Reductions for Proving NP-Hardness

Yufei Tao's Teaching Team

Department of Computer Science and Engineering
Chinese University of Hong Kong

This tutorial will discuss how to prove a problem to be **NP-hard**. The technique we will use is called **reduction**.

Remark: Reductions are discussed in detail in CSCI3130 (Formal Languages and Automata). The purpose of today's material is to permit students without CSCI3130 experiences to learn about reductions.

Review

In computer science, there is a set of **NP-hard** problems for which no polynomial-time algorithms can exist **unless** $\mathcal{P} = \mathcal{NP}$.

- $\mathcal{P}$ = the set of problems that can be solved in polynomial time on a **deterministic** Turing machine
- $\mathcal{NP}$ = the set of problems that can be solved in polynomial time on a **non-deterministic** Turing machine

Reduction

We can argue for the NP-hardness of a problem P_1 in two steps:

1. Identify another problem P_2 that is **already known** to be NP-hard.
2. Prove that, if we are given an arbitrary polynomial-time algorithm $\mathcal{A}_1$ for P_1, then we can always design a polynomial time algorithm $\mathcal{A}_2$ for P_2 (by treating $\mathcal{A}_1$ as a **black box**).

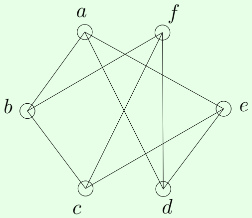
This method is called **reduction**.

- We say that P_2 can be **reduced** (i.e., converted) to P_1 in polynomial time.
- Since P_2 is NP-hard, so is P_1.

The Clique Decision Problem: Let $G = (V, E)$ be an undirected graph. Given an integer k, decide whether we can find a set S of at least k vertices in V that are mutually connected (i.e., there is an edge between any two vertices in S).

Those k vertices and the edges among them form a ***k*-clique**.

Example: Consider



The answer is "yes" for $k \leq 3$, but "no" for $k \geq 4$.

We will prove that the clique decision problem is NP-hard. This means that no algorithm can solve the problem in time polynomial in both $|V|$ and k unless $\mathcal{P} = \mathcal{NP}$.

- $O(|V|^k)$ is **not** polynomial in k.

> **Think:** If k is a constant (e.g., 3), can you solve the problem in polynomial time?

This is our problem P_1. To apply reduction, we need to identify a problem P_2.

3-SAT

Variable: a boolean unknown x that can be assigned 0 or 1.
Literal: a variable x or its negation $\bar{x}$.
Clause: the OR of **up to** 3 literals.
Formula: the AND of clauses

The 3-SAT problem: Is there a truth assignment for the variables under which the formula evaluates to 1? Such an assignment is called a **certificate**.

Example:

$$(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\bar{x_1} \vee \bar{x_4})$$

The answer is "yes". A certificate: $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$.

$$(x_1) \wedge (\bar{x_1} \vee x_2) \wedge (\bar{x_2})$$

The answer is "no".

The **input size** of 3-SAT is the number of clauses.

> **Lemma:** 3-SAT is NP-hard.

In other words, no algorithm can solve 3-SAT in time polynomial in the number of clauses. The proof of the lemma is not required in this course.

We will reduce 3-SAT to clique decision. Specifically, we will prove:

Theorem: If we have an algorithm $\mathcal{A}$ solving the clique decision problem in time in $|V|$ and k, we can solve the 3-SAT problem using $\mathcal{A}$ in time polynomial in the number of clauses.

The next few slides serve as a proof of the theorem.

Given an input to 3-SAT — namely a formula F with k clauses — we will construct a graph $G(V, E)$ such that F has a truth assignment **if and only if** G has a k-clique.

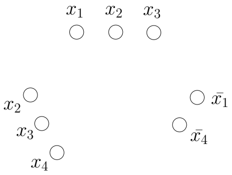
We construct $G(V, E)$ as follows:

- For each clause, create a vertex in V for every literal in the clause.
- For each pair of distinct vertices $u, v \in V$, create an edge $\{u, v\}$ in E if the literals corresponding to u, v
  - do not appear in the same clause, **and**
  - are not negations of each other.

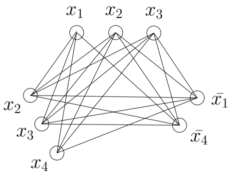
Example 1

Consider formula $F = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\bar{x_1} \vee \bar{x_4})$

First step: create vertices



Second step: create edges

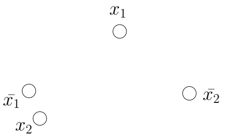


There are 3-cliques in the graph.

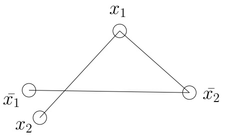
Example 2

Consider formula $F = (x_1) \wedge (\bar{x_1} \vee x_2) \wedge (\bar{x_2})$

First step: create vertices



Second step: create edges



There are no 3-cliques in the graph.

Claim 1: If F has a certificate, then G has a k-clique.

Proof: Every clause has a literal equal to 1 under the certificate. Pick one such literal from every clause (if a clause has multiple literals equal to 1, any of them can be picked).

No two literals picked can be negations of each other (because x and $\bar{x}$ cannot both be 1).

Let v_i be the vertex in G corresponding to the literal picked from the i-th clause ($1 \leq i \leq k$). The claims follows from the fact that there is an edge between any two distinct vertices in $\{v_1, v_2, ..., v_k\}$. $\square$

Claim 2: If G has a k-clique, F has a certificate.

Proof: Let $v_1, v_2, ..., v_k$ be the vertices of the k-clique in G.

The literals corresponding to the k vertices must come from different clauses (because the vertices of two literals from the same clause are not connected).

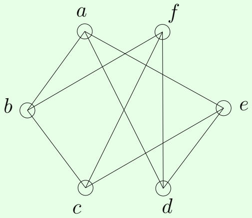
The literals corresponding to the k vertices cannot be negations of each other (because if two literals are negations of each other, their vertices are not connected).

We can therefore construct a certificate by setting those k literals to 1. $\square$

We now know that clique decision is NP-hard. Let us now consider its optimization version:

The Maximum Clique Problem: Let $G = (V, E)$ be an undirected graph. Find the maximize $k \in [1, |V|]$ such that G has a k-clique.

Example: Consider the following graph.



The value of k is 3.

Think: How to prove that the maximum clique problem cannot be solved in polynomial time unless $\mathcal{P} = \mathcal{NP}$?