NP Hardness Reductions
Overview So Far

- We have defined classes $P$ and $NP$
- We have some notion of $NP$ hardness and $NP$ completeness
- We said a problem $X$ is $NP$-hard $\equiv$ if $X \in P$ then $P = NP$
  - Alternate definition: every problem in $NP$ poly-time reduces to it
- A problem $X$ is $NP$-complete if it is $NP$-hard and in $NP$

Focus on decision problems

We will define these reductions today
Overview

• We have defined classes \( P \) and \( NP \)
• We have some notion of \( NP \) hardness and \( NP \) completeness
• We said a problem \( X \) is \( NP \)-hard \( \equiv \) if \( X \in P \) then \( P = NP \)
  • Alternate definition: every problem in \( NP \) poly-time reduces to it
• A problem \( X \) is \( NP \)-complete if it is \( NP \)-hard and in \( NP \)
• (Cook-Levin). 3SAT/SAT is \( NP \) hard
• Today: Problem reductions!
  • Strategy to prove a problem is \( NP \) hard: Reduce a known \( NP \) hard problem to it
• Will do a bunch of reductions next few days
Relative Hardness

• How do we compare the relative hardness of problems?

• Recurring idea in this class: reductions!

• Informally, we say a problem $X$ reduces to a problem $Y$, if can use an algorithm for $Y$ to solve $X$
  
  • E.g., Bipartite matching reduces to max flow

Intuitively, if problem $X$ reduces to problem $Y$, then solving $X$ is no harder than solving $Y
**[Karp] Reductions**

**Definition.** Decision problem $X$ polynomial-time (Karp) reduces to decision problem $Y$ if given any instance $x$ of $X$, we can construct an instance $y$ of $Y$ in polynomial time s.t. $x \in X$ if and only if $y \in Y$.

**Notation.** $X \leq_p Y$

- Solving $X$ is no harder than solving $Y$: if we have an algorithm for $Y$, we can use it + a polynomial-time reduction to solve $X$. 

![Diagram](attachment:image.png)
Reductions Quiz

Say $X \leq_p Y$. Which of the following can we infer?

- If $X$ can be solved in polynomial time, then so can $Y$.
- $X$ can be solved in poly time iff $Y$ can be solved in poly time.
- If $X$ cannot be solved in polynomial time, then neither can $Y$.
- If $Y$ cannot be solved in polynomial time, then neither can $X$.
Reductions Quiz

Say $X \leq_p Y$. Which of the following can we infer?

- If $X$ can be solved in polynomial time, then so can $Y$.
- $X$ can be solved in poly time iff $Y$ can be solved in poly time.
- $\Box$ If $X$ cannot be solved in polynomial time, then neither can $Y$.
- If $Y$ cannot be solved in polynomial time, then neither can $X$.
Digging Deeper

- **Graph 2-Color** reduces to **Graph 3-color**
  - We'll see this soon

- **Graph 2-Color** can be solved in polynomial time
  - How?
    - Can decide if a graph is bipartite in $O(n + m)$ time using BFS

- **Graph 3-color** (we’ll show) is NP hard and unlikely to have a polynomial-time solution

Intuitively, if problem $X$ reduces to problem $Y$, then solving $X$ is no harder than solving $Y$
Use of Reductions: $X \leq_p Y$

**Design algorithms:**

- If $Y$ can be solved in polynomial time, we know $X$ can also be solved in polynomial time.

**Establish intractability:**

- If we know that $X$ is known to be impossible/hard to solve in polynomial-time, then we can conclude the same about problem $Y$.

**Establish Equivalence:**

- If $X \leq_p Y$ and $Y \leq_p X$ then $X$ can be solved in poly-time iff $Y$ can be solved in poly time and we use the notation $X \equiv_p Y$.
**NP hard: Operational Definition**

- **New definition of NP hard using reductions.**
  - A problem $Y$ is NP hard, if for any problem $X \in \text{NP}$, $X \leq_p Y$

- Recall we said $Y$ is NP hard if $Y \in \text{P}$, then $\text{P} = \text{NP}$.

- Let's show that both definitions are equivalent:
  - ($\Rightarrow$) every problem in $\text{NP}$ reduces to $Y$ in poly-time, and if $Y \in \text{P}$, then $\text{P} = \text{NP}$
  - ($\Leftarrow$) Suppose $Y \in \text{P}$, then $\text{P} = \text{NP}$: which means every problem in $\text{NP}(= \text{P})$ reduces to $Y$
Proving NP Hardness

- To prove problem $Y$ is NP-hard
  - Difficult to prove every problem in NP reduces to $Y$
  - Instead, we use a known-NP-hard problem $Z$
  - We know every problem $X$ in NP, $X \leq_p Z$
  - Notice that $\leq_p$ is transitive
  - Thus, enough to prove $Z \leq_p Y$

To prove that a problem $Y$ is NP hard, reduce a known NP hard problem $Z$ to $Y$
Known NP Hard Problems?

- For now: **3SAT** and **SAT** (Cook-Levin Theorem)

- We will prove a whole repertoire of NP hard and NP complete problems by using reductions

- Before reducing **3SAT** to other problems to prove them NP hard, let us practice some easier reductions first

---

**To prove that a problem \( Y \) is NP hard, reduce a known NP hard problem \( Z \) to \( Y \)**
VERTEX-COVER $\equiv_p$ IND-SET
Given a graph $G = (V, E)$, an independent set is a subset of vertices $S \subseteq V$ such that no two of them are adjacent, that is, for any $x, y \in S$, $(x, y) \notin E$

- What is the decision version of the IND-SET problem?
- IND-SET decision Problem. Given a graph $G = (V, E)$ and an integer $k$, does $G$ have an independent set of size at least $k$?
Vertex-Cover

Given a graph $G = (V, E)$, a vertex cover is a subset of vertices $T \subseteq V$ such that for every edge $e = (u, v) \in E$, either $u \in T$ or $v \in T$.

• What is the decision version of the VERTEX_COVER problem?

• VERTEX-COVER decision Problem. Given a graph $G = (V, E)$ and an integer $k$, does $G$ have a vertex cover of size at most $k$?
Our First Reduction

- **VERTEX-COVER \( \leq_p \) IND-SET
  - Suppose we know how to solve independent set, can we use it to solve vertex cover?

- **Claim.** \( S \) is an independent set of size \( k \) iff \( V - S \) is a vertex cover of size \( n - k \).

- **Proof.** (\( \Rightarrow \)) Consider an edge \( e = (u, v) \in E \)
  - \( S \) is independent: \( u, v \) both cannot be in \( S \)
  - At least one of \( u, v \in V - S \)
  - \( V - S \) covers \( e \)
  - ■
Our First Reduction

- **VERTEX-COVER \( \leq_p \) IND-SET**
  - Suppose we know how to solve independent set, can we use it to solve vertex cover?

- **Claim.** \( S \) is an independent set of size \( k \) iff \( V - S \) is a vertex cover of size \( n - k \).

- **Proof.** \( \leftrightarrow \) Consider an edge \( e = (u, v) \in E \)
  - \( V - S \) is a vertex cover: at least one of \( u, v \) must be in \( V - S \)
  - Both \( u, v \) cannot be in \( S \)
  - Thus, \( S \) is an independent set. \( \blacksquare \)
Vertex Cover $\equiv_p$ IND Set

- $\text{VERTEX-COVER} \leq_p \text{IND-SET}$

- **Reduction.** Let $G' = G$, $k' = n - k$.
  
  - ($\Rightarrow$) If $G$ has a vertex cover of size at most $k$ then $G'$ has an independent set of size at least $k'$
  
  - ($\Leftarrow$) If $G'$ has an independent set of size at least $k'$ then $G$ has a vertex cover of size at most $k$

- $\text{IND-SET} \leq_p \text{VERTEX-COVER}$
  
  - Same reduction works: $G' = G$, $k' = n - k$

- $\text{VERTEX-COVER} \equiv_p \text{IND-SET}$
Acknowledgments

• Some of the material in these slides are taken from
  
  
  • Jeff Erickson’s Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)