5CCS2FC2: Foundations of Computing II

P vs NP: The Million Dollar Question

Week 3

Dr Christopher Hampson

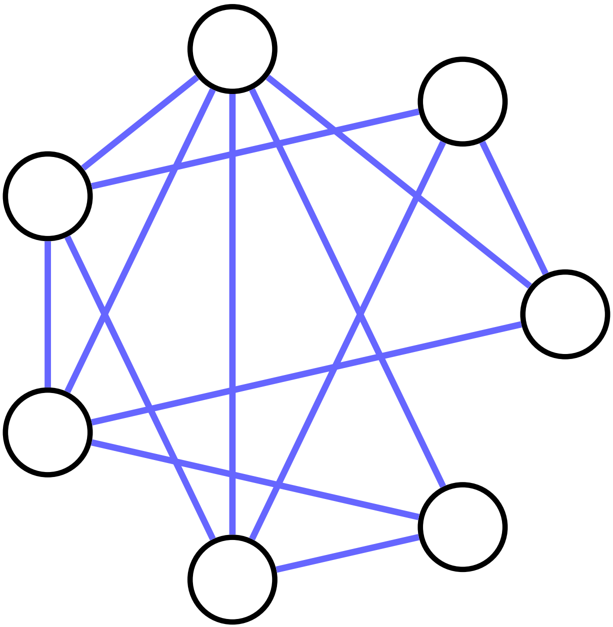
Department of Informatics

King's College London

Warm-up : Colour the Graph

- What is the **minimum number** of colours required to colour the graph such that no two **adjacent vertices** share are the same colour.

(note that the graph is *not planar*)



A Million Dollar Problem

A Million Dollar Problem

"The importance of the **P vs NP** *question stems from the successful theories of* **NP-completeness** *and complexity-based cryptography, as well as the potentially stunning practical consequences of a constructive proof of P = NP.*

Although a practical algorithm for solving an NP-complete problem (showing P=NP) would have devastating consequences for cryptography, it would also have stunning practical consequences of a more positive nature, and not just because of the **efficient solutions** *to the many NP-hard problems important to industry.*

Stephen Cook, 2000

`http://www.claymath.org/millennium-problems/p-vs-np-problem`

Asymptotic Notation

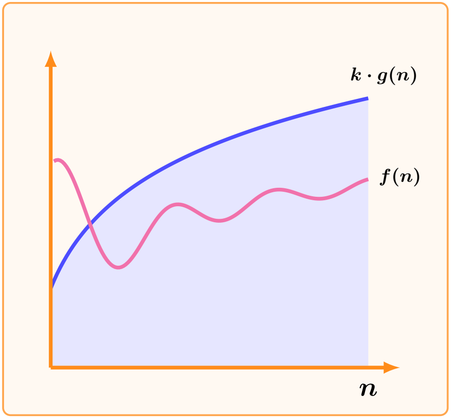
Asymptotic Notation – Big Oh

- **Big Oh Notation (upper bounds)**
  - Let $f(n)$ and $g(n)$ be any real-valued function. We say that g **eventually dominates** f if there is some constant $k > 0$ such that

$$f(n) \leq k \cdot g(n) \qquad \text{for all `large' } n$$

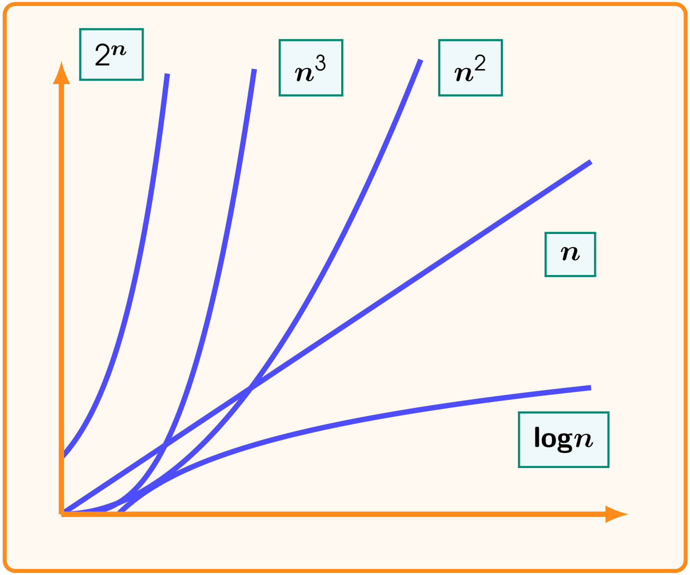
  - We say that $f(n)$ belongs to the class $O(g(n))$, read **'big oh of g'**, if $f(n)$ is eventually dominated by $g(n)$.

$$f(n) = O(g(n))$$
or
$$f(n) \in O(g(n))$$



Asymptotic Notation – Big Oh

- **Big Oh Notation** (upper bounds)



$$O(1) \subsetneq O(\log n) \subsetneq O(\sqrt{n}) \subsetneq O(n) \subsetneq O(n^2) \subsetneq O(n^3) \subsetneq \ldots \subsetneq O(2^n)$$

Asymptotic Notation – Big Oh

Quick Guide to Big Oh

- Disregard any constant factors,
- Disregard anything bounded by a constant,
- Identify the largest term,
- For logarithms, disregard the base

(since $\log_b x = \log_2 x / \log_2 b$)

Example

$$T(n) = 5^{27}\sqrt{\frac{n}{\pi}} + n^4 \sin(n) + \log_{2018}(6n) + 999! = O(n^4)$$

Complexity Classes P and NP

Complexity Classes P and NP

- **Polynomial Time Problems**

  - A decision problem X is said to be decidable/solvable in **polynomial time** if there is a **deterministic Turing Machine** $\mathcal{M}$ such that:

    (i) $\mathcal{M}$ **accepts** X,

    (ii) $T(n) \in O(n^k)$ is dominated by a **polynomial function**, where

$$T(n) = \begin{cases} \text{number of steps required to} \\ \text{terminate on input of length } n \end{cases}$$

  - The **complexity class P** is the class of all *problems* that are decidable in polynomial time

$$\mathbf{P} \ = \ \{ \text{ all problems decidable in polynomial time } \}$$

Complexity Classes P and NP

- **Non-deterministic Polynomial Time Problems**

  - The class of **non-deterministic polynomial time** problems is defined similarly but replacing $\mathcal{M}$ with a non-deterministic TM, for which

$$T(n) = \left\{ \begin{array}{c} \text{number of steps required to terminate on input} \\ \text{of length } n \text{ for } \textit{some} \text{ possible computation} \end{array} \right.$$

$$\textbf{NP} \;=\; \left\{ \begin{array}{c} \text{all problems decidable in} \\ \text{non-deterministic polynomial time} \end{array} \right\}$$

Problems that belong to **NP** are those for which we can **verify** the solution in polynomial time — you only need to show me a single computation that accepts the input. However, to find the solution may require an **exhaustive search** of all possible computations.

Complexity Class PSpace

- **Polynomial Space Problems**

  - A decision problem X is said to be decidable/solvable in **polynomial space** if there is a **deterministic Turing Machine** $\mathcal{M}$ such that:

    (i) $\mathcal{M}$ **accepts** X,

    (ii) $S(n) \in O(n^k)$ is dominated by a **polynomial function**, where

$$S(n) = \left\{ \begin{array}{c} \text{amount of } \textit{tape} \text{ used for an} \\ \text{input of length } n \end{array} \right.$$

  - The **complexity class P** is the class of all *problems* that are decidable in polynomial time

$$\textbf{PSpace} \;=\; \{ \text{ all problems decidable in polynomial space } \}$$

What we know (and don't know)

Things we know

Things we don't (yet) know

$\mathsf{P} = \mathsf{NP}$

$\mathsf{NP} \subseteq \mathsf{PSpace}$

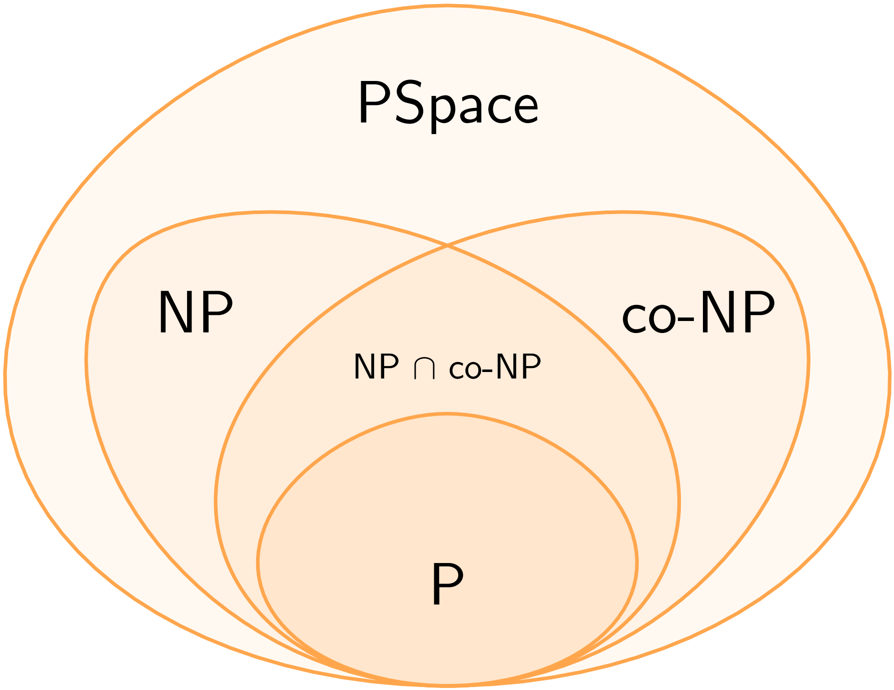
$\mathsf{P} = \mathsf{coP}$

$\mathsf{P} \subseteq \mathsf{NP}$

$\mathsf{P} = \mathsf{NP} \cap \mathsf{coNP}$

$\mathsf{NP} \subseteq \mathsf{coNP}$

Complexity Hierarchy



Constructing NP Algorithms

(Some Examples)

The Boolean Satisfiability Problem

The Boolean Satisfiability Problem SAT

Input) A propositional formula F

Output) **True** if and only if F is *satisfiable*

Satisfiable

so output True

P	Q	R	$(P \vee \neg R) \rightarrow \neg(\neg Q \vee R)$
True	True	True	False
True	True	False	True
True	False	True	False
True	False	False	False
False	True	True	True
False	True	False	True
False	False	True	True
False	False	False	False

The Boolean Satisfiability Problem

Theorem The Boolean Satisfiability Problem **SAT** belongs to the class **NP**.

(*i.e.* there is a non-deterministic algorithm for SAT that runs in polynomial time)

Proof:

Step 1) Given a propositional formula F, we can decide whether F is **satisfiable** by computing its **truth table**.

However the truth table contains 2^n rows – **NOT** polynomial!

Step 2) However, a non-deterministic algorithm can evaluate each row in a separate **parallel processor**, each of which takes at most **polynomial time**.

Q.E.D.

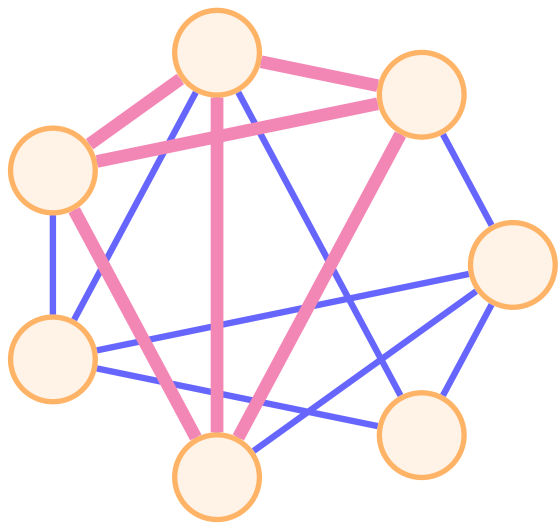
The Clique Problem

The Clique Problem CLIQUE

Input) An undirected graph $G = (V, E)$ and integer $k > 2$

Output) **True** if and only if G contains a clique of size k

Has clique of size 4



The Clique Problem

Theorem The Clique Problem **CLIQUE** belongs to the class **NP**.

Proof:

Step 1) Given an undirected graph $G = (V, E)$ and integer $k > 2$, we can decide whether G contains a clique of size k by checking every subset of vertices of size k.

> However there are $\sim n^n$ possible subsets – **NOT** polynomial!

Step 2) However, a non-deterministic algorithm can check every possible subset of vertices in **parallel**, each of which takes at most **polynomial time**.

Q.E.D.

Polynomial Reductions

(Some Examples)

Polynomial Reductions

- **Polynomial Reduction**

  - A polynomial reduction from a problem A to a problem B is a function $\boxed{f : \Sigma^* \to \Sigma^*}$ computable in polynomial-time, that maps instances A to instances of B such that

$$\boxed{w \in A \iff f(w) \in B}$$

For mapping reductions we did not care about the time taken to compute the function f since we were not concerned about efficiency, since we were only interested in whether or not a problem was decidable.

Polynomial Reductions

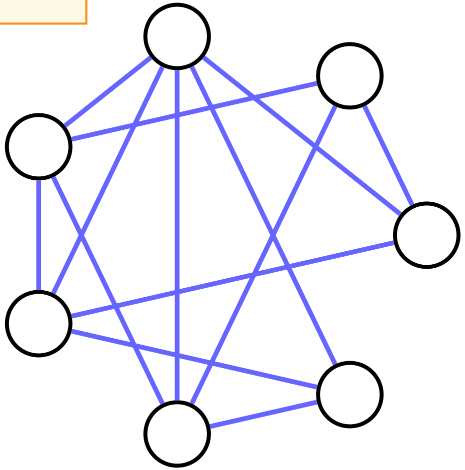
The Graph Colouring Problem COLOURING

Input) An undirected graph $G = (V, E)$ and set of colours C

Output) **True** if and only if V can be coloured so that adjacent verticies are different colours

Colours

$G = \{\mathsf{B}, \mathsf{G}, \mathsf{R}\}$



Polynomial Reductions

Theorem The Graph Colouring problem is polynomially reducible to the Boolean Satisfiability Problem.
i.e. **COLOURING** $\leq_p$ **SAT**.

Proof:

Step 1) Let $G = (V, E)$ be an **undirected graph** and $C = \{$**B**, **G**, **R**$\}$ be any **set of colours** (we are using three here for illustration)

Step 2) For each vertex $v \in V$ and each colour $i \in C$ designate a propositional variable $P_{v,i}$ that says

$$P_{v,i} \;=\; \text{vertex } v \text{ can be coloured with } i.$$

Polynomial Reductions

Step 3) We can write down a **set of formulas** F_G that say that the graph can be coloured with only colours from C,

- Every vertex must be coloured with **some colour**

$$(P_{v,\mathsf{B}} \lor P_{v,\mathsf{G}} \lor P_{v,\mathsf{R}}) \qquad \text{for all } v \in V$$

- No vertex can be coloured with **more than one colour**

$$\neg(P_{v,\mathsf{B}} \land P_{v,\mathsf{G}}) \land \neg(P_{v,\mathsf{B}} \land P_{v,\mathsf{R}}) \land \neg(P_{v,\mathsf{G}} \land P_{v,\mathsf{R}}) \qquad \text{for all } v \in V$$

- **Adjacent vertices** should be different colours

$$\neg(P_{v,\mathsf{B}} \land P_{u,\mathsf{B}}) \land \neg(P_{v,\mathsf{G}} \land P_{u,\mathsf{G}}) \land \neg(P_{v,\mathsf{R}} \land P_{v,\mathsf{R}}) \qquad \text{for all } (u, v) \in E$$

Polynomial Reductions

Step 3) This set of formulas F_G is **satisfiable** if and only if the graph G can be coloured with k colours

$$\boxed{G \in \textsf{COLOURING} \iff F_G \in \textsf{SAT}}$$

(this is a polynomial reduction from COLOURING to SAT)

Q.E.D.

Polynomial Reductions

Theorem The Boolean Satisfiability problem is polynomially reducible to the Clique finding problem. *i.e.* **SAT** $\leq_p$ **CLIQUE**

Proof: Given a formula F with k clauses, we want to construct a graph G_F such that F is satisfiable if and only if G_F has a k-clique.

Step 1) Let $G_F = (V, E)$ where

$$V = \{L^i : L \text{ is a literal appearing in the } i\text{th clause of } F\}$$

Step 2) Connect each vertex to all literals appearing in **different** clauses **UNLESS** they are the negation of the literal

$$(L_1^i, L_2^j) \in E \quad \Longleftrightarrow \quad i \neq j \text{ and } L_1 \not\equiv \neg L_2$$

Step 3) Note the following two observations:

Obv 1) Any clique of size k must contain a **literal from each clause**

(since literals in the same clause are not connected with an edge)

Obv 2) A clique does not contain a **literal** and its **negation**.

(since literals and their negations are not connected with an edge)

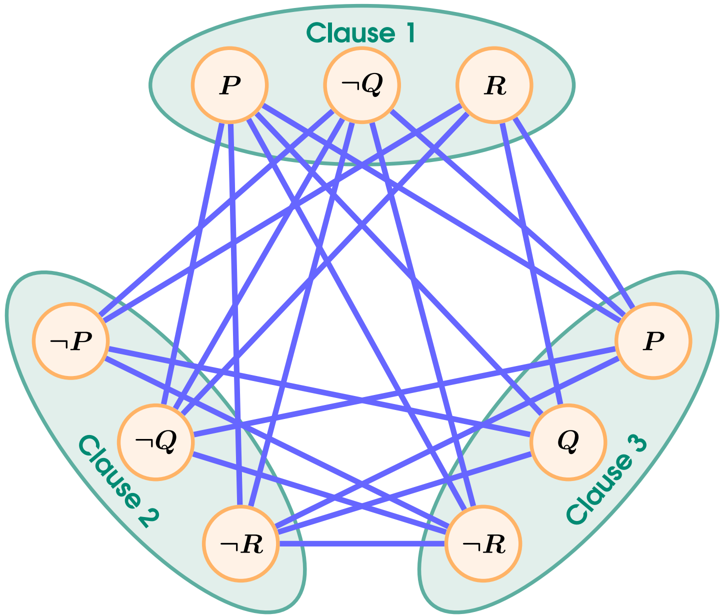
Step 5) Hence, it follows that

$$\boxed{G_F \text{ contains a } k\text{-clique} \quad \Longleftrightarrow \quad F \text{ is satisfiable}}$$

(just make all the literals in the clique 'true')

Q.E.D.

Polynomial Reductions



$$F = (P \vee \neg Q \vee R) \wedge (\neg P \vee \neg Q \vee \neg R) \wedge (P \vee Q \vee \neg R)$$

NP-completeness

NP-completeness

- **NP-hardness**

  - A problem X is said to be **NP-hard** if *every* problem in **NP** can be polynomially reduced to it.

$$\boxed{Y \leq_p X \qquad \text{for all } Y \in X}$$

(X is at least as hard as *every* NP-problem)

- **NP-completeness**

  - A problem X is said to be **NP-complete** if

    (i) X is **NP-hard** (lower bound),

    (ii) X also **belongs** to the class **NP** (upper bound).

Theorem If Y is **NP**-hard and $Y \leq_p X$, then X is **NP**-hard.

Proof:

Step 1) If Y is **NP**-hard, then by definition

$$Z \leq_p Y \qquad \text{for all } Z \in \mathbf{NP}$$

Step 2) But we also have that $Y \leq_p X$, so that

$$Z \leq_p Y \leq_p X \qquad \text{for all } Z \in \mathbf{NP}$$

Q.E.D.

A typical approach to demonstrating that a problem is **NP**-hard is to show that **SAT** is reducible to it. *i.e.* that **SAT** $\leq_p X$.

List of NP-complete Problems

- **(Incomplete) List of NP-complete Problems**
  - The Boolean Satisfiability Problem
  - The Graph Colouring Problem
  - The Clique problem
  - The Hamiltonian Cycle Problem
  - The Travelling Salesman Problem (TSP)
  - The Knapsack Problem

`https://en.wikipedia.org/wiki/List_of_NP-complete_problems`

List of NP-complete Problems

- **(Incomplete) List of NP-complete Problems (cont.)**
  - Many Games and Puzzles
    - Minesweeper (checking for consistency)
    - Lemmings
    - (generalised versions of) Sudoku
    - Pokémon

Et cetera, et cetera...

https://en.wikipedia.org/wiki/List_of_NP-complete_problems

End of Slides!

Feedback

- Let me know how you found today's lecture!

https://goo.gl/forms/xKCooYMGmuyhBjvx1