COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**). The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

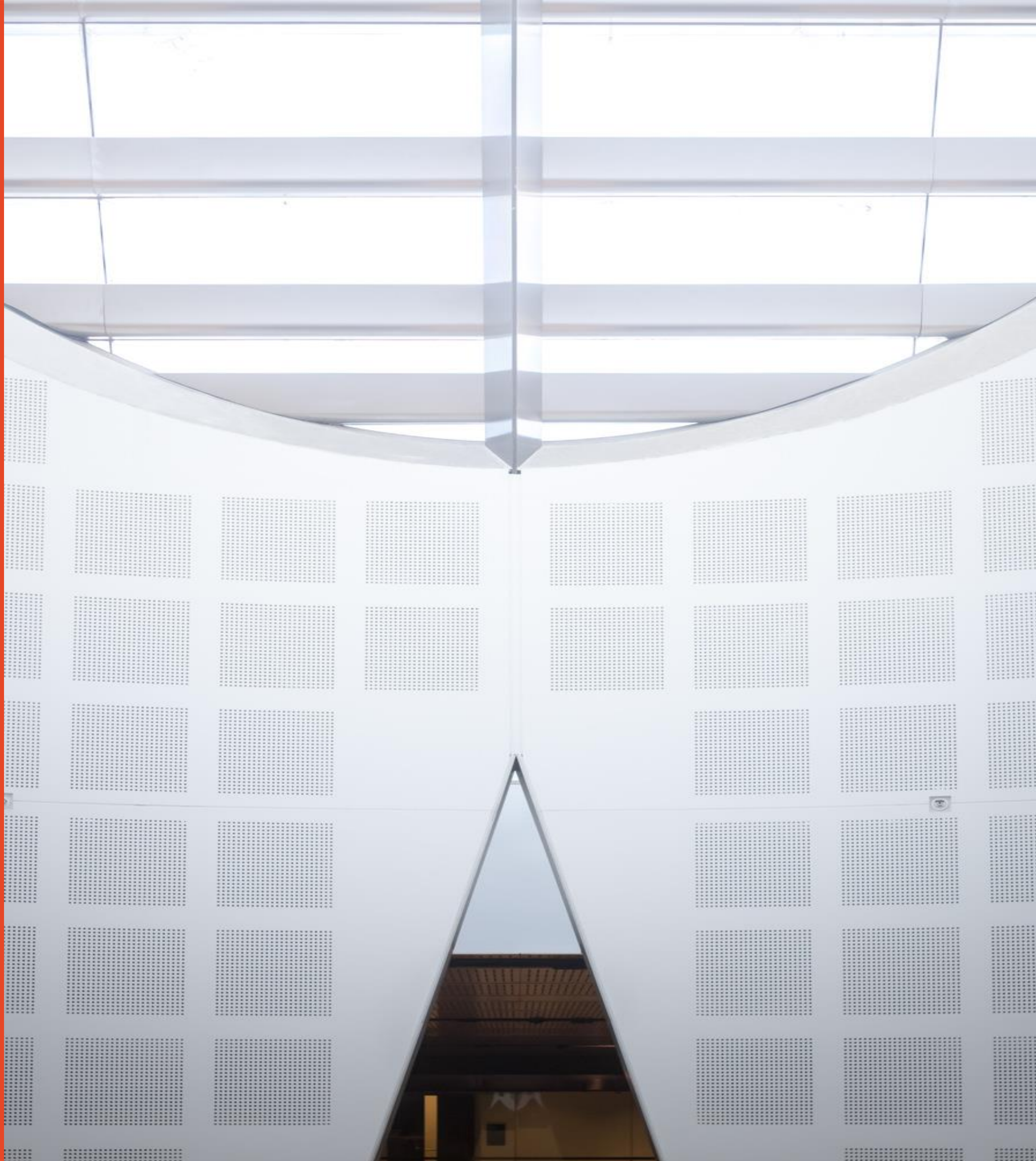
Do not remove this notice.

COMPx270: Randomised and Advanced Algorithms
Lecture 4: Derandomisation

Clément Canonne

School of Computer Science

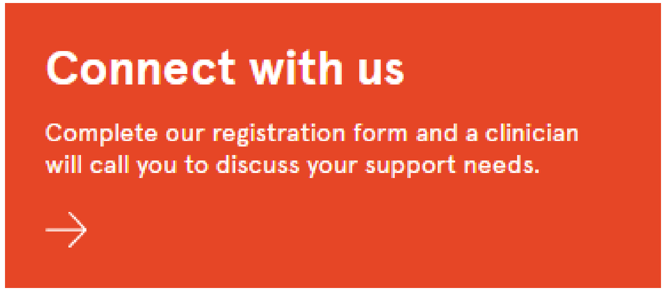
THE UNIVERSITY OF SYDNEY



Connect with Student Wellbeing

Everything you need to know about student life, wellbeing and support can be found on the landing page:

Students can self-refer via the webpage by clicking *"Connect with us"*:



Phone: 02 7255 1562 / 8627 8433

Address: Level 5, Jane Foss Russell Building, G02

An announcement (or three)

- HW1, Problem 5: β should be 2β (updated assignment tonight)
- Office Hours (OH) this **Friday, 3:30-5pm, J12 302 + Zoom**
- Simple extensions (you have them by default!)

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

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

↑

class of decision problems w/ a poly-time Monte Carlo Algo

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.

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

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

$TV(A(G(U_\ell)), A(U_n)) \leq \varepsilon$

statistical distance

FACT Under "plausible assumption", PRNGs for $\mathcal{A} = \{$poly time algos$\}$ exist for $\ell = O(\log n)$

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 ∈ {0,1}^R do
    y ← 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
```

T_A

T_V

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

Details.

Method 2: Brute force 💪

What if verifying is hard?

① $\Pr[A \text{ is correct}] > 0$

② I can verify. (implies $\geq \frac{1}{2^R}$)

Method 2: Brute force 💪

$A(x) \in \{\text{yes}, \text{no}\}$

What if verifying is hard?

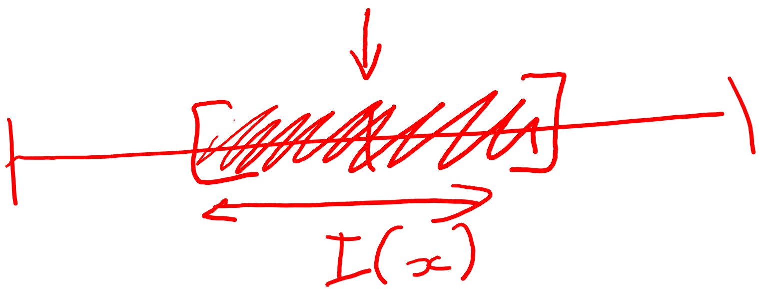
$\Pr[A \text{ is correct}] \geq \frac{2}{3}$

$A(x; R)$

- Majority vote!
- Median trick!

"good"

$\Pr[A(x) \in I(x)] \geq \frac{2}{3}$



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…

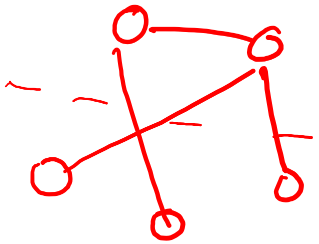
Method 2: Brute force 💪

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

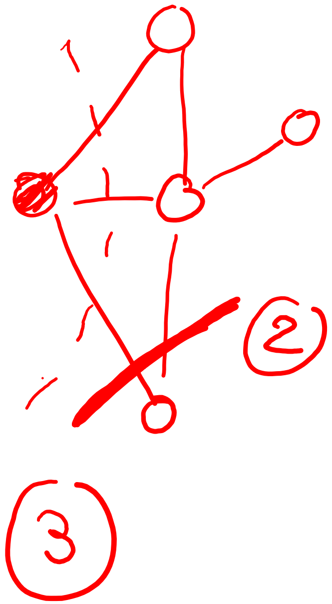
Derandomizing Max-Cut



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

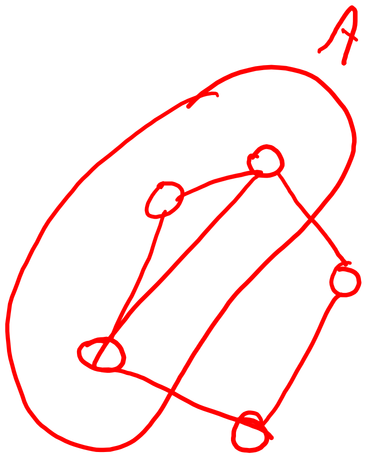
> MAX-CUT: Given an (undirected) graph $G = (V, E)$ on n vertices and m edges, output a cut (A, B) (partition of V) *maximising* the number $c(A, B)$ of edges between A and B.

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



$c(A,B) = 2$

Theorem.

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

Proof. For $e \in E$, $\mathbb{1}_{e \in C}$ indicates if $e \in$ cut

$e = (u,v)$

$$\mathbb{E}[c(A,B)] = \sum_{e \in E} \mathbb{E}[\mathbb{1}_{e \in C}] = \sum_{e \in E} \Pr[e \text{ is "cut"}] = \sum_{e=(u,v)} \Pr[u \in A, v \in B \text{ or } u \in B, v \in A]$$

$$= \sum_{e=(u,v)} \left(\Pr[u \in A, v \in B] + \Pr[u \in B, v \in A] \right)$$

$\Pr[u \in A, v \in B] = \Pr[u \in A] \Pr[v \in B] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$

$$= |E| \left(\frac{1}{4} + \frac{1}{4} \right) = \frac{m}{2}$$

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**

  $X_v \leftarrow \text{Bern}(1/2)$ ▷ Independent of previous choices

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

  else add v to B

return (A, B)

$X_v = h(v)$

Overkill!

Need: $\forall\, u \neq v$; X_v, X_u indep't

Method 2: Pairwise independent hash functions

$|\{h: \mathcal{X} \to \mathcal{Y}\}| = |\mathcal{Y}|^{|\mathcal{X}|}$

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

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.

There exist $\mathcal{H} \subseteq \{ \{1, \ldots, n\} \to \{0,1\} \}$ strongly u. hash family

s.t. $\log |\mathcal{H}| = O(\log n)$

Method 2: Pairwise independent hash functions

logn + O(1)

Fact. Small families of pairwise independent hash functions exist.

Proof of derandomization claim.

Pick $h \sim \mathcal{H}$ u.a.r. (O(log n) bits)

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

$$\mathbb{E}[c(A,B)] = \sum_{\substack{e \in E \\ e=(u,v)}} \left(\Pr[u \in A, v \in B] + \Pr[u \in B, v \in A] \right)$$

$$= \sum_{e=(u,v)} \left(\underbrace{\Pr[h(u)=1, h(v)=0]}_{\frac{1}{4} \text{ by pairwise ind}} + \underbrace{\cdots}_{\frac{1}{4}} \right)$$

$$= \frac{m}{2}$$

Go over every $h \in \mathcal{H}$ → $O(|\mathcal{H}|) = O(n)$ time.
For each, $O(m)$ time to check $c(A,B)$

(Important) Fact. If E[X] exists, then Pr[X ≥ E[X]] > 0.

Assume $= 0$ by contradiction

Proof.

$\mu = \mathbb{E}X$

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

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

$< \mathbb{E}[\mu\mathbb{1}_{X<\mu}] + \_\_\_\_$

$\leq \mu \underbrace{\Pr[X<\mu]}_{\leq 1} + \underbrace{\_\_\_\_}_{= 0}$

Method 2: Brute force 💪 via pairwise independence

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

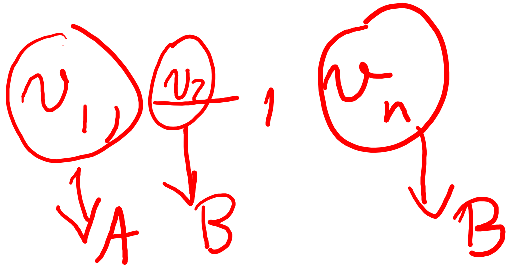
Method 3: The Method of Conditional Expectations

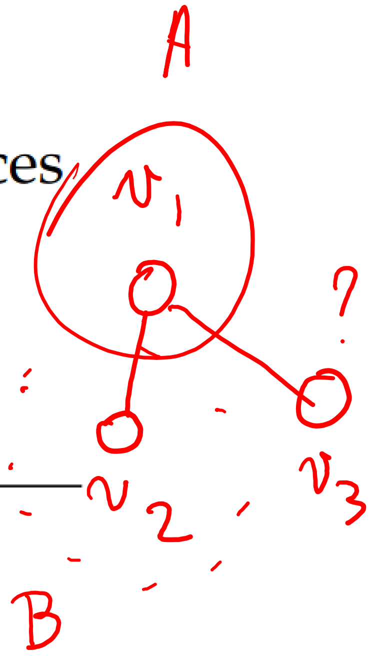
Idea: sequentially do the greedy choice. Sometimes it works!

Method 3: The Method of Conditional Expectations

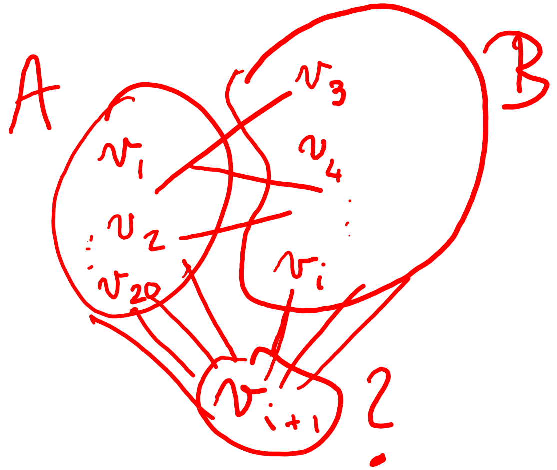
Idea: sequentially do the greedy choice. Sometimes it works!

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





Details.



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

Want $\mathbb{E}[c(A,B) \mid X_1, \dots, X_i] \leq \mathbb{E}[c(A,B) \mid X_1, \dots, X_i, X_{i+1}]$

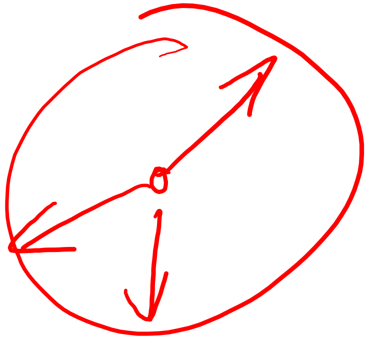
Method 3: The Method of Conditional Expectations

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

Can do better!

(Aside)

Goemans - Williamson



0.878-approx.

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)