Lecture 22 More on Reductions and NP-Completeness

Before we give a formal definition of reduction, let us clarify the notion of a *decision problem*. Informally, a decision problem is a yes-or-no question. A decision problem is given by a description of the *problem domain*, *i.e.* the set of all possible instances of the problem, along with a description of the set of "yes" instances.

For example, consider the problem of determining whether a given undirected graph G has a k-clique. An instance of the problem is a pair (G, k), and the problem domain is the set of all such pairs. The "yes" instances are the pairs (G, k) for which G has a clique of size k.

There are many interesting discrete problems that are not decision problems. For example, many optimization problems like the traveling salesman problem or the integer programming problem ask for the calculation of an object that maximizes some objective function. However, many of these problems have closely related decision problems that are no simpler to solve than the optimization problem. For the purposes of this discussion of reductions and NP-completeness, we will restrict our attention to decision problems.

Definition 22.1 Let $A \subseteq \Sigma$ and $B \subseteq \Gamma$ be decision problems. (Here Σ and Γ are the problem domains, and A and B are the "yes" instances.) We write $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ and say that A *reduces to* B *in polynomial time* if there is a function $\sigma : \Sigma \to \Gamma$ such that

- σ is computable by a deterministic Turing machine in polynomial time;

- for all problem instances $x \in \Sigma$,

$$x \in A \quad \text{iff} \quad \sigma(x) \in B \ .$$

We write $A \equiv_{\mathrm{m}}^{\mathrm{p}} B$ if both $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ and $B \leq_{\mathrm{m}}^{\mathrm{p}} A$. □

The reducibility relation $\leq_{\mathrm{m}}^{\mathrm{p}}$ is often called *polynomial-time many-one* or *Karp* reducibility. The superscript p stands for *polynomial-time*. The subscript m stands for *many-one* and describes the function σ, and is included to distinguish $\leq_{\mathrm{m}}^{\mathrm{p}}$ from another popular polynomial-time reducibility relation $\leq_{\mathrm{T}}^{\mathrm{p}}$, often called *polynomial-time Turing* or *Cook* reducibility. The relation $\leq_{\mathrm{m}}^{\mathrm{p}}$ is stronger than $\leq_{\mathrm{T}}^{\mathrm{p}}$ in the sense that

$$A \leq_{\mathrm{m}}^{\mathrm{p}} B \quad \rightarrow \quad A \leq_{\mathrm{T}}^{\mathrm{p}} B \ .$$

The formal definition of $\leq_{\mathrm{T}}^{\mathrm{p}}$ involves oracle Turing machines and can be found in [39, pp. 111ff.].

Intuitively, if $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ then A is no harder than B. In particular,

Theorem 22.2 *If $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ and B has a polynomial-time algorithm, then so does A.*

Proof. Given an instance x of the problem A, compute $\sigma(x)$ and ask whether $\sigma(x) \in B$. Note that the algorithm for B runs in polynomial time in the size of its input $\sigma(x)$, which might be bigger than x; but since σ is computable in polynomial time on a Turing machine, the size of $\sigma(x)$ is at most polynomial in the size of x, and the composition of two polynomials is still a polynomial, so the overall algorithm is polynomial in the size of x. □

In the last lecture we showed that CNFSat $\equiv_{\mathrm{m}}^{\mathrm{p}}$ Clique. Below we give some more examples of polynomial-time reductions between problems.

Definition 22.3 (Independent Set) An *independent set* in an undirected graph $G = (V, E)$ is a subset U of V such that $U^2 \cap E = \emptyset$, *i.e.* no two vertices in U are connected by an edge in E. The *independent set problem* is to determine, given $G = (V, E)$ and $k \geq 0$, whether G has an independent set U of cardinality at least k. □

Note that the use of "independent" here is *not* in the sense of matroids.

There exist easy polynomial reductions from/to the clique problem. Consider the complementary graph $\overline{G} = (V, \overline{E})$, where

$$\overline{E} \ = \ \{(u, v) \mid u \neq v, \ (u, v) \notin E\} \ .$$

Then G has a clique of size k iff $\overline{G}$ has an independent set of size k. This simple one-to-one correspondence gives reductions in both directions, therefore Independent Set $\equiv_{\mathrm{m}}^{\mathrm{p}}$ Clique.

Definition 22.4 (Vertex Cover) A *vertex cover* in an undirected graph $G = (V, E)$ is a set of vertices $U \subseteq V$ such that every edge in E is adjacent to some vertex in U. The *vertex cover problem* is to determine, given $G = (V, E)$ and $k \geq 0$, whether there exists a vertex cover U in G of cardinality at most k. □

Again, there exist easy polynomial reductions from/to Independent Set: $U \subseteq V$ is a vertex cover iff $V - U$ is an independent set. Therefore Vertex Cover $\equiv_{\mathrm{m}}^{\mathrm{p}}$ Independent Set.

Definition 22.5 (k-CNFSat) A Boolean formula is in *k-conjunctive normal form* (*k-CNF*) if it is in conjuctive normal form and has at most k literals per clause. The problem k-CNFSat is just CNFSat with input instances restricted to formulas in k-CNF. In other words, given a Boolean formula in k-CNF, does it have a satisfying assignment? □

In the general CNFSat problem, the number of literals per clause is not restricted and can grow as much as linearly with the size of the formula. In the k-CNFSat problem, the number of literals per clause is restricted to k, independent of the size of the formula. The k-CNFSat problem is therefore a restriction of the CNFSat problem, and could conceivably be easier to solve than CNFSat. It turns out that 2CNFSat (and hence 1CNFSat also) is solvable in linear time, whereas k-CNFSat is as hard as CNFSat for any $k \geq 3$. We prove the latter statement by exhibiting a reduction CNFSat $\leq_{\mathrm{m}}^{\mathrm{p}}$ 3CNFSat.

Let $\mathcal{B}$ be an arbitrary Boolean formula in CNF. For each clause of the form

$$(\ell_1 \vee \ell_2 \vee \cdots \vee \ell_{m-1} \vee \ell_m) \tag{27}$$

with $m \geq 4$, let $x_1, x_2, \ldots, x_{m-3}$ be new variables and replace the clause (27) in $\mathcal{B}$ with the formula

$$\begin{aligned}(\ell_1 \vee \ell_2 \vee x_1) \wedge (\neg x_1 \vee \ell_3 \vee x_2) \wedge (\neg x_2 \vee \ell_4 \vee x_3) \wedge \cdots \\ \wedge (\neg x_{m-4} \vee \ell_{m-2} \vee x_{m-3}) \wedge (\neg x_{m-3} \vee \ell_{m-1} \vee \ell_m)\ .\end{aligned}$$

Let $\mathcal{B}'$ be the resulting formula. Then $\mathcal{B}'$ is in 3CNF, and $\mathcal{B}'$ is satisfiable iff $\mathcal{B}$ is. This follows from several applications of the following lemma:

Lemma 22.6 *For any Boolean formulas $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$ and variable x not appearing in $\mathcal{C}, \mathcal{D}$, or $\mathcal{E}$, the formula*

$$(x \vee \mathcal{C}) \wedge (\neg x \vee \mathcal{D}) \wedge \mathcal{E} \tag{28}$$

is satisfiable if and only if the formula

$$(\mathcal{C} \vee \mathcal{D}) \wedge \mathcal{E} \tag{29}$$

is satisfiable.

Proof. This is just the *resolution rule* of propositional logic. Any satisfying truth assignment for (28) gives a satisfying truth assignment for (29), since one of x, $\neg x$ is false, so either $\mathcal{C}$ or $\mathcal{D}$ is true. Conversely, in any satisfying truth assignment for (29), one of $\mathcal{C}, \mathcal{D}$ is true. If $\mathcal{C}$, assign $x := \mathit{false}$. If $\mathcal{D}$, assign $x := \mathit{true}$. We can assign x freely since it does not appear in $\mathcal{C}, \mathcal{D}$ or $\mathcal{E}$. In either case (28) is satisfied. □

The formula $\mathcal{B}'$ is easily constructed from $\mathcal{B}$ in polynomial time. This constitutes a polynomial-time reduction from CNFSat to 3CNFSat. Furthermore, 3CNFSat is trivially reducible to k-CNFSat for any $k \geq 3$, which in turn is trivially reducible to CNFSat. Since $\leq_{\mathrm{m}}^{\mathrm{p}}$ is transitive, k-CNFSat $\equiv_{\mathrm{m}}^{\mathrm{p}}$ CNFSat for $k \geq 3$.

The problem 2CNFSat is solvable in linear time. In this case the clauses in $\mathcal{B}$ contain at most two literals, and we can assume exactly two without loss of generality by replacing any clause of the form (ℓ) with $(\ell \vee \ell)$. Now we think of every two-literal clause $(\ell \vee \ell')$ as a pair of implications

$$(\neg\ell \to \ell') \quad \text{and} \quad (\neg\ell' \to \ell) \ . \tag{30}$$

Construct a directed graph $G = (V, E)$ with a vertex for every literal and directed edges corresponding to the implications (30).

We claim that $\mathcal{B}$ is satisfiable iff no pair of complementary literals both appear in the same strongly connected component of G. Under any satisfying truth assignment, all literals in a strong component of G must have the same truth value. Therefore, if any variable x appears both positively and negatively in the same strong component of G, $\mathcal{B}$ is not satisfiable.

Conversely, suppose that no pair of complementary literals both appear in the same strong component of G. Consider the quotient graph G' obtained by collapsing the strong components of G as described in Lecture 4. As proved in that lecture, the graph G' is acyclic, therefore induces a partial order on its vertices. This partial order extends to a total order. We assign $x := \mathit{false}$ if the strong component of x occurs before the strong component of $\neg x$ in this total order, and $x := \mathit{true}$ if the strong component of $\neg x$ occurs before the strong component of x. It can be shown that this gives a satisfying assignment.

We know how to find the strong components of G in linear time. This gives a linear-time algorithm test for 2CNF satisfiability. We can also produce a satisfying assignment in linear time, if one exists, using topological sort to totally order the strong components.

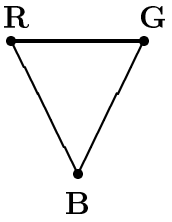
Definition 22.7 (k-Colorability) Let C a finite set of *colors* and $G = (V, E)$ an undirected graph. A *coloring* is a map $\chi : V \to C$ such that $\chi(u) \neq \chi(v)$ for $(u, v) \in E$. Given G and k, the *k-colorability problem* is to determine whether there exists a coloring using no more than k colors. □

For $k = 2$, the problem is easy: a graph is 2-colorable iff it is bipartite iff it has no odd cycles. This can be checked by BFS or DFS in linear time. We

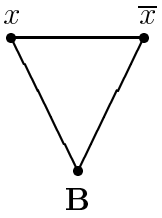
show that for $k = 3$, the problem is as hard as CNFSat by giving a reduction CNFSat $\leq_{\mathrm{m}}^{\mathrm{p}}$ 3-colorability.

Let $\mathcal{B}$ be a Boolean formula in CNF. We will construct a graph G that is 3-colorable iff $\mathcal{B}$ is satisfiable.

There will be three special vertices called **R**, **B**, and **G**, which will be connected in a triangle. In any 3-coloring, they will have to be colored with different colors, so we assume without loss of generality that they are colored red, blue, and green, respectively.

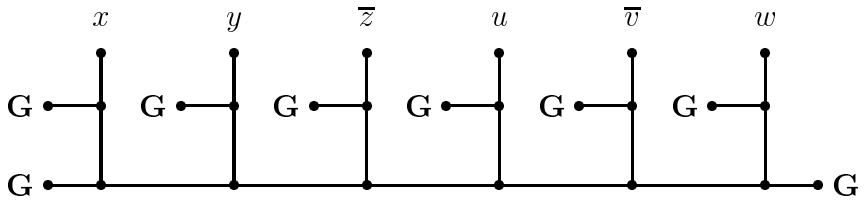


We include a vertex for each literal, and connect each literal to its complement and to the vertex **B** as shown.



In any 3-coloring, the vertices corresponding to the literals x and $\overline{x}$ will have to be colored either red or green, and not both red or both green. Intuitively, a legal 3-coloring will represent a satisfying truth assignment in which the green literals are true and the red literals are false.

To complete the graph, we add a subgraph like the one shown below for each clause in $\mathcal{B}$. The one shown below would be added for the clause $(x \vee y \vee \overline{z} \vee u \vee \overline{v} \vee w)$. The vertices in the picture labeled **G** are all the same vertex, namely the vertex **G**.



This subgraph has the property that a coloring of the vertices on the top row with either red or green can be extended to a 3-coloring of the whole subgraph iff at least one of them is colored green. If all vertices on the top row are colored red, then all the vertices on the middle row adjacent to vertices on the top row must be colored blue. Starting from the left, the vertices along the bottom row must be colored alternately red and green. This will lead to

a conflict with the last vertex in the bottom row. (If the number of literals in the clause is odd instead of even as pictured, then the rightmost vertex in the bottom row is **R** instead of **G**.)

Conversely, suppose one of the vertices on the top row is colored green. Pick one such vertex. Color the vertex directly below it in the middle row red and the vertex directly below that on the bottom row blue. Color all other vertices on the middle row blue. Starting from the left and right ends, color the vertices along the bottom row as forced, either red or green. The coloring can always be completed.

Thus if there is a legal 3-coloring, then the subgraph corresponding to each clause must have at least one green literal, and truth values can be assigned so that the green literals are true. This gives a satisfying assignment. Conversely, if there is a satisfying assignment, color the true variables green and the false ones red. Then there is a green literal in each clause, so the coloring can be extended to a 3-coloring of the whole graph.

From this it follows that $\mathcal{B}$ is satisfiable iff G is 3-colorable, and the graph G can be constructed in polynomial time. Therefore CNFSat $\leq_{\mathrm{m}}^{\mathrm{p}}$ 3-colorability.

One can trivially reduce 3-colorability to k-colorability for $k > 3$ by appending a $k-3$ clique and edges from every vertex of the $k-3$ clique to every other vertex.

One may be tempted to conclude that in problems like k-CNFSat and k-colorability, larger values of k always make the problem harder. On the contrary, we shall see in the next lecture that the k-colorability problem for planar graphs is easy for $k \leq 2$ and $k \geq 4$, but as hard as CNFSat for $k = 3$.