POLYNOMIAL TRANSFORMATIONS

A subclass of poly-time reductions
commonly used for NP-completeness and impossibility results
POLYNOMIAL TRANSFORMATIONS

For a decision problem $\Pi$, let $\mathcal{I}(\Pi)$ denote the set of all instances of $\Pi$. Let $\mathcal{I}_{\text{yes}}(\Pi)$ and $\mathcal{I}_{\text{no}}(\Pi)$ denote the set of all yes-instances and no-instances (respectively) of $\Pi$.

Suppose that $\Pi_1$ and $\Pi_2$ are decision problems. We say that there is a polynomial transformation from $\Pi_1$ to $\Pi_2$ (denoted $\Pi_1 \leq_P \Pi_2$) if there exists a function $f : \mathcal{I}(\Pi_1) \to \mathcal{I}(\Pi_2)$ such that the following properties are satisfied:

- $f(I)$ is computable in polynomial time (as a function of $\text{size}(I)$, where $I \in \mathcal{I}(\Pi_1)$)
- if $I \in \mathcal{I}_{\text{yes}}(\Pi_1)$, then $f(I) \in \mathcal{I}_{\text{yes}}(\Pi_2)$
- if $I \in \mathcal{I}_{\text{no}}(\Pi_1)$, then $f(I) \in \mathcal{I}_{\text{no}}(\Pi_2)$

[Mechanics] to give a polynomial transformation, you must:
1. specify $f(I)$,
2. show it runs in poly-time, and
3. show $I$ is a yes-instance of $\Pi_1$ IFF $f(I)$ is a yes-instance of $\Pi_2$.

$\Pi_1$'s solution (true/false) is equivalent to $\Pi_2$'s solution

So, after transforming $\Pi_1$'s input, you can run a solution to $\Pi_2$ and just return the result!
Polynomial Transformations (Cont.)

A polynomial transformation can be thought of as a (simple) special case of a polynomial-time Turing reduction, i.e., if $\Pi_1 \leq_P \Pi_2$, then $\Pi_1 \leq_P \Pi_2$.

Given a polynomial transformation $f$ from $\Pi_1$ to $\Pi_2$, the corresponding Turing reduction is as follows:

- Given $I \in \mathcal{I}(\Pi_1)$, construct $f(I) \in \mathcal{I}(\Pi_2)$.
- Given an oracle for $\Pi_2$, say $A$, run $A(f(I))$.

We transform the instance, and then make a single call to the oracle.

Very important point: We do not know whether $I$ is a yes-instance or a no-instance of $\Pi_1$ when we transform it to an instance $f(I)$ of $\Pi_2$.

To prove the implication "if $I \in \mathcal{I}_{\text{no}}(\Pi_1)$, then $f(I) \in \mathcal{I}_{\text{no}}(\Pi_2)$", we usually prove the contrapositive statement "if $f(I) \in \mathcal{I}_{\text{yes}}(\Pi_2)$, then $I \in \mathcal{I}_{\text{yes}}(\Pi_1)$".

This can help when it is hard to precisely characterize certificates for no-instances (or when such certificates don't prove much).

Also known as Karp reductions and many-one reductions.

We haven't solved the problem yet, so we don't know much about the input...

We saw one instance where a contrapositive was easier to prove when we discussed Hamiltonian cycles.
SUMMARIZING
THE MORE CONVENIENT DEFINITION

○ Let \( \Pi_1 \) and \( \Pi_2 \) be decision problems

○ \( \Pi_1 \leq_P \Pi_2 \iff \) there exists \( f : I(\Pi_1) \to I(\Pi_2) \) such that:
  ○ \( f(I) \) is computable in poly-time, for all \( I \in I(\Pi_1) \)
  ○ If \( I \in I_{\text{yes}}(\Pi_1) \) then \( f(I) \in I_{\text{yes}}(\Pi_2) \)
  ○ If \( f(I) \in I_{\text{yes}}(\Pi_2) \) then \( I \in I_{\text{yes}}(\Pi_1) \)

This is the contrapositive. Was previously (2 slides ago):

\[
\text{If } I \in I_{\text{no}}(\Pi_1) \text{ then } f(I) \in I_{\text{no}}(\Pi_2)
\]

Note: this is the same as saying

\[
(I \in I_{\text{yes}}(\Pi_1)) \iff (f(I) \in I_{\text{yes}}(\Pi_2))
\]

This property justifies correctness for the following generic poly-time transformation code:

\[
P1toP2polyTransformation(I) \\
fI = f(I) \\
\text{return OracleP2}(fI)
\]
Problem 7.8

**Clique**

**Instance:** An undirected graph $G = (V, E)$ and an integer $k$, where $1 \leq k \leq |V|$.  

**Question:** Does $G$ contain a clique of size $\geq k$? (A **clique** is a subset of vertices $W \subseteq V$ such that $uv \in E$ for all $u, v \in W$, $u \neq v$.)

---

Problem 7.9

**Vertex Cover**

**Instance:** An undirected graph $G = (V, E)$ and an integer $k$, where $1 \leq k \leq |V|$.  

**Question:** Does $G$ contain a vertex cover of size $\leq k$? (A **vertex cover** is a subset of vertices $W \subseteq V$ such that $\{u, v\} \cap W \neq \emptyset$ for all edges $uv \in E$.)

---

*Every edge must touch a node in $W$*

*These $k$ nodes touch every edge in $G$*
CLIQUE $\leq_P$ VERTEX-COVER

- Suppose $I = (G, k)$ is an instance of Clique where $G = (V, E), V = \{v_1, ..., v_n\}$ and $1 \leq k \leq n$

Want to solve $\text{Clique}(G, k)$

Claim: there is a $k$-clique in $G$ iff there is an $(n - k)$ Vertex-Cover in $H$

- Construct instance $f(I) = (H, n - k)$ of Vertex-Cover, where $H = (V, F)$ and $v_i v_j \in F \iff v_i v_j \notin E$

Idea: reduce to $\text{VertexCover}(H, n - k)$

Consider the complement graph $H$ of $G$

Every edge of $G$ is a non-edge of $H$. Every non-edge of $G$ is an edge of $H$.

Given an adjacency matrix for $G$, get $H$ by flipping 0's and 1's.
PROVING THIS IS A POLYNOMIAL TRANSFORMATION

- We denote Clique by $CL$ and Vertex-Cover by $VC$
- $CL \leq_p VC$ if there exists $f : I(CL) \rightarrow I(VC)$ such that:
  - $f(I)$ is computable in poly-time, for all $I \in I(CL)$
  - If $I \in I_{yes}(CL)$ then $f(I) \in I_{yes}(VC)$
  - If $f(I) \in I_{yes}(VC)$ then $I \in I_{yes}(CL)$

First let’s show this
COMPLEXITY OF THE TRANSFORMATION

- Suppose $I = (G, k)$ is an instance of Clique where $G = (V, E), V = \{v_1, \ldots, v_n\}$ and $1 \leq k \leq n$

  \[
  \text{Want to solve } \text{Clique}(G, k)
  \]

- Construct instance $f(I) = (H, n - k)$ of Vertex-Cover, where $H = (V, F)$ and $v_i v_j \in F \iff v_i v_j \notin E$

  \[
  \text{Idea: reduce to } \text{VertexCover}(H, n - k)
  \]

Assuming adjacency matrix,
\[
\text{Size}(I) = \Theta(n^2 + \log_2 k)
\]

Time to compute $f(I)$?

Constructing $H$ takes $\Theta(n^2)$ time, and computing $n - k$ takes $\Theta(\log n)$ time.

So computing $f(I)$ takes $\Theta(n^2)$ time, which is polynomial in $\text{Size}(I)$.
PROVING THIS IS A POLYNOMIAL TRANSFORMATION

- We denote Clique by $CL$ and Vertex-Cover by $VC$

- $CL \leq_p VC$ iff there exists $f : I(CL) \rightarrow I(VC)$ such that:
  - $f(I)$ is computable in poly-time, for all $I \in I(CL)$
  - If $I \in I_{yes}(CL)$ then $f(I) \in I_{yes}(VC)$
  - If $f(I) \in I_{yes}(VC)$ then $I \in I_{yes}(CL)$

Now let's show this, i.e., if $G$ contains a $k$-clique then $H$ contains an $(n - k)$ vertex cover.
PROVING: $I \in I_{yes}(CL) \Rightarrow f(I) \in I_{yes}(VC)$

- Suppose $I = (G, k)$ is a yes-instance of Clique
- Then there is a set $W$ of $k$ vertices in a clique (with all-to-all edges)
- Define $W' = V \setminus W$. Clearly $|W'| = n - k$.
- We claim $W'$ is a vertex cover of $H$
- Consider any edge $(u, v) \in H$
- If either $u$ or $v$ is in $W'$, then we are done, so assume $u, v \notin W'$ to obtain a contradiction
- Then $u, v \in W$, and $W$ is a clique in $G$, so $(u, v) \in G$
- But $(u, v) \in H$ implies $(u, v) \notin G$. Contradiction!
PROVING THIS IS A POLYNOMIAL TRANSFORMATION

- We denote Clique by $CL$ and Vertex-Cover by $VC$

$CL \leq_p VC$ **iff** there exists $f : I(CL) \rightarrow I(VC)$ such that:

- $f(I)$ is computable in poly-time, for all $I \in I(CL)$
- If $I \in I_{yes}(CL)$ then $f(I) \in I_{yes}(VC)$
- If $f(I) \in I_{yes}(VC)$ then $I \in I_{yes}(CL)$

Now let’s show this, i.e., if $H$ contains an $(n - k)$ vertex cover, then $G$ contains a $k$-clique
PROVING: $f(I) \in I_{yes}(VC) \Rightarrow I \in I_{yes}(CL)$

- Suppose $f(I) = (H, n - k)$ is a yes-instance of $VC$
- Then there is a set of $n - k$ vertices $W'$ that is a vertex cover of $H$
- Define $W = V \setminus W'$. Clearly $|W| = k$.
- We claim $W$ is a clique in $G$
- Since $W'$ is a vertex cover of $H$, every edge in $H$ has at least one endpoint in $W'$
- Therefore, no edge in $H$ has two endpoints in $W$
- So, in $G$, there are edges between all pairs of nodes in $W$. So, $W$ is a clique in $G$.

So, we have demonstrated a polynomial transformation from CLIQUE to VERTEX-COVER.
Theorem 7.10

If $\Pi_1$ and $\Pi_2$ are decision problems, $\Pi_1 \leq_P \Pi_2$ and $\Pi_2 \in \mathbf{P}$, then $\Pi_1 \in \mathbf{P}$.

Proof.

Suppose $A$ is a poly-time algorithm for $\Pi_2$, having complexity $O(m^k)$ on an instance of size $m$. Suppose $f$ is a transformation from $\Pi_1$ to $\Pi_2$ having complexity $O(n^k)$ on an instance of size $n$. We solve $\Pi_1$ as follows:

1. Given $I \in \mathcal{I}(\Pi_1)$, construct $f(I) \in \mathcal{I}(\Pi_2)$.
2. Run $A(f(I))$.

It is clear that this yields the correct answer. We need to show that these two steps can be carried out in polynomial time as a function of $n = \text{Size}(I)$. Step (1) can be executed in time $O(n^k)$ and it yields an instance $f(I)$ having size $m \in O(n^k)$. Step (2) takes time $O(m^k)$. Since $m \in O(n^k)$, the time for step (2) is $O(n^{k\ell})$, as is the total time to execute both steps.
Theorem 7.11

Suppose that $\Pi_1$, $\Pi_2$ and $\Pi_3$ are decision problems. If $\Pi_1 \leq_P \Pi_2$ and $\Pi_2 \leq_P \Pi_3$, then $\Pi_1 \leq_P \Pi_3$.

Proof.

We have a polynomial transformation $f$ from $\Pi_1$ to $\Pi_2$, and another polynomial transformation $g$ from $\Pi_2$ to $\Pi_3$. We define $h = f \circ g$, i.e., $h(I) = g(f(I))$ for all instances $I$ of $\Pi_1$. (Exercise: fill in the details.)