. choices

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

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

- #P consists of all functions

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

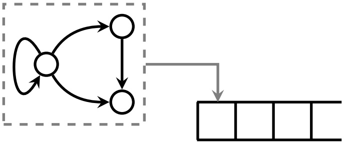
- Every NP problem has a #P variant.

  not unique

- #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



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

input witness/nondet. choices

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

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

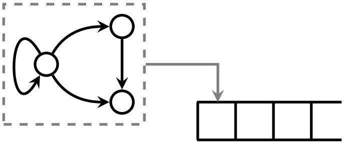
- #P consists of all functions

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

- Every NP problem has a #P variant.
  not unique
- #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



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

input

witness/nondet. choices

Example: SAT

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

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

- NP consists of all functions

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

- #P consists of all functions

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

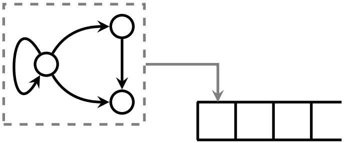
- Every NP problem has a #P variant.

not unique

- #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



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

input

witness/nondet. choices

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

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

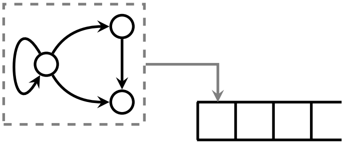
- #P consists of all functions

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

- Every NP problem has a #P variant.
  not unique
- #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



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

input

witness/nondet. choices

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

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

- #P consists of all functions

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

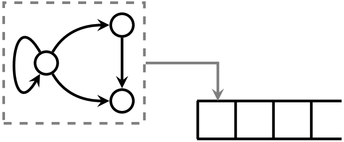
- Every NP problem has a #P variant.

  not unique

- #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



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

input

witness/nondet. choices

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

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

- #P consists of all functions

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

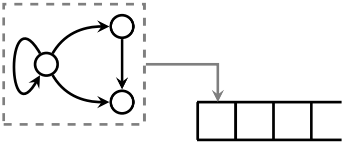
- Every NP problem has a #P variant.

  not unique

- #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



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

input

witness/nondet. choices

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

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

- #P consists of all functions

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

- Every NP problem has a #P variant.

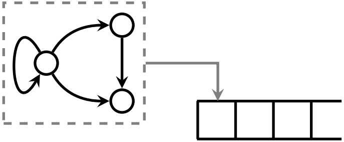
not unique

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

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

Reductions

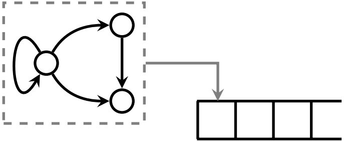
All NP problems reduce to SAT [Cook-Levin].



$\xmapsto{\text{[Cook-Levin] reduction}} (x_1 \vee \overline{x_2} \vee \cdots) \wedge \cdots$

Reductions

All NP problems reduce to SAT [Cook-Levin].



$\xmapsto{\text{[Cook-Levin] reduction}} (x_1 \vee \overline{x_2} \vee \cdots) \wedge \cdots$

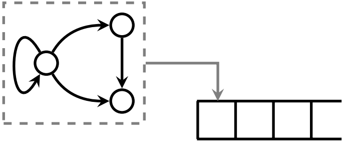
- This reduction is parsimonious. There is a one-to-one correspondence:

  accepting paths $\leftrightarrow$ sat assignments

  m-to-n is also called parsimonious

Reductions

All NP problems reduce to SAT [Cook-Levin].



$\xmapsto{\text{[Cook-Levin] reduction}}$ $(x_1 \vee \overline{x_2} \vee \cdots) \wedge \cdots$

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

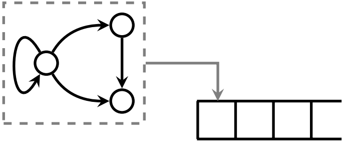
  accepting paths $\leftrightarrow$ sat assignments

- Thus #SAT is #P-complete.

m-to-n is also called parsimonious

Reductions

All **NP** problems reduce to SAT [Cook-Levin].



$\longmapsto$ [Cook-Levin] reduction $(x_1 \vee \overline{x_2} \vee \cdots) \wedge \cdots$

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

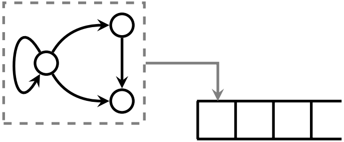
  accepting paths $\leftrightarrow$ sat assignments

  m-to-n is also called parsimonious

- Thus #SAT is #**P**-complete.
- In fact, all the natural **NP**-complete problems we know admit parsimonious reductions: #3-Colorings, #Hamiltonian Cycles, ...

Reductions

All NP problems reduce to SAT [Cook-Levin].



[Cook-Levin] reduction $\longmapsto (x_1 \vee \overline{x_2} \vee \cdots) \wedge \cdots$

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

accepting paths$\leftrightarrow$sat assignments

- Thus #SAT is #P-complete.

m-to-n is also called parsimonious

- 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: PH$\subseteq$ P$^{\#\mathrm{P}}$ [Toda'91]. ☹

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

- #P-complete problems are really hard. At least as hard as NP.
- Much harder: PH$\subseteq$ P$^{\#\mathrm{P}}$ [Toda'91]. ☹

poly hierarchy: $x \mapsto \mathbb{1}[\exists y \forall z \exists \cdots M(x, y, z, \dots)]$

- Even P can yield #P-complete!

- #P-complete problems are really hard. At least as hard as NP.
- Much harder: PH$\subseteq$ P$^{\#\mathrm{P}}$ [Toda'91]. ☹

poly hierarchy: $x \mapsto \mathbb{1}[\exists y \forall z \exists \cdots M(x, y, z, \dots)]$

- Even P can yield #P-complete!

Example: #DNF

Count sat assignments to DNF:

$$(x_1 \wedge \overline{x_2} \wedge x_3) \vee (\cdots) \vee \cdots$$

Proof of hardness: #DNF $= 2^n -$#CNF.

- #P-complete problems are really hard. At least as hard as NP.
- Much harder: PH$\subseteq$ P$^{\#P}$ [Toda'91]. 🙁

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

- Even P can yield #P-complete!

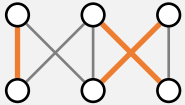
Example: #DNF

Count sat assignments to DNF:

$$(x_1 \wedge \overline{x_2} \wedge x_3) \vee (\cdots) \vee \cdots$$

Proof of hardness: #DNF $= 2^n -$#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: PH$\subseteq$ P$^{\#P}$ [Toda'91]. ☹

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

- Even P can yield #P-complete!

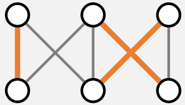
Example: #DNF

Count sat assignments to DNF:

$$(x_1 \wedge \overline{x_2} \wedge x_3) \vee (\cdots) \vee \cdots$$

Proof of hardness: #DNF $= 2^n -$#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: PH$\subseteq$ P$^{\#P}$ [Toda'91]. 😟

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

- Even P can yield #P-complete!

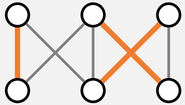
Example: #DNF

Count sat assignments to DNF:

$$(x_1 \wedge \overline{x_2} \wedge x_3) \vee (\cdots) \vee \cdots$$

Proof of hardness: #DNF $= 2^n -$#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, ...

will come back to them

All hope is lost?

What is “Counting and Sampling”?

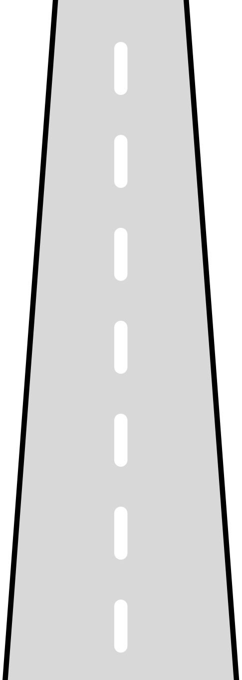
Bit of Complexity Theory

- The class #P
- Parsimonious reductions

Approximation

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

First Algorithm: DNFs



What is “Counting and Sampling”?

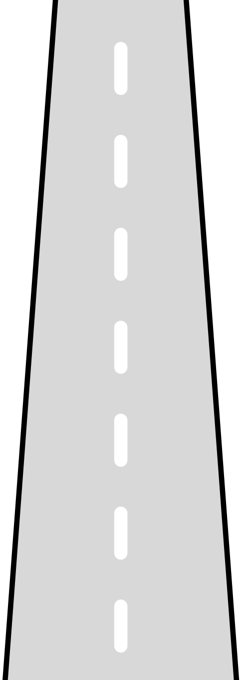
Bit of Complexity Theory

- The class #P
- Parsimonious reductions

Approximation

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

First Algorithm: DNFs



Approximation to the rescue

- Approx. counting: output Z with

$$Z \leqslant \mathsf{count} \leqslant (1+\epsilon)Z.$$

Approximation to the rescue

- Approx. counting: output Z with

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

FPTAS

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

input size

Approximation to the rescue

- Approx. counting: output Z with

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

FPTAS

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

input size

FPRAS

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

Approximation to the rescue

- Approx. counting: output Z with

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

FPTAS

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

input size

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 $\mathbf{poly}(n, 1/\epsilon, \log(1/\delta))$.

Approximation to the rescue

- Approx. counting: output Z with

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

FPTAS

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

FPRAS

input size

- 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 $\mathbf{poly}(n, 1/\epsilon, \log(1/\delta))$.
- Why all ϵ? Why not 100-approx?

Approximation to the rescue

- Approx. counting: output Z with

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

FPTAS

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

input size

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 $\mathsf{poly}(n, 1/\epsilon, \log(1/\delta))$.

- Why all ϵ? Why not 100-approx?
- Approx. counting is all-or-nothing.

Approximation to the rescue

- Approx. counting: output Z with
$$Z \leqslant \text{count} \leqslant (1+\epsilon)Z.$$
FPTAS
- Fully poly-time approx. scheme: above with runtime $\mathsf{poly}(n, 1/\epsilon)$.

input size

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 $\mathsf{poly}(n, 1/\epsilon, \log(1/\delta))$.

- Why all ϵ? Why not 100-approx?
- Approx. counting is all-or-nothing.

Example: #SAT

Suppose A is $f(n)$-approx. alg. Give
$$\phi^{(1)} \wedge \phi^{(2)} \wedge \cdots \wedge \phi^{(t)}$$
with $\phi^{(i)}$ being disjoint copies of ϕ.
$$\sqrt[t]{\text{output}} \approx \#\mathsf{SAT}(\phi).$$
Approx. ratio is $\sqrt[t]{f(nt)}$. Even for $f(n) = 2^{n^{0.99}}$, enough to set $t = \mathsf{poly}(n, 1/\epsilon)$ to get $\sqrt[t]{f(nt)} \leqslant 1+\epsilon$.

Approximation is all-or-nothing

- For any "tensorizable" problem, nothing between $1 + \epsilon$ and exponential.

Approximation is all-or-nothing

- For any "tensorizable" problem, nothing between $1+\epsilon$ and exponential.
- For "self-reducible" probs, $\mathsf{poly}(n)$-approx gives an FPRAS. [Jerrum-Sinclair].
  will see later in the course

Approximate sampling

- Notion of approximation: For dists ν, μ on Ω we use total variation:

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

Approximate sampling

- Notion of approximation: For dists ν, μ on Ω we use total variation:

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

FPAUS

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

Approximate sampling

- Notion of approximation: For dists ν, μ on Ω we use total variation:

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

FPAUS

- Fully poly approx. unif. sampler: output distribution has $d_{TV} \leqslant \delta$ and runtime is $\mathrm{poly}(n, \log(1/\delta))$.
- Note: the $\log$ dependence on δ is similarly "all-or-nothing".

Approximate sampling

- Notion of approximation: For dists ν, μ on Ω we use total variation:

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

FPAUS

- Fully poly approx. unif. sampler: output distribution has $d_{TV} \leqslant \delta$ and runtime is $\mathrm{poly}(n, \log(1/\delta))$.
- Note: the $\log$ dependence on δ is similarly "all-or-nothing".

Theorem [Jerrum-Valiant-Vazirani]

For "self-reducible" problems:

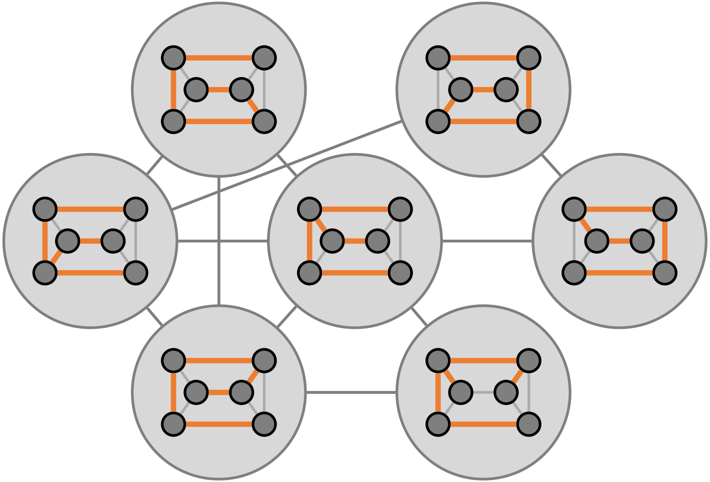
approx counting $\equiv$ approx sampling

$$
\begin{tikzcd}
& \text{(FPRAS)} \\
\text{Exact Counting} \arrow[r] \arrow[d] & \text{Approx Counting} \arrow[dl, "\text{if FPTAS}"'] \arrow[d, leftrightarrow] \\
\text{Exact Sampling} \arrow[r] & \text{Approx Sampling} \\
& \text{(FPAUS)}
\end{tikzcd}
$$

arrows are poly-time reductions

Counting via Markov chains

- Basis of Markov Chain Monte Carlo: Approx Sampler → Approx Counter.

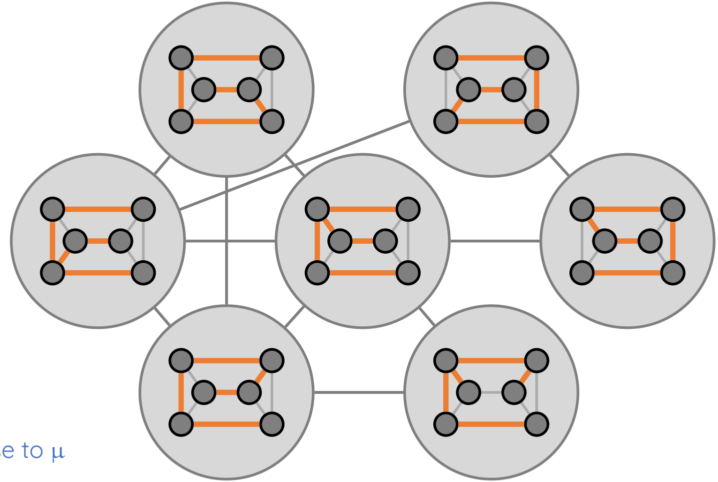


Counting via Markov chains

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

$$x_0 \to x_1 \to \cdots \to x_t$$

hope this is close to μ



What is "Counting and Sampling"?

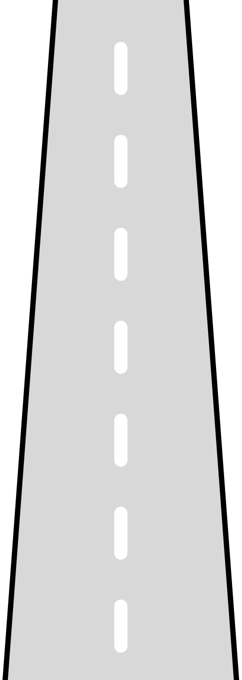
Bit of Complexity Theory

- The class #P
- Parsimonious reductions

Approximation

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

First Algorithm: DNFs



What is “Counting and Sampling”?

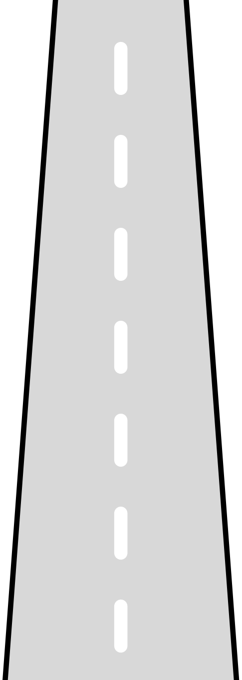
Bit of Complexity Theory

- The class #P
- Parsimonious reductions

Approximation

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

First Algorithm: DNFs



Given DNF formula

$$\phi = (x_1 \wedge \overline{x_2} \wedge x_3) \vee \cdots ,$$

can we approx sample/count satisfying assignments?

Given DNF formula

$$\phi = (x_1 \wedge \overline{x_2} \wedge x_3) \vee \cdots ,$$

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* ϕ **then**
    accept and return x

Given DNF formula

$$\phi = (x_1 \wedge \overline{x_2} \wedge x_3) \vee \cdots ,$$

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* ϕ **then**
    accept and return x

- This is an instance of rejection sampling.

Given DNF formula

$$\phi = (x_1 \wedge \overline{x_2} \wedge x_3) \vee \cdots ,$$

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* ϕ **then**
  - accept and return x

- This is an instance of rejection sampling.

Rejection sampling

We have access to sampler for ν, 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 $\leqslant 1$

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* ϕ **then**
    accept and return x

- This is an instance of rejection sampling.

Rejection sampling

We have access to sampler for ν, 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 $\leqslant 1$

- Output is always $\sim$ normalized μ 🙂

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* ϕ **then**
  - accept and return x

- This is an instance of rejection sampling.

Rejection sampling

We have access to sampler for ν, 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 $\leqslant 1$

- Output is always $\sim$ normalized μ 🙂
- Can take a long time 🙁

Given DNF formula

$$\phi = (x_1 \wedge \overline{x_2} \wedge x_3) \vee \cdots,$$

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* ϕ **then**
  - accept and return x

- This is an instance of rejection sampling.

Rejection sampling

We have access to sampler for ν, 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 $\leqslant 1$

- Output is always $\sim$ normalized μ 🙂
- Can take a long time 🙁
- If μ is normalized, the best c is $\min\{\nu(x)/\mu(x)\}$, and it takes $\simeq \max\{\mu(x)/\nu(x)\}$ iterations.

Given DNF formula

$$\phi = (x_1 \wedge \overline{x_2} \wedge x_3) \vee \cdots ,$$

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* ϕ **then**
    accept and return x

- This is an instance of rejection sampling.

Rejection sampling

We have access to sampler for ν, 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 $\leqslant 1$

- Output is always $\sim$ normalized μ 🙂
- Can take a long time ☹️
- If μ is normalized, the best c is $\min\{\nu(x)/\mu(x)\}$, and it takes $\simeq \max\{\mu(x)/\nu(x)\}$ iterations.
- For $\phi = (x_1 \wedge x_2 \wedge \cdots \wedge x_n)$ it takes 2^n tries on average. ☹️

A better envelope [Karp-Luby]

$\phi = C_1 \vee C_2 \vee \cdots \vee C_m$

A better envelope [Karp-Luby]

$$\phi = C_1 \vee C_2 \vee \cdots \vee C_m$$

- Let $A_i = \{\text{sat assignments of } C_i\}$.

A better envelope [Karp-Luby]

$$\phi = C_1 \vee C_2 \vee \cdots \vee C_m$$

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

A better envelope [Karp-Luby]

$$\phi = C_1 \vee C_2 \vee \cdots \vee C_m$$

- Let $A_i = \{$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 $A_1 \cup \cdots \cup A_m$.

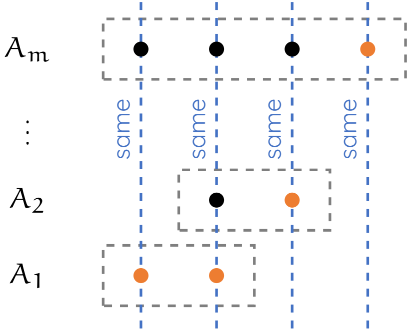
disjoint union

A better envelope [Karp-Luby]

$$\phi = C_1 \vee C_2 \vee \cdots \vee C_m$$

- Let $A_i = \{$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 $A_1 \cup \cdots \cup A_m$.

disjoint union



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**

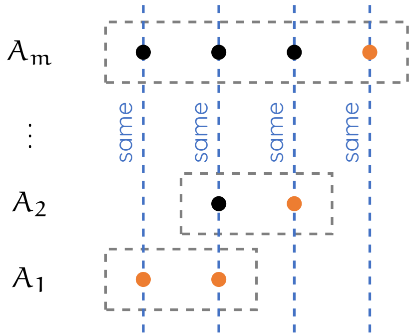
    accept and return x

A better envelope [Karp-Luby]

$$\phi = C_1 \vee C_2 \vee \cdots \vee C_m$$

- Let $A_i = \{$sat assignments of $C_i\}$.
- Want to sample from $A_1 \cup \cdots \cup A_m$.
- Idea: sample from $A_1 \sqcup \cdots \sqcup A_m$ (disjoint union) 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 $\geqslant 1/m$.

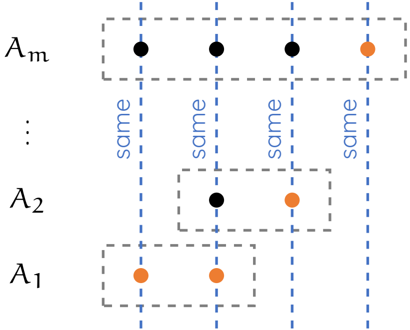
A better envelope [Karp-Luby]

$$\phi = C_1 \vee C_2 \vee \cdots \vee C_m$$

- Let $A_i = \{$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 $A_1 \cup \cdots \cup A_m$.

disjoint union

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 $\geqslant 1/m$.
- On average $\simeq m$ iterations suffice.

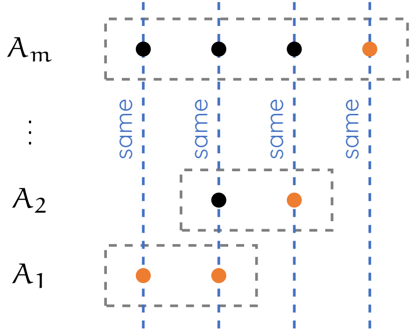
A better envelope [Karp-Luby]

$$\phi = C_1 \vee C_2 \vee \cdots \vee C_m$$

- Let $A_i = \{$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 $A_1 \cup \cdots \cup A_m$.

disjoint union

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 $\geqslant 1/m$.
- On average $\simeq m$ iterations suffice.
- Next lecture: turning this into approx counting.