CMSC 451: Polynomial-time Reductions & NP-completeness

Carl Kingsford

April 28, 2009
We will define two classes of problems called \textbf{NP} and \textbf{NP}-complete.

We need some new ideas.
Recall the independent set problem (decision version):

**Independent Set**

Given a graph $G$, is there set $S$ of size $\geq k$ such that no two nodes in $S$ are connected by an edge?

Finding the set $S$ is hard (we will see).

But if I give you a set $S^*$, checking whether $S^*$ is the answer is easy: check that $|S| \geq k$ and no edges go between 2 nodes in $S^*$.

$S^*$ acts as a certificate that $\langle G, k \rangle$ is a yes instance of Independent Set.
Def. An algorithm $B$ is an efficient certifier for problem $X$ if:

1. $B$ is a polynomial time algorithm that takes two input strings $I$ (instance of $X$) and $C$ (a certificate).
2. $B$ outputs either yes or no.
3. There is a polynomial $p(n)$ such that for every string $I$:

$$I \in X \text{ if and only if there exists string } C \text{ of length } \leq p(|I|) \text{ such that } B(I, C) = yes.$$ 

$B$ is an algorithm that can decide whether an instance $I$ is a yes instance if it is given some “help” in the form of a polynomially long certificate.
Certifiers

User provides instance as usual

Certificate is magically guessed
The class NP

\textbf{NP} is the set of languages for which there exists an efficient certifier.
The class $\textbf{NP}$

$\textbf{NP}$ is the set of languages for which there exists an efficient certifier.

$\textbf{P}$ is the set of languages for which there exists an efficient certifier that ignores the certificate.

That’s the difference: A problem is in $\textbf{P}$ if we can decided them in polynomial time. It is in $\textbf{NP}$ if we can decide them in polynomial time, if we are given the right certificate.
**Theorem**

$P \subseteq NP$

*Proof.* Suppose $X \in P$. Then there is a polynomial time algorithm $A$ for $X$.

To show that $X \in NP$, we need to design an efficient certifier $B(I, C)$.

Just take $B(I, C) = A(I)$. □

Every problem with a polynomial time algorithm is in $NP$. 
The big question:

\[ P \neq NP \text{?} \]

We know \( P \subseteq NP \). So the question is:

Is there some problem in \( NP \) that is not in \( P \)?

Seems like the power of the certificate would help a lot. But no one knows...
We want to prove some problems are computationally difficult.

As a first step, we settle for relative judgements:

**Problem X is at least as hard as problem Y**

To prove such a statement, we reduce problem Y to problem X:

*If you had a black box that can solve instances of problem X, can you solve any instance of Y using polynomial number of steps, plus a polynomial number of calls to the black box that solves X?*
Polynomial Reductions

- If problem $Y$ can be reduced to problem $X$, we denote this by $Y \leq_P X$.

- This means “$Y$ is polynomial-time reducible to $X$.”

- It also means that $X$ is at least as hard as $Y$ because if you can solve $X$, you can solve $Y$.

- **Note:** We reduce to the problem we want to show is the harder problem.
Polynomial Problems

Suppose:

• $Y \leq_p X$, and

• there is an polynomial time algorithm for $X$.

Then, there is a polynomial time algorithm for $Y$.

Why?
Suppose:

- \( Y \leq_P X \), and
- there is a polynomial time algorithm for \( X \).

Then, there is a polynomial time algorithm for \( Y \).

Why?
**Theorem**

*If* $Y \leq_P X$ *and* $Y$ *cannot be solved in polynomial time, then* $X$ *cannot be solved in polynomial time.*

Why? If we *could* solve $X$ in polynomial time, then we’d be able to solve $Y$ in polynomial time, contradicting the assumption.

So: If we could find one hard problem $Y$, we could prove that another problem $X$ is hard by reducing $Y$ to $X$. 
**Def.** A *vertex cover* of a graph is a set $S$ of nodes such that every edge has at least one endpoint in $S$.

In other words, we try to “cover” each of the edges by choosing at least one of its vertices.

(Yes, “Vertex Cover” is a horrible name: we’re covering *edges* with vertices. There’s no hope to change this now.)
Independent Set to Vertex Cover

**Independent Set**

Given graph $G$ and a number $k$, does $G$ contain a set of at least $k$ independent vertices?

**Vertex Cover**

Given a graph $G$ and a number $k$, does $G$ contain a vertex cover of size at most $k$.

Can we reduce independent set to vertex cover?
**Theorem**

If $G = (V, E)$ is a graph, then $S$ is an independent set $\iff V - S$ is a vertex cover.

**Proof.** $\Rightarrow$ Suppose $S$ is an independent set, and let $e = (u, v)$ be some edge. Only one of $u, v$ can be in $S$. Hence, at least one of $u, v$ is in $V - S$. So, $V - S$ is a vertex cover.

$\Leftarrow$ Suppose $V - S$ is a vertex cover, and let $u, v \in S$. There can’t be an edge between $u$ and $v$ (otherwise, that edge wouldn’t be covered in $V - S$). So, $S$ is an independent set. □
Independent Set $\leq_P$ Vertex Cover

To show this, we change any instance of Independent Set into an instance of Vertex Cover.

Proof.

- Given an instance of Independent Set $\langle G, k \rangle$, with $|G| = n$
- we ask our Vertex Cover black box if there is a vertex cover of with $n - k$ vertices.

By our previous theorem, $S$ is an independent set iff $V - S$ is a vertex cover.

So: $G$ has a independent set of size $k$ iff $G$ has a vertex cover of size $n - k$. 
Actually, we also have:

\[ \text{Vertex Cover} \leq_P \text{Independent Set} \]

**Proof.** To decide if \( G \) has an vertex cover of size \( k \), we ask if it has a independent set of size \( n - k \). □

So: Vertex Cover and Independent Set are equivalently difficult.
**NP-completeness**

**Def.** We say $X$ is NP-complete if:

- $X \in \textbf{NP}$
- for all $Y \in \textbf{NP}$, $Y \leq_P X$.

If these hold, then $X$ can be used to solve every problem in $\textbf{NP}$.

Therefore, $X$ is definitely at least as hard as every problem in $\textbf{NP}$. 
**Theorem**

If $X$ is NP-complete, then $X$ is solvable in polynomial time if and only if $P = NP$.

*Proof.* If $P = NP$, then $X$ can be solved in polytime.

Suppose $X$ is solvable in polytime, and let $Y$ be any problem in NP. We can solve $Y$ in polynomial time: reduce it to $X$.

Therefore, every problem in NP has a polytime algorithm and $P = NP$. 
Theorem

If Y is NP-complete, and

1. X is in NP
2. \( Y \leq_{P} X \)

then X is NP-complete.

In other words, we can prove a new problem in NP-complete by reducing some other NP-complete problem to it.

Proof. Let Z be any problem in \( \textbf{NP} \). Since Y is NP-complete, \( Z \leq_{P} Y \). By assumption, \( Y \leq_{P} X \). Therefore: \( Z \leq_{P} Y \leq_{P} X \). \( \square \)
**Boolean Formulas:**

**Variables:** \( x_1, x_2, x_3 \) (can be either \textbf{true} or \textbf{false})

**Terms:** \( t_1, t_2, \ldots, t_\ell \): \( t_j \) is either \( x_i \) or \( \bar{x}_i \)
(meaning either \( x_i \) or \textbf{not} \( x_i \)).

**Clauses:** \( t_1 \lor t_2 \lor \cdots \lor t_\ell \) (\( \lor \) stands for “OR”)
A clause is \textbf{true} if any term in it is \textbf{true}.

**Example 1:** \((x_1 \lor \bar{x}_2), (\bar{x}_1 \lor \bar{x}_3), (x_2 \lor \bar{v}_3)\)

**Example 2:** \((x_1 \lor x_2 \lor \bar{x}_3), (\bar{x}_2 \lor x_1)\)
Def. A truth assignment is a choice of true or false for each variable, ie, a function \( v : \{x_1, \ldots, x_n\} \rightarrow \{\text{true, false}\} \).

Def. A CNF formula is a conjunction of clauses:

\[ C_1 \land C_2, \land \cdots \land C_k \]

Example: \((x_1 \lor \overline{x}_2) \land (\overline{x}_1 \lor \overline{x}_3) \land (x_2 \lor \overline{v}_3)\)

Def. A truth assignment is a satisfying assignment for such a formula if it makes every clause true.
**SAT and 3-SAT**

**Satisfiability (SAT)**

Given a set of clauses $C_1, \ldots, C_k$ over variables $X = \{x_1, \ldots, x_n\}$ is there a satisfying assignment?

**Satisfiability (3-SAT)**

Given a set of clauses $C_1, \ldots, C_k$, each of length 3, over variables $X = \{x_1, \ldots, x_n\}$ is there a satisfying assignment?
Cook-Levin Theorem

**Theorem (Cook-Levin)**

3-SAT is NP-complete.

Proven in early 1970s by Cook. Slightly different proof by Levin independently.

**Idea of the proof:** encode the workings of a Nondeterministic Turing machine for an instance \(I\) of problem \(X \in \text{NP}\) as a SAT formula so that the formula is satisfiable if and only if the nondeterministic Turing machine would accept instance \(I\).

We won’t have time to prove this, but it gives us our first hard problem.
Reducing 3-SAT to Independent Set

**Thm.** 3-SAT $\leq_P$ Independent Set

*Proof.* Suppose we have an algorithm to solve Independent Set, how can we use it to solve 3-SAT?

To solve 3-SAT,

- you have to choose a term from each clause to set to true,
- but you can’t set both $x_i$ and $\overline{x}_i$ to true.

How do we do the reduction?
3-SAT $\leq_P$ INDEPENDENT SET

$$(x_1 \lor x_2 \lor \overline{x_3}) \land (x_2 \lor x_3 \lor \overline{x_4}) \land (x_1 \lor \overline{x_2} \lor x_4)$$
Proof

**Theorem**

_This graph has an independent set of size k iff the formula is satisfiable._

*Proof.* $\implies$ If the formula is satisfiable, there is at least one true literal in each clause. Let $S$ be a set of one such true literal from each clause. $|S| = k$ and no two nodes in $S$ are connected by an edge.

$\implies$ If the graph has an independent set $S$ of size $k$, we know that it has one node from each “clause triangle.” Set those terms to true. This is possible because no two are negations of each other.

□
General Strategy for Proving Something is NP-complete:

1. Must show that $X \in \textbf{NP}$. Do this by showing there is an certificate that can be efficiently checked.

2. Look at all the problems that are known to be NP-complete (there are thousands), and choose one $Y$ that seems “similar” to your problem in some way.

3. Show that $Y \leq_{P} X$. 