COMP 355
Advanced Algorithms
NP-Completeness: Reductions
Chapter 8 (KT)
Section 34.2-34.3 (CLRS)
Recap

**Decision Problems/Language recognition:** are problems for which the answer is either yes or no. These can also be thought of as language recognition problems, assuming that the input has been encoded as a string. For example:

\[
\begin{align*}
\text{HC} & = \{ G \mid G \text{ has a Hamiltonian cycle} \} \\
\text{MST} & = \{ (G, c) \mid G \text{ has a MST of cost at most } c \}.
\end{align*}
\]

**P:** is the class of all decision problems which can be solved in polynomial time. While MST \(\in\) P, we do not know whether HC \(\in\) P (but we suspect not).

**Certificate:** is a piece of evidence that allows us to verify in polynomial time that a string is in a given language. For example, the language HC above, a certificate could be a sequence of vertices along the cycle. (If the string is not in the language, the certificate can be anything.)

**NP:** is defined to be the class of all languages that can be verified in polynomial time. (Formally, it stands for *Nondeterministic Polynomial time.*) Clearly, P \(\subseteq\) NP. It is widely believed that P \(\neq\) NP.
Algorithm Design Patterns and Anti-Patterns

Algorithm design patterns.
- Greed.
- Divide-and-conquer.
- Dynamic programming.
- Duality.
- Reductions.
- Local search.
- Randomization.

Ex.
- $O(n \log n)$ interval scheduling.
- $O(n \log n)$ FFT.
- $O(n^2)$ edit distance.
- $O(n^3)$ bipartite matching.

Algorithm design anti-patterns.
- NP-completeness.
- PSPACE-completeness.
- Undecidability.

- $O(n^k)$ algorithm unlikely.
- $O(n^k)$ certification algorithm unlikely.
- No algorithm possible.
Q. Which problems will we be able to solve in practice?


<table>
<thead>
<tr>
<th>Yes</th>
<th>Probably no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest path</td>
<td>Longest path</td>
</tr>
<tr>
<td>Matching</td>
<td>3D-matching</td>
</tr>
<tr>
<td>Min cut</td>
<td>Max cut</td>
</tr>
<tr>
<td>2-SAT</td>
<td>3-SAT</td>
</tr>
<tr>
<td>Planar 4-color</td>
<td>Planar 3-color</td>
</tr>
<tr>
<td>Bipartite vertex cover</td>
<td>Vertex cover</td>
</tr>
<tr>
<td>Primality testing</td>
<td>Factoring</td>
</tr>
</tbody>
</table>
Polynomial-Time Reduction

**Purpose.** Classify problems according to relative difficulty.

**Design algorithms.** If $X \leq_p Y$ and $Y$ can be solved in polynomial-time, then $X$ can also be solved in polynomial time.

**Establish intractability.** If $X \leq_p Y$ and $X$ cannot be solved in polynomial-time, then $Y$ cannot be solved in polynomial time.

**Establish equivalence.** If $X \leq_p Y$ and $Y \leq_p X$, we use notation $X \equiv_p Y$. 

up to cost of reduction
Desiderata'. Suppose we could solve X in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y if arbitrary instances of problem X can be solved using:
• Polynomial number of standard computational steps, plus
• Polynomial number of calls to oracle that solves problem Y.

Notation. $X \leq_p Y$.

Remarks.
• We pay for time to write down instances sent to black box $\Rightarrow$ instances of Y must be of polynomial size.
• Note: Cook reducibility.

Computational model supplemented by special piece of hardware that solves instances of Y in a single step

In contrast to Karp reductions
Suppose that there are two problems, H and U.

If we know that H is hard (cannot be solved in polynomial time), can we prove that U is also hard?

We effectively want to show that:
• \((H \not\in P) \Rightarrow (U \not\in P)\).

To do this, we could prove the contrapositive,
• \((U \in P) \Rightarrow (H \in P)\).

To show that U is not solvable in polynomial time, we will suppose (towards a contradiction) that a polynomial time algorithm for U did exist, and then we will use this algorithm to solve H in polynomial time, thus yielding a contradiction.
Suppose we have a subroutine that can solve any instance of problem $U$ in polynomial time.

Given an input $x$ for the problem $H$, we could translate it into an equivalent input $x'$ for $U$. (where $x \in H$ if and only if $x' \in U$)

Run subroutine on $x'$ and output whatever it outputs. It is easy to see that if $U$ is solvable in polynomial time, then so is $H$.

We assume that the translation module runs in polynomial time. If so, we say we have a polynomial reduction of problem $H$ to problem $U$, which is denoted $H \leq_p U$ (Karp reduction)
3-coloring (3Col): Given a graph $G$, can each of its vertices be labeled with one of three different “colors”, such that no two adjacent vertices have the same label (see (a) and (b)).

Clique Cover (CCov): Given a graph $G = (V, E)$ and an integer $k$, can we partition the vertex set into $k$ subsets of vertices $V_1, \ldots, V_k$ such that each $V_i$ is a clique of $G$. 
3-Colorability and Clique Cover

Reducing 3Col to CCov
Claim: A graph $G = (V, E)$ is 3-colorable if and only if its complement $G = (V, E)$ has a clique-cover of size 3. In other words, $G \in 3\text{Col} \iff (G, 3) \in \text{CCov}$.

Proof:

(⇒) If $G$ 3-colorable, then let $V_1, V_2, V_3$ be the three color classes. We claim that this is a clique cover of size 3 for $G$, since if $u$ and $v$ are distinct vertices in $V_i$, then $\{u, v\} / \in E$ (since adjacent vertices cannot have the same color) which implies that $\{u, v\} \in E$. Thus every pair of distinct vertices in $V_i$ are adjacent in $G$.

(⇐) Suppose $G$ has a clique cover of size 3, denoted $V_1, V_2, V_3$. For $i \in \{1, 2, 3\}$ give the vertices of $V_i$ color $i$. We assert that this is a legal coloring for $G$, since if distinct vertices $u$ and $v$ are both in $V_i$, then $\{u, v\} \in E$ (since they are in a common clique), implying that $\{u, v\} \in E$. Hence, two vertices with the same color are not adjacent.
**Definition:** We say that a language (i.e. decision problem) $L_1$ is polynomial-time reducible to language $L_2$ (written $L_1 \leq_P L_2$) if there is a polynomial time computable function $f$, such that for all $x$, $x \in L_1$ if and only if $f(x) \in L_2$.

**Lemma:** If $L_1 \leq_P L_2$ and $L_2 \in P$ then $L_1 \in P$.

**Lemma:** If $L_1 \leq_P L_2$ and $L_1 \not\in P$ then $L_2 \not\in P$.

Because the composition of two polynomials is a polynomial, we can chain reductions together.

**Lemma:** If $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$ then $L_1 \leq_P L_3$.
Definition: A language \( L \) is NP-hard if \( L' \leq_p L \), for all \( L' \in \text{NP} \). (Note that \( L \) does not need to be in \text{NP}.)

Definition: A language \( L \) is NP-complete if:
1. \( L \in \text{NP} \) (that is, it can be verified in polynomial time), and
2. \( L \) is NP-hard (that is, every problem in \text{NP} is polynomially reducible to it).

Lemma: \( L \) is NP-complete if
1. \( L \in \text{NP} \) and
2. \( L' \leq_p L \) for some known NP-complete language \( L' \).
Structure of NPC and reductions

All problems in NP are reducible to SAT

If SAT ≤_p X then X is NP-hard

If Y ∈ NP and SAT ≤_p Y then Y is NP-complete
Next Time

NP-Completeness: Reductions