# Introduction to Algorithms and Data Structures

# Lecture 25: Satisfiability and NP-completeness

Mary Cryan

School of Informatics
University of Edinburgh

# Reductions between (decision) problems

*If I could solve problem $Q$ in polynomial-time, then I would also be able to solve problem $R$ in polynomial-time.*

## Definition

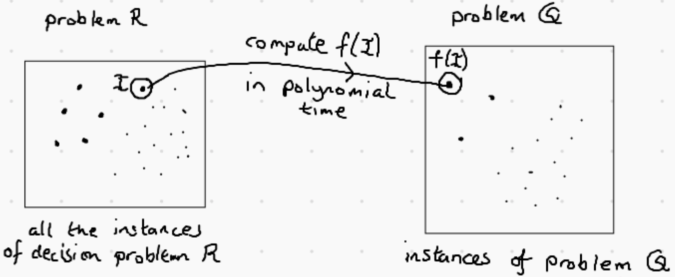
A problem $R$ can be reduced to the problem $Q$ if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all instances $\mathcal{I}$ of $R$

$$R(\mathcal{I}) = 1 \qquad \Leftrightarrow \qquad Q(f(\mathcal{I})) = 1$$

- Means that $R$ is no harder (in the sense of polynomial-time computation) than $Q$. And that $Q$ is "at least as hard" as $R$.
- We write $R \leq_{\mathrm{P}} Q$.

  ($\leq_P$ is not like $\leq$, or even $O(\cdot)$. We can ignore polynomial factors)

# Reductions between (decision) problems




<u>need</u> to ensure $R(I)=1 \iff Q(f(I))=1$ (need both directions of the $\iff$)

<u>then</u> if we had a polynomial-time algorithm to answer Q, we could incorporate f to also build a polynomial-time algorithm to answer R.

# Meaning of $R \leq_P Q$

*If I could solve problem Q in polynomial-time, then I would also be able to solve problem R in polynomial-time.*

- IF $Q$ happens to be in P (is polynomial-time solvable), then I can also solve $R$ in polynomial-time:
  - Take the input instance $\mathcal{I}$ of $R$ and do the polynomial-time work to compute $f(\mathcal{I})$.
  - Pass $f(\mathcal{I})$ to our polynomial-time algorithm for deciding $Q$.
  - Return that answer
- IF $R$ happens to be NP-complete (and, we believe, probably not polynomial-time solvable), then $Q$ is also NP-complete.
  - For any problem $H$ in NP, we can reduce it to $R$ with some $g$ function (because $R$ is NP-complete). But if we instead apply $f(g(\cdot))$ to instances of $H$, this reduces $H$ down to $Q$.
- Note: $\leq_P$ is not like $\leq$, or even $O(\cdot)$. It allows us to ignore polynomial factors.

# NP-completeness

*No (NP) problem is any harder than me.*

## Definition

A decision problem $Q$ is said to be NP-complete if it belongs to the class NP, and it is also the case that for every problem $R$ in NP, $R \leq_{\mathrm{P}} Q$.

The canonical NP-complete problem is Satisfiability.

- This was the first problem to be shown to be NP-complete (late 1960s/early 1970s).
- In the years that followed many other decision problems were shown to be NP complete by reduction to Satisfiability (and the increasing pool of NP-complete problems).

# Satisfiability

## Definition

We say a propositional logical formula $\phi$ over the variables $\{x_1, \ldots, x_n\}$ is in Conjunctive Normal (CNF) if it is written in the form

$$\phi = C_1 \wedge C_2 \wedge \ldots C_m$$

where each of the clauses $C_i$ is a "disjunction of literals" over $\{x_1, \ldots, x_n\}$

For example, if $n = 5$, here are two example CNF formulae:

- $\phi_1 = (x_2 \vee \bar{x_4}) \wedge (x_4 \vee \bar{x_2}) \wedge (x_1 \vee x_2 \vee \bar{x_4})$
- $\phi_2 = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\bar{x_3} \vee x_4) \wedge (\bar{x_4} \vee x_5) \wedge (\bar{x_5} \vee \bar{x_1})$

There are $2^n$ possible assignments to the logical variables of a CNF.

For $n = 5$, consider $x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1$.

- This assignment makes $\phi_1$ true (all clauses are satisfied).
- This assignment does not satisfy $\phi_2$ (2nd clause is violated).

# Cook-Levin theorem

SAT: Given a CNF formula $\phi = C_1 \wedge \ldots \wedge C_m$ over variables $\{x_1, \ldots, x_n\}$, determine whether there is some satisfying assignment for $\phi$.

## Theorem (Cook-Levin)

SAT *is NP-complete.*

Proved/published by Cook in 1971, and independently (behind the Iron Curtain) by Levin in late 1960s.

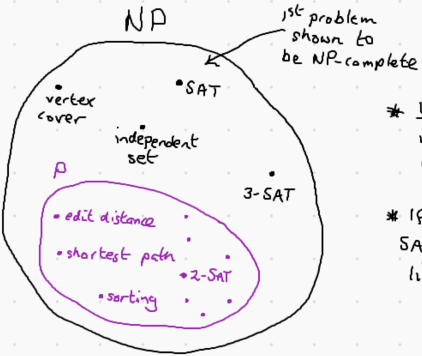
# Cook-Levin theorem

For every NP problem $R$, $R$ "reduces to" $\textsc{Sat}$

Proof works by characterising the behaviour of the polynomial-time verifier for $R$ (on some instance $\mathcal{I}$) as a specially-designed CNF formula.

- Proof works from the "polynomial-time verifier" for the problem $R$ (which must exist), considering the different operations/steps of that verifier.
- All operations/steps of a verifier for the computation of the comparison of a "certificate" against $\mathcal{I}$ can be considered as operations on binary data, can be encoded as Boolean/logical operations.
- Can be used to build a big CNF formula which is true $\Leftrightarrow$ there was some "certificate" for instance $\mathcal{I}$ wrt $R$.
- The encoding of this algorithm can be shown to be "polynomial-size" in the size of $\mathcal{I}$ (because of the verifier being polynomial-time).
- Full details in *Introduction to Theoretical Computer Science (ITCS)*.

# world of NP




* Don't know whether P=NP or whether there are problems in NP but not in P

* If NP and P are different, then SAT (and other NP-complete problems) lie outside P.

# Independent Sets

Our interest is in the INDEPENDENT SET problem.

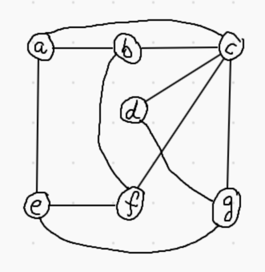
## Definition

Given an undirected graph $G = (V, E)$, an Independent Set (IS) is a subset $I \subseteq V$ such that for every pair $u, v \in I$, $(u, v) \notin E$. The *size* of such an independent set is the cardinality $|I|$.

We can consider the following decision problem:

INDEPENDENT SET: Given an undirected graph $G = (V, E)$, and a natural number $k \in \mathbb{N}$, determine whether $G$ has an IS of size $\geq k$.

# Independent Set




For this graph, the maximum Independent Set will have size 3.
One solution is $\{b, e, d\}$.

# 3-CNF and 3-SAT

## Definition

The CNF formula $\phi$ is said to be 3-CNF if each of its clauses $C_j$ is a disjunction of exactly three literals.

$3\text{-}\mathrm{SAT}$: Given a 3-CNF formula $\phi$ over the variables $\{x_1, \ldots, x_n\}$, determine whether there is an assignment of binary values to $\{x_1, \ldots, x_n\}$ that causes all clauses to be satisfied.

## Theorem

$3\text{-}\mathrm{SAT}$ *is NP-complete.*

We will use $3\text{-}\mathrm{SAT}$ as our "reducing" problem ($R$), to show that INDEPENDENT SET ($Q$) is also NP-complete.

# Reduction: 3-SAT to Independent Set

Our starting point is the $3\text{-SAT}$ problem.

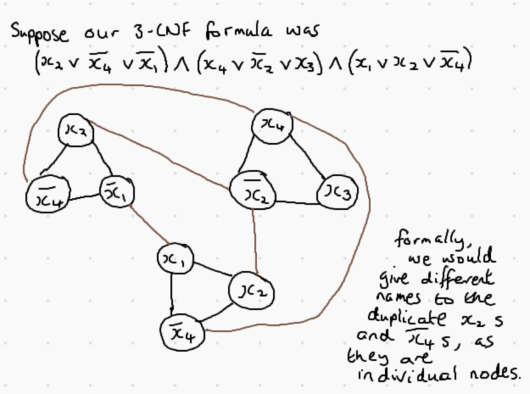
We are given $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, and each of the $C_j$ is $(\ell_{j,1} \vee \ell_{j,2} \vee \ell_{j,3})$ for three *literals* over $\{x_1, \ldots, x_n\}$ (for example, $\ell_{j,1} = x_4, \ell_{j,2} = \bar{x_1}, \ell_{j,3} = \bar{x_9}$).

Interested in an *assignment* to the $\{x_1, \ldots, x_n\}$ which makes *every* clause satisfied.

- $C_j$ is satisfied if *at least one* of its literals are satisfied: $\ell_{j,1}$ or $\ell_{j,2}$, or $\ell_{j,3}$
- For every $j$, we will add nodes $\ell_{j1,}, \ell_{j,2}, \ell_{j,3}$ to our "Independent Set" graph. We will add edges to connect $\ell_{j,1}, \ell_{j,2}, \ell_{j,3}$ as a triangle ... so *at most one* of $\ell_{j,1}$, $\ell_{j,2}, \ell_{j,3}$ can be chosen to make $C_j$ satisfied.
- For every variable $x_i$, add an edge between every *positive literal* $\ell_{j,\cdot} = x_i$ and every *negative literal* $\ell_{k,\cdot} = \bar{x}_i$ ... so if we choose $x_i$ to satisfy some clause, the "opposite literal" cannot be used to satisfy any other clause.
- We will have a Independent Set of size $m \Leftrightarrow$ there is some satisfying assignment for $\phi$.

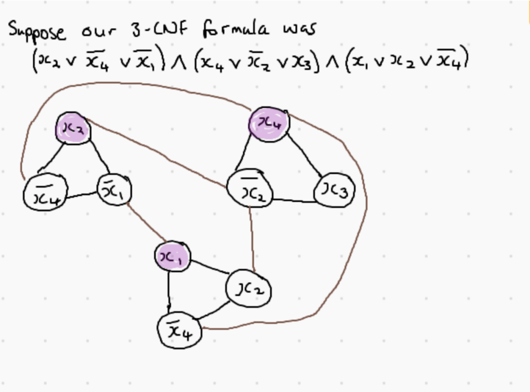
# Example: Independent Set reduction

Suppose our 3-CNF formula was $(x_2 \vee \bar{x_4} \vee \bar{x_1}) \wedge (x_4 \vee \bar{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x_4})$.



# Example: Independent Set solution

Suppose our 3-CNF formula was $(x_2 \vee \bar{x_4} \vee \bar{x_1}) \wedge (x_4 \vee \bar{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x_4})$.



# Example: Independent Set is NP-complete

Size of our "reduction"

- Number of nodes is equal to the number of *literals* of the 3-CNF - $3m$
- Number of edges is at most $3m + 3\frac{m}{2}^2 = O(m^2)$

The construction of the graph (for the $\textsc{Independent-Set}$ instance $f(\mathcal{I})$) is methodical; can be done in polynomial-time:

- First build the "triangle" for each clause $C_j$ and its literals.
  (Collectively, all these can be added to $G$ in $O(m)$ time)
- Next iterate through each variable $i$ adding all "clashing pair" edges for $x_i$.
  (Done naïvely, would take $O(m^2)$ for each $x_i$, so at most $O(n \cdot m^2)$ overall.)

We have argued why the constructed graph
will have an Independent Set of size $\geq m \Leftrightarrow$ our 3-CNF formula was satisfiable.

$$\text{3-SAT} \quad \leq_P \quad \textsc{Independent Set}$$

# Reading and Working

**Reading:**

CLRS Section 34.3 (reducibility, NP-Completeness of Circuit Satisfiability), Section 34.4 (reduction of problems to each other.)

Algs.Illum. OR "Algorithms Illuminated" Sections 22.3, 22.4, 22.5

The following short video is worth a look: Short video about Cook-Levin

**Working:**

- We never showed that $3\text{-SAT}$ was NP-complete. Think about making a reduction from $\text{SAT}$ to $3\text{-SAT}$ to show that $3\text{-SAT}$ is also NP-complete.