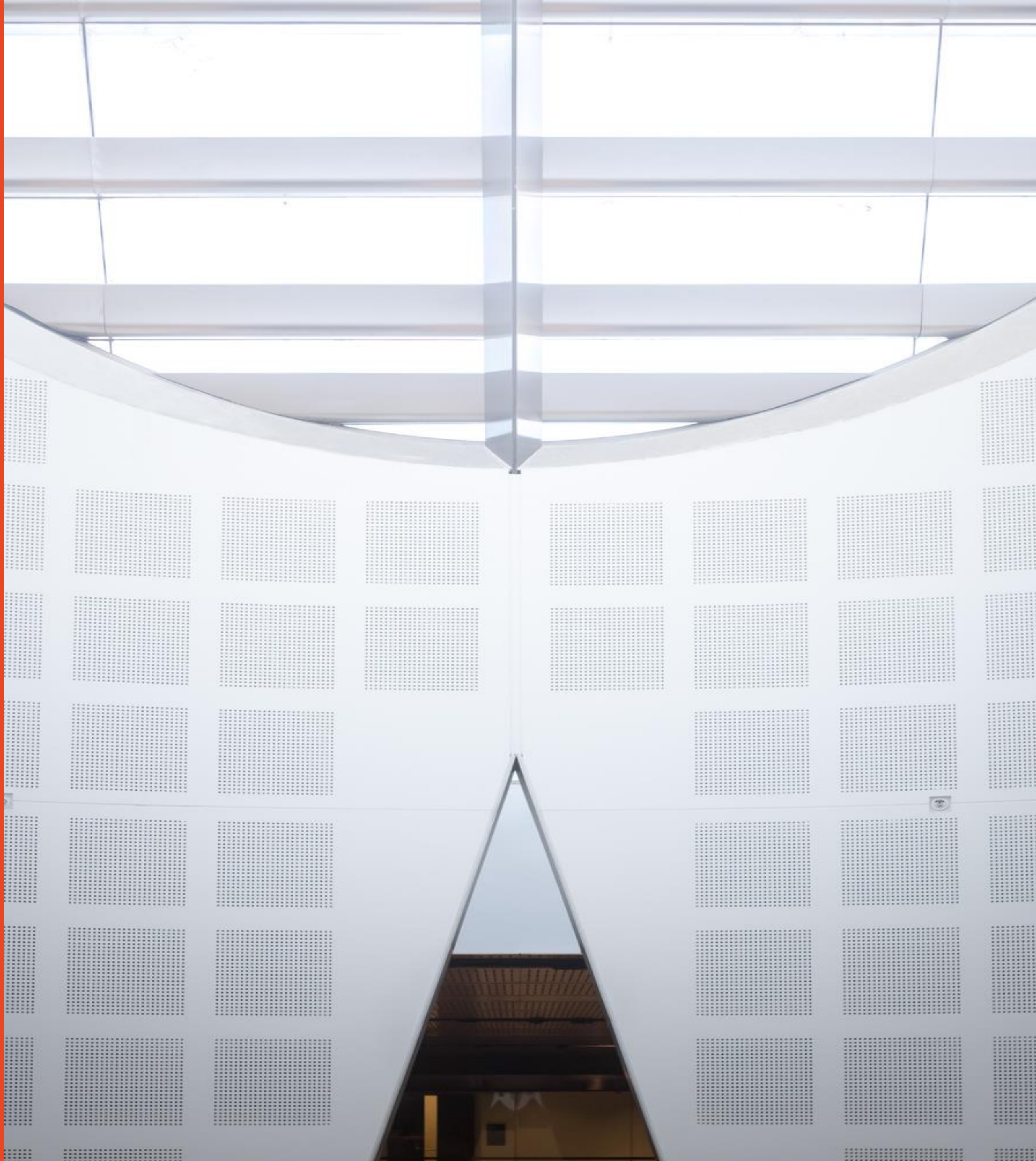
COMPx270: Randomised and Advanced Algorithms
Lecture 4: Derandomisation

Clément Canonne

School of Computer Science



A question

You have a randomised algorithm A which runs in time $T(n)$ and solves task X (say, decision problem) with probability .99. Is there a deterministic algorithm B which solves X and runs in time...

- $O(T(n))$
- $\mathrm{poly}(T(n))$
- $\exp(T(n))$
- No/we don't know

A question

You have a randomised algorithm A which runs in time T(n) and solves task X (say, decision problem) with probability .99. Is there a deterministic algorithm B which solves X and runs in time...

- O(T(n)) ?
- poly(T(n)) ?
- exp(T(n)) ✓
- No/we don't know

An answer?

That's **complicated**. This is what derandomization asks, and there is a lot of work on this: one of the major unsolved question in theoretical computer science.

P v. BPP

decision problems solvable in poly time

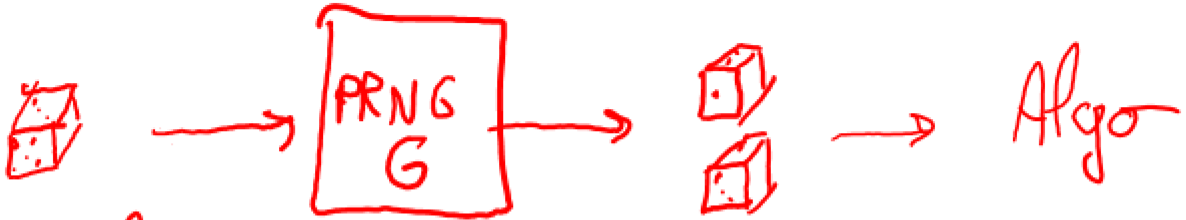
decision problems solvable by a Monte Carlo algorithm in poly time

Let's not stop here though

We know how to derandomize **some** algorithms, and there are **some** general techniques.

Method 1: PRNG 🎲

The goal is to reduce the amount of randomness required, by generating a lot of "good enough" pseudorandom bits: good enough to fool the algorithm.



$G: \{0,1\}^\ell \to \{0,1\}^n \qquad \ell \ll n$

$\forall A \in \mathcal{A}$

class of algorithms

$A(G(U_\ell)) \approx A(U_n)$

n uniformly random bits

Fact: Under plausible assumptions, there exist PRNGs for $\mathcal{A} = \{\text{poly time algos}\}$ with $\ell = O(\log n)$

Method 1: PRNG 🎲

Why is that useful?

- Random bits don't grow on trees!
- Derandomisation (method 2)

Why is this bad?

- Conditional (under assumptions)

Method 2: Brute force 💪

If the algorithm uses a small number of random seeds, check 'em all.

Method 2: Brute force 💪

If the algorithm uses a small number of random seeds, check 'em all.

Require: Input x

for all $r \in \{0,1\}^R$ **do** ← 2^R

  $y \leftarrow A(x;r)$ ▷ Run A on x with randomness r

  if $V(x,y) = 1$ **then** ▷ Verify if y is a good solution

    return y ▷ If so, we are done

A uses R random bits

Details.

- If A runs in time T_A
- If we have a "verifier" V : on input x, and y, check if y is a solution for x in time T_V

then this runs in time

$$2^R (T_A + T_V)$$

- There is at least one random string r^* for which $A(x; r^*)$ is correct (i.e., $\Pr_r[A(x;r) \text{ is correct}] \geq \frac{1}{2^R}$)

r^* can depend on x!

Method 2: Brute force 💪

What if verifying is hard?

Method 2: Brute force 💪

What if verifying is hard?

- Majority vote!
- Median trick!

If A is correct w.p. $\geq 51\%$

this means $\geq 51\%$ of the 2^R random seeds will lead to the right yes/no answer

→ no verifier V needed

for decision problem

$I(x)$ "good interval"

$$\Pr_r[A(x;r) \in I(x)] \geq 51\%$$

Take median of the 2^R outputs

Works for a lot of problems (real-valued answers)

Method 2: Brute force 💪

What if the algorithm does not use a small number of random bits?

Method 2: Brute force 💪

What if the algorithm does not use a small number of random bits?

Well, these PRNGs can come in handy…

Remember fact:

$\ell = O(\log n)$ seed length to fool poly-time algos "under plausible assumptions"

→ poly-time randomized MC (BPP) algo uses $R = \text{poly}(n)$ random bits

→ $\ell = O(\log R) = O(\log n^c) = O(\log n)$ seed length suffice

↑ "R'"

→ $2^{R'} = 2^{O(\log n)} = \text{poly}(n)$

Method 2: Brute force 💪

What if the algorithm does not use a small number of random bits?

Or (sometimes) we can reduce the randomness by carefully looking at the proof.

Method 2: Brute force 💪 via pairwise independence

Derandomizing Max-Cut

MAX-CUT: Given an (undirected) graph $G = (V, E)$ on n vertices and m edges, output a cut (A, B) (partition of V) *maximising* the number $c(A, B)$ of edges between A and B.

(It's NP-Hard)

Method 2: Brute force 💪 via pairwise independence

Derandomizing Max-Cut

MAX-CUT: Given an (undirected) graph $G = (V, E)$ on n vertices and m edges, output a cut (A, B) (partition of V) *maximising* the number $c(A, B)$ of edges between A and B.

But we can get a ½-approximation!

(Can't get better than 0.95 unless P=NP)

Method 2: Brute force 💪 via pairwise independence

1: $(A, B) \leftarrow (\emptyset, \emptyset)$
2: **for all** $v \in V$ **do**
3: $\quad X_v \leftarrow \mathrm{Bern}(1/2)$ ▷ Independent of previous choices
4: $\quad$ **if** $X_v = 1$ **then** add v to A
5: $\quad$ **else** add v to B
6: **return** (A, B)

Bern = Bernoulli

$X \sim \mathrm{Bern}(p)$ means

$X = \begin{cases} 0 & \text{w.p. } 1-p \\ 1 & \text{w.p. } p \end{cases}$

$\mathrm{Bern}(\frac{1}{2})$ = fair coin

Method 2: Brute force 💪 via pairwise independence

Theorem.

$$\mathbb{E}[c(A,B)] \geq \frac{1}{2} m \geq \frac{1}{2}\,\mathrm{OPT}(G)\,.$$

Proof. For $e \in E$, $X_e = \begin{cases} 1 & \text{if edge is cut} \\ 0 & \text{o/w} \end{cases}$

$$\mathbb{E}[X_e] = \Pr[e \text{ is cut}] = \Pr[u \in A, v \in B \text{ or } u \in B, v \in A]$$

$$= \Pr[u \in A, v \in B] + \Pr[u \in B, v \in A]$$

$$= \Pr[u \in A] \cdot \Pr[v \in B] + \circlearrowleft \quad (\text{by indep}^{ce})$$

$$= \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$$

$$\mathbb{E}[c(A,B)] = \mathbb{E}\Big[\sum_{e \in E} X_e\Big] = \sum_{e \in E} \mathbb{E}[X_e] = \frac{m}{2}.$$

Method 2: Brute force 💪 via pairwise independence

Theorem. This can be derandomised. (efficiently)

Method 2: Brute force 💪 via pairwise independence

Theorem. This can be derandomised.

$(A, B) \leftarrow (\emptyset, \emptyset)$
for all $v \in V$ **do**
  $X_v \leftarrow \mathrm{Bern}(1/2)$ ▷ Independent of previous choices
  if $X_v = 1$ **then** add v to A
  else add v to B
return (A, B)

Method 2: Pairwise independent hash functions

Definition 22.1. A family of functions $\mathcal{H} \subseteq \{h\colon \mathcal{X} \to \mathcal{Y}\}$ is a *family of pairwise independent hash functions*, or a *strongly universal hash family*, if, for every $x, x' \in \mathcal{X}$ with $x \neq x'$ and every $y, y' \in \mathcal{Y}$,

$$\Pr_{h \sim \mathcal{H}}\left[\, h(x) = y, h(x') = y' \,\right] = \frac{1}{|\mathcal{Y}|^2}$$

where the probability is over the uniformly random choice of $h \in \mathcal{H}$.

Want this to be small

$\log_2 |\mathcal{H}|$ bits of randomness

Here: $\mathcal{X} = V \; (= \{1, 2, \dots, n\})$

$\mathcal{Y} = \{0, 1\}$

Random choices:

Hash function: Pick h at random

Method 2: Pairwise independent hash functions

Fact. Small families of pairwise independent hash functions exist.

Method 2: Pairwise independent hash functions

Fact. Small families of pairwise independent hash functions exist.

For $\mathcal{X} = \{1, 2, \ldots, n\}$

$\mathcal{Y} = \{0, 1\}$

There is a family $\mathcal{H}$ of pairwise hash functions from $\{1, 2, \ldots, n\}$ to $\{0, 1\}$ with

$$|\mathcal{H}| = n$$

and so $\log_2 |\mathcal{H}| = \log \lceil n+1 \rceil = O(\log n)$

Method 2: Pairwise independent hash functions

Fact. Small families of pairwise independent hash functions exist.

Proof of derandomization claim.

Pick h from $\mathcal{H}$ u.a.r

1: $(A, B) \leftarrow (\emptyset, \emptyset)$
2: **for all** $v \in V$ **do**
3: $\quad X_v \leftarrow$ Bern$\cancel{(1/2)}$ $h(v)$ ▷ Independent of previous choices
4: $\quad$ **if** $X_v = 1$ **then** add v to A
5: $\quad$ **else** add v to B
6: **return** (A, B)

① This algorithm still gives $\mathbb{E}$ cut $\geq \frac{1}{2}$OPT (exactly same proof!)

② It uses $\log_2 |\mathcal{H}| = \lceil \log_2 (n+1) \rceil$ random bits

③ Can verify "easily" if a cut has value $\geq \frac{m}{2}$ $\quad (O(m+n))$

→ Time $\quad O(n(m+n))$

④ Is there a good solution? $\quad \Pr[\text{cut}(A,B) \geq \frac{m}{2}] \overset{?}{>} 0$

(Important) Fact. If $\mathbb{E}[X]$ exists, then $\Pr[X \geq \mathbb{E}[X]\,] > 0$.

Proof. (discrete case)

$$\underbrace{\mathbb{E}[X]}_{\mu} = \mathbb{E}[X(\mathbb{1}_{X<\mu} + \mathbb{1}_{X\geq\mu})] = \mathbb{E}[X\mathbb{1}_{X<\mu}] + \mathbb{E}[X\mathbb{1}_{X\geq\mu}]$$

By contradiction, assume $\Pr[X\geq\mu]=0$

$$\mu = \mathbb{E}[X\mathbb{1}_{X<\mu}] < \mathbb{E}[\mu\mathbb{1}_{X<\mu}] = \mu\underbrace{\Pr[X<\mu]}_{=1}\;{}^{=\,0}\; = \mu$$

nope

Method 2: Brute force 💪 via pairwise independence

Theorem. There exists a deterministic ½-approximation algorithm for Max-CUT which runs in time O(~~m~~n(m+n)).

Method 3: The Method of Conditional Expectations

Idea: sequentially do the greedy choice. Sometimes it works!

Method 3: The Method of Conditional Expectations

Idea: sequentially do the greedy choice. Sometimes it works!

$(A, B) \leftarrow (\emptyset, \emptyset)$

for $v = 1, 2, \dots, n$ ~~**for all** $v \in V$~~ **do**

choose the best greedy choice to "preserve future expectation"

  $X_v \leftarrow$ ~~Bern(1/2)~~ ▷ ~~Independent of previous choices~~

  if $X_v = 1$ **then** add v to A

  else add v to B

return (A, B)

Details.

want to choose $X_1 \in \{0,1\}$ such that

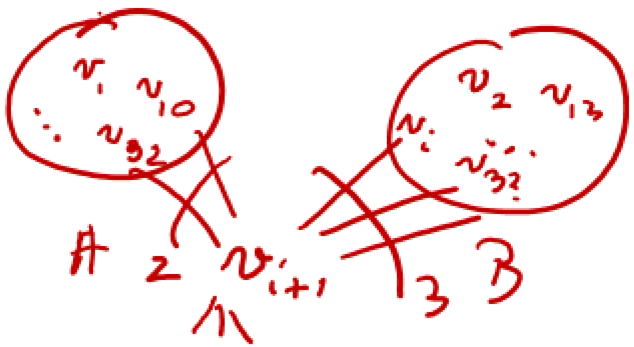
$$\frac{m}{2} = \mathbb{E}[c(A,B)] \le \mathbb{E}[c(A,B) \mid X_1]$$
$$\le \mathbb{E}[c(A,B) \mid X_1, X_2]$$
$$\vdots$$
$$\le \mathbb{E}[c(A,B) \mid X_1, \dots, X_n] = c(A,B)$$

At step $i+1$: I have chosen $X_1, \dots, X_i$

$$\mathbb{E}[c(A,B) \mid X_1, \dots, X_{i+1}] = \frac{1}{2} \cdot \mathbb{E}[c(A,B) \mid X_1, \dots, X_i, X_{i+1} = 0] + \frac{1}{2} \mathbb{E}[c(A,B) \mid X_1, \dots, X_i, X_{i+1} = 1]$$

$$\le \max\Big(\underbrace{\mathbb{E}[c(A,B) \mid X_1, \dots, X_i, X_{i+1} = 0]}_{①}, \underbrace{\mathbb{E}[c(A,B) \mid X_1, \dots, X_i, X_{i+1} = 1]}_{②} \Big)$$

Can I efficiently figure out which of ① and ② is biggest?



$2 < 3$

Go for Ⓐ

Method 3: The Method of Conditional Expectations

Theorem. There exists a deterministic ½-approximation algorithm for Max-CUT which runs in time $O(mn)$.

Derandomisation: summary

- PRNG
- Brute-Force
- Pairwise (k-wise) independence
- Method of Conditional Expectations

(there is more!)

Bonus: The Probabilistic Method

"We can prove things exist without knowing how to build them."

(also can be derandomised, sometimes)