1 Definition of Reduction

Even if I can’t prove a problem hard in an absolute sense, I can prove that it is at least as hard as another problem.

- Let $L$ and $L'$ be two decision problems.
- Say that $L'$ polynomially reduces to $L$, denoted $L' \leq_p L$, if there exists a function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ such that
  1. $f$ runs in time polynomial in its input size
  2. For every $x \in \{0, 1\}^*$, $x \in L'$ iff $f(x) \in L$
- Recall that $x$s are problem instances.
- So, $f$ turns instances of problem $L'$ into instances of problem $L$.

How about an example?

- Recall problem CLIQUE($G, k$): given an undirected graph $G$ and integer $k$, does $G$ contain a complete subgraph on at least $k$ vertices?
- A related problem: ISET($G, k$): given an undirected graph $G$ and integer $k$, does $G$ contain an independent set on at least $k$ vertices?
- (An independent set is a subgraph in which no two vertices have an edge between them.)

**Lemma**: ISET $\leq_p$ CLIQUE.

**Pf**: Let $(G, k)$ be an input to ISET. Define $f$ as follows.

- Let $G'$ be a graph with the same vertex set as $G$, such that for each pair of vertices $u, v$, edge $(u, v)$ is in $G'$ iff it is not in $G$.
- Set $f((G, k)) = (G', k)$. 

• Now \( f \) runs in worst-case time \( \Theta(|G|^2) \), so it is clearly polynomial in its input size.

• **Claim 1**: If \((G, k) \in \text{ISET}\), then \((G', k) \in \text{CLIQUE}\).

• **Pf**: If \( G \) contains an independent set of size \( k \), then no pair of vertices in this subgraph is joined by an edge.

• By construction of \( G' \), all pairs in the set will be joined by edges in \( G' \), yielding a \( k \)-clique. QED

• **Claim 2**: If \((G', k) \in \text{CLIQUE}\), then \((G, k) \in \text{ISET}\).

• **Pf**: if \( G' \) contains a clique of size \( k \), then every pair of vertices in this subgraph is joined by an edge.

• By construction of \( G' \), no pair in the set was joined by an edge in \( G \), so the subgraph forms an iset in \( G \). QED

• Note four main steps in the reduction:
  1. Specify the transformation \( f \)
  2. Prove that \( f \) runs in time polynomial in its input size.
  3. Prove one direction of the iff: \( x \in L' \rightarrow f(x) \in L \).
  4. Prove the other direction of the iff: \( x \in L' \leftarrow f(x) \in L \).

So what?

• Suppose \( L' \leq_p L \), and let \( f \) be as above.

• Let \( M \) be an algorithm to decide \( L \). Using \( M \) and \( f \), we can implement an algorithm \( M' \) to decide \( L' \).

• Given an input \( x \), implement \( M' \) as follows:
  1. Compute \( y = f(x) \).
  2. Compute the result \( r = M(y) \).
  3. Return \( r \).

• By properties of \( f \), \( x \in L' \) iff \( M(y) = \text{true} \).

• Moreover, suppose \( M \) runs in time polynomial in \(|y|\).

• \( f \) is also polynomial-time, so \( M' \) runs in time polynomial in \(|x|\)! We have proven that . . .
Lemma: if $L \in P$, and $L' \leq_p L$, then $L' \in P$.

(Hence, if $L$ is easy, so is $L'$.)

Equivalently: if $L' \notin P$ and $L' \leq_p L$, then $L \notin P$.

(Hence, if $L'$ is hard, so is $L$.)

2 NP-Completeness

Let’s get back to the whole $P=NP$ business.

Defn: a decision problem $L$ is said to be $NP$-complete if

1. $L \in NP$.
2. For every $L' \in NP$, $L' \leq_p L$.

Can read second condition as “$L$ is as hard as any problem in NP.”

Hence, if $L$ satisfies only condition 2, we say that $L$ is $NP$-hard.

(Condition 1 is also necessary – an NP-hard problem need not be in NP!)

What can we say about NP-complete problems?

Fact 1: let $L$ be a decision problem, and let $L'$ be an NP-complete problem.

If $L' \leq_p L$, then $L$ is NP-hard.

If we also know that $L \in NP$, then $L$ is NP-complete.

(Proof follows by transitivity of polytime reduction.)

Fact 2: if any NP-complete problem is in $P$, then $P = NP$.

Pf: Suppose $L$ is NP-complete. For any $L' \in NP$,

$$L' \leq_p L.$$  

Hence, if $L \in P$, then $L' \in P$. QED

Contrapositive says: if $P \neq NP$, then no NP-complete problem is in $P$.

We don’t know whether $P = NP$, however:

1. It’s really hard to answer this question, so you are not likely to do so by finding a polynomial-time algorithm for an NP-complete problem.
2. Most people conjecture that $P \neq NP$.

In conclusion, NP-complete problems are practically impossible to solve in polynomial time.

(Is there such a thing as an NP-complete problem? Hold on...)
3 What about Optimization?

Oh wait, this is a class on optimization.

- Let $Q$ be an optimization problem, and let $Q(x)$ be the value of an optimal (maximal) feasible solution for input $x$.

- Let $\text{DEC}_Q(x, y)$ be the corresponding canonical decision problem: is there a feasible solution to $x$ with value at least $y$?

- **Defn:** if $\text{DEC}_Q(x, y)$ is NP-complete, $Q$ is said to be an **NP-optimization problem**.

- **Claim:** Let $Q$ be an NP-optimization problem. Suppose there exists an algorithm to compute $Q(x)$ in time polynomial in $|x|$. Then $P = NP$.

- **Pf:** suppose we can compute $Q(x)$ in time polynomial in $|x|$.

- The following algorithm solves $\text{DEC}_Q(x, y)$:
  
  1. Compute $y^* = Q(x)$.
  2. If $y \leq y^*$, return true; else, return false.

- By assumption, we can find $y^*$ in time polynomial in $|x|$, so we can solve $\text{DEC}_Q(x, y)$ in time polynomial in $|x|$ and $|y|$.

- But $\text{DEC}_Q(x, y)$ is NP-complete; hence, $P = NP$. QED

**Conclusion:** NP-optimization problems, like NP-complete decision problems, are **practically impossible** to solve (exactly) in polynomial time.

4 The Program

I haven’t actually shown you that NP-complete problems exist.

- First, we will sketch a proof that *one* NP-complete problem exists: Boolean formula satisfiability (“SAT”).

- (follows Cook, 1971)

- Then, we will prove a bunch of other problems NP-complete using reduction arguments.

- Your job is to learn to do your own reduction arguments.