In fact, every problem in $\mathbf{NP}$ can be reduced to CIRCUIT SAT.

Definition of $\mathbf{NP}$ completeness

A problem $Q$ is $\mathbf{NP}$-complete if it is in $\mathbf{NP}$ and every problem in $\mathbf{NP}$ can be reduced to $Q$.

CIRCUIT SATISFIABILITY is $\mathbf{NP}$-complete.

So are all the other problems we’ve discussed.

If you can solve one $\mathbf{NP}$-complete problem in polynomial time, you can solve every $\mathbf{NP}$ problem in polynomial time!

Unfortunately, it is probably the case that we can’t solve any $\mathbf{NP}$-complete problem in polynomial time.
Can we say that one problem is as hard as another problem?

Yes - notion of a reduction from one problem to another.

For example, suppose we could design an algorithm A which runs in polynomial time and does the following:

- takes as input an instance of CLIQUE problem \((G,k)\)
- designs and outputs a circuit \(C\) such that
  \[ C \text{ is satisfiable } \left( \text{there exist } x_1,\ldots,x_n \text{ such that } C(x_1,\ldots,x_n)=1 \right) \text{ if and only if there is a clique in } G \text{ of size } k. \]

Then we say that CLIQUE reduces to CIRCUIT SAT; meaning that CIRCUIT SAT is at least as hard as CLIQUE:

- If there was a polynomial-time algorithm B to solve CIRCUIT SAT, then we could make a polynomial-time algorithm D to solve CLIQUE:
  - D takes as input an instance of CLIQUE problem \((G,k)\)
  - D calls algorithm A on \((G,k)\); A output a circuit \(C\)
  - D calls algorithm B on circuit \(C\); D output the answer it gets from B
- Running time of D: polynomial
A few other problems that are in \textbf{NP}:

\textbf{TRAVELLING SALESMAN TOUR}(G,k)

Given a graph of cities and inter-city distances, is there a way for the travelling salesman to go around visiting each city exactly once travelling no more than \( k \) miles?

\textbf{INDEPENDENT SET}(G,k)

Given a graph \( G \) and an integer \( k \), are there \( k \) nodes such that no two are connected?

\textbf{CIRCUIT SATISFIABILITY}(C) \textbf{(CIRCUIT SAT)}

Given a circuit \( C \), is there a setting of its inputs, \( x_1, \ldots, x_n \) such that \( C(x_1, \ldots, x_n) = 1 \)?

How are these problems different?

Seem to be totally unrelated - e.g. not all are about graphs

How are they the same?

All of these problems have a yes/no answer.

All of these problems are hard -- no one has ever found a good (polynomial time) algorithm for any of them...

(even though there are good algorithms to check a claimed solution)
The “checking” algorithm takes as input an instance of the problem and a solution that is claimed to be correct. The checking algorithm then checks whether this claimed solution is indeed a correct solution to this instance of the problem.

Example: CLIQUE(G,k)
Given a graph G and an integer k, are there k nodes in G such that any two of these k nodes are connected by an edge?

Check-Clique(G,k,C) /* check if C is a k-clique of G */
check that C contains k nodes
for every pair of nodes u and v in C, check that edge (u,v) is in G
if any of these checks fail, return FALSE, otherwise return TRUE
Running time: O(V+E)
Thus, CLIQUE is in \textbf{NP}
Definition: **Nondeterministic Polynomial Time**

A problem is in **NP** iff:

The problem asks a yes/no question.

We can devise an algorithm for this problem that “checks” a solution to the problem in $T(n)$ steps, and $T(n) < n^c$ for some constant $c$.

What does it mean to have this “checking” algorithm?

Example: HAMILTONIAN CYCLE (G)

Given a graph G, is there a simple cycle that visits every node exactly once?

Check-Hamiltonian-Cycle(G,H) /*check if H is a Hamiltonian cycle of G*/

check that the edges given in H indeed exist in graph G
check that the path given by H hits every vertex in G exactly once
check that the path given by H is a cycle (starts & ends in the same vertex)
if any of these checks fail, return FALSE, otherwise return TRUE

Running time: $O(V+E)$

Thus, HAMILTONIAN CYCLE is in **NP**
So if there is an algorithm that runs in time $n^{4000}$, the problem is in \textbf{P}. But if every algorithm for the problem takes $n^{\log\log n}$ time, the problem is not in \textbf{P}.

Is this reasonable?

Not always: $n^{4000}$ is a lot bigger than $n^{\log\log n}$ for any useful values of $n$.

However: mostly, problems in \textbf{P} have reasonable constants $c$.

Are all problems in \textbf{P}?

No! Some problems provably take exponential ($2^n$) time to be solved. Some problems can’t be solved \textit{at all} by any computer.

Can we tell if a problem has a polynomial-time solution?

Not always. Some polynomial-time solutions are very tricky - can take decades to find a provably polynomial-time solution.

Can we prove that a problem is hard?

Usually not... hard to rule out the existence of a clever algorithm.

Can we find evidence that a problem is hard?

Often yes - theory of \textit{NP-Completeness}.
Many problems shown to be “probably very hard in the worst case”. 
TOPIC 21: P and NP

What problems are easy? What problems are hard?

Intuition: a problem is easy if it doesn’t take “too much” time to solve.
Example: SHORTEST PATH
Given a graph with n vertices, can find the shortest path in O(VE) time.
That’s pretty fast -- SHORTEST PATH is easy.

How much time is “too much”?
Taking time $2^n$ is certainly too much, e.g. $2^{100}$ is really big!
What about $n^3$, $n^5$, $n^{50}$, $2^{\sqrt{n}}$?

Definition: Polynomial Time
A problem is in $\textbf{P}$ (can be solved in polynomial time) iff
The problem asks a yes/no question.
The problem can be solved in $T(n)$ steps, and $T(n) < n^c$ for some constant $c$

$n$ is the size of the problem’s input
$c$ can be set to some arbitrarily large constant but can’t grow with $n$.

informally, $\textbf{P}$ corresponds to problems that are easy (doable on a computer)