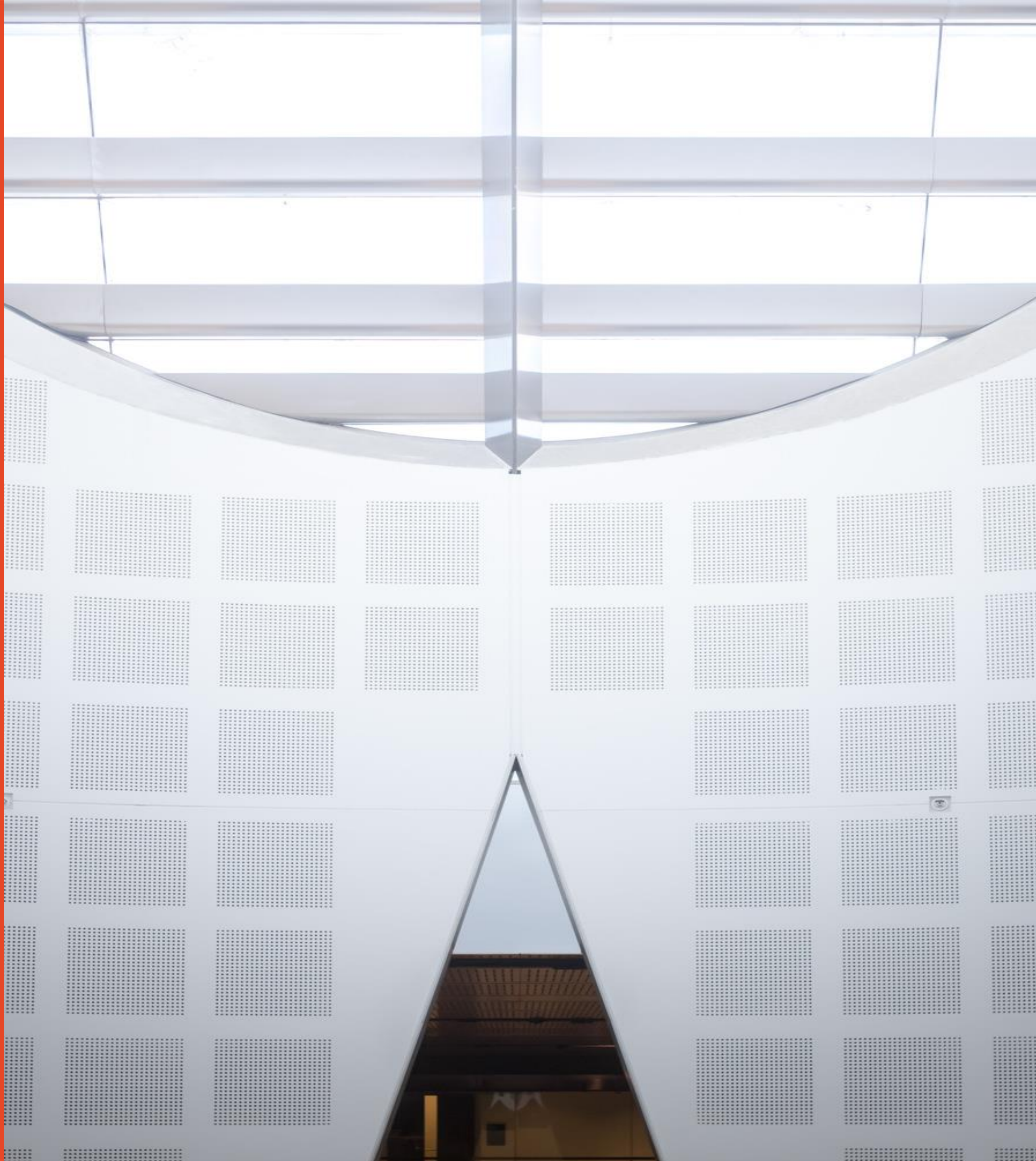
COMPx270: Randomised and Advanced Algorithms
Lecture 4: Derandomisation

Clément Canonne

School of Computer Science

THE UNIVERSITY OF SYDNEY



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...

- $O(T(n))$?
- $\mathrm{poly}(T(n))$?
- $\exp(T(n))$ ✓
- 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

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.

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** ▷

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

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

$\quad\quad$ **return** y ▷ If so, we are done

Details.

Method 2: Brute force 💪

What if verifying is hard?

Method 2: Brute force 💪

What if verifying is hard?

- Majority vote!
- Median trick!

Method 2: Brute force 💪

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

Method 2: Brute force 💪

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

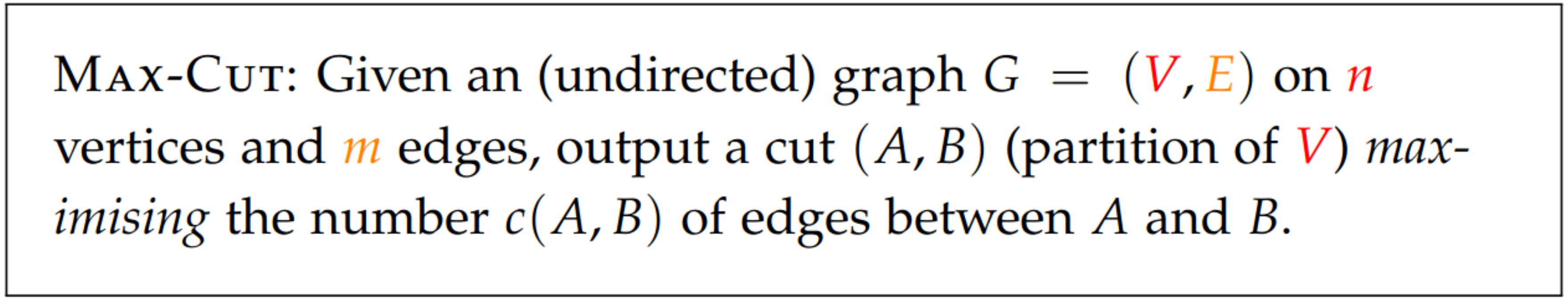
Derandomizing Max-Cut

> 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!

Method 2: Brute force 💪 via pairwise independence

$(A, B) \leftarrow (\emptyset, \emptyset)$
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: Brute force 💪 via pairwise independence

Theorem.

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

Proof.

Method 2: Brute force 💪 via pairwise independence

Theorem. This can be derandomised.

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 $h \in \mathcal{H}$.

Method 2: Pairwise independent hash functions

Fact. Small families of pairwise independent hash functions exist.

Method 2: Pairwise independent hash functions

Method 2: Pairwise independent hash functions

Fact. Small families of pairwise independent hash functions exist.

Proof of derandomization claim.

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

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

Proof.

Method 2: Brute force 💪 via pairwise independence

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

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 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)

Details.

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)