9  NP-Hardness

9.1 Definitions

We will give a pretty informal definition of when an optimization problem is NP-hard. To
give a more formal version would mean considerably more work, with little extra benefit for
our purpose here.

NP-hardness helps us classify which optimization problems are hard in the sense that there is
no polynomial algorithm for them (unless \( P = NP \)), and which ones are the hardest among
these. To make sense of this, we will first have to go through several other definitions.

- **Decision problems**: We have to distinguish between decision problems (yes/no questions,
like “is there a perfect matching in a graph”) and general computational problems, which
include optimization problems (“find the maximum weight perfect matching”). The notions
of \( NP \) and \( NP \)-completeness are only defined for decision problems, while \( NP \)-hardness
will be defined for optimization problems.

- **Definitions**: A decision problem is
  
  - in \( P \) if there is an algorithm that solves any instance in polynomial time;
  
  - in \( NP \) if, when the answer for an instance is YES, there is a certificate proving it that
can be checked in polynomial time;
  
  - \( NP \)-complete if any problem in \( NP \) can be polynomially reduced to it.

- **\( NP \)**: \( NP \) does not stand for “non-polynomial” (and does not exactly mean that), but for
  “non-deterministically polynomial”; very roughly, this means that you (actually, a Turing
  machine) could “non-deterministically” guess an answer, and then check it in polynomial
time.
  
  Most important for us is that if a decision problem is not in \( NP \), then there cannot be a
  polynomial algorithm for it, since that would provide a way of polynomially checking an
  answer. In other words, \( P \subseteq NP \), and the million-dollar question is whether \( P = NP \) or
  \( P \neq NP \).

- **Reductions**: We say that problem A reduces polynomially to problem B if any instance
  of A can be transformed to an instance of B, by a process that takes polynomial time.
  What this comes down to is that if we have a polynomial algorithm for problem B, then this
  transformation also gives a polynomial algorithm for A. So if one had a polynomial algorithm
  for one \( NP \)-complete decision problem, then one would have polynomial algorithms for all
  problems in \( NP \).
  
  Below we will leave out the word “polynomially”, since the reductions that we will consider
  are clearly polynomial.

 Optimization problems: An optimization problem (or a computational problem) is

  - \( NP \)-hard if some \( NP \)-complete problem can be polynomially reduced to it.

This implies that if a problem is \( NP \)-hard, then all \( NP \) problems can be reduced to it. Note
that a decision problem is also a computational problem, so a decision problem is \( NP \)-hard
if and only if it is \( NP \)-complete.
9.2 $\mathcal{NP}$-hard problems

We will show for a number of standard optimization problems that they are $\mathcal{NP}$-hard. This is usually done by proving that all $\mathcal{NP}$ problems reduce to the decision problem SAT, and then, for a given optimization problem X, reducing SAT to X, perhaps indirectly. This implies that X is $\mathcal{NP}$-hard, since if there was a polynomial algorithm for X, then that would give one for SAT, which would give a polynomial algorithm for every other $\mathcal{NP}$ problem.

SAT asks if a logical formula is satisfiable, i.e. if there is an assignment of true/false to the variables such that the formula comes out true. But introducing SAT in full detail would be too much of a digression for us, since it has more to do with logic than with optimization, and proving that it is $\mathcal{NP}$-complete would also require more formal definitions. For a proof, see for instance Schrijver’s notes, 6.8.

So instead we will start by assuming that one decision problem, $k$-Vertex Cover, is $\mathcal{NP}$-hard, and then use that to prove it for other problems. If one knows SAT, then it is not too hard to prove that SAT reduces to $k$-Vertex Cover; see for instance Erickson’s notes, 29.9.

Here is a diagram of the reductions that we will prove in this lecture. An arrow from A to B means that we will reduce A to B, which means that if there is a polynomial algorithm for B, then there is also one for A.

Recall that a vertex cover is a $C \subset V(G)$ such that $e \cap C \neq \emptyset$ for all $e \in E(G)$.

**$k$-Vertex Cover:** Given an undirected graph $G$, decide whether there is a vertex cover with $\leq k$ vertices.

**Vertex Cover:** Given an undirected graph $G$, find a minimum cardinality vertex cover.

**Theorem 9.1.** $k$-Vertex Cover is $\mathcal{NP}$-complete and Vertex Cover is $\mathcal{NP}$-hard.

**Proof.** The first can be deduced from the $\mathcal{NP}$-completeness of SAT as mentioned above.

The second follows from the first by reducing $k$-Vertex Cover to Vertex Cover. This is easy, since given $G$ and $k$, we can use Vertex Cover to get a minimum vertex cover $C$, and we just have to check if $|C| \leq k$.  

\end{proof}
SET COVER: Given a finite set $X$, a set of subsets $S = \{S_i\}_{i \in I}$ of $X$, and $c: S \to \mathbb{R}_{>0}$, find $J \subset I$ such that

$$\bigcup_{j \in J} S_j = X$$

with minimum $c(J) = \sum_{j \in J} c(S_j)$.

**Theorem 9.2.** Set Cover is $\mathcal{NP}$-hard.

*Proof.* Vertex Cover reduces to Set Cover as follows. Given $G$, let $X = E(G)$ and $S = \{S_v\}_{v \in V(G)}$ with $S_v = \delta(v)$. Then for $c$ defined by $c(S_v) = 1$, a set cover for $X$ is a vertex cover for $G$, with the same weight. Indeed, it is a $J \subset I = V(G)$ such that $\cup_{v \in J} \delta(v) = E(G)$, which implies that $e \cap J \neq \emptyset$ for all $e \in E(G)$, and we have $c(J) = |J|$. Hence a minimum weight set cover for $X$ corresponds exactly to a minimum cardinality vertex cover in $G$. □

STEINER TREE: Given an undirected graph $G$, weights $w: E(G) \to \mathbb{R}_{>0}$, and $S \subset V(G)$, find a minimum-weight tree $T$ in $G$ such that $S \subset V(T)$.

**Theorem 9.3.** Steiner Tree is $\mathcal{NP}$-hard.

*Proof.* We reduce Vertex Cover to Steiner Tree as follows. Given a graph $G$ in which to find a vertex cover, we define a new graph $H$ in which to find a Steiner tree. Let $V(H) = V(G) \cup E(G)$ and

$$E(H) = \{uv : u, v \in V(G)\} \cup \{ve : v \in e \in E(G)\}.$$  

In other words, we take the complete graph on $V(G)$, add a vertex for every $e \in E(G)$, and connect a $v$ in the complete graph to the vertex $e$ if $v \in e$ in $G$.

Let $w(e) = 1$ for all $e \in E(H)$ and $S = E(G) \subset V(H)$. We show that a Steiner tree $T$ for $S$ in $H$ corresponds to a vertex cover $C$ in $G$ with $w(T) = |C| + |S| - 1$, hence a minimum Steiner tree corresponds to a minimum vertex cover.

Let $T$ be a Steiner tree for $S$ in $H$. If we take $C = V(T) \setminus S$, then $C \subset V(G)$ is a vertex cover in $G$. Indeed, any $e = uv \in E(G)$ forms a vertex in $S$, which is only connected to the vertices $u$ and $v$, so one of these must be in $V(T) \setminus S = C$, and it covers $e$. The vertex cover $C$ has size $|C| = |V(T) \setminus S| = w(T) + 1 - |S|$, since $w(T) = |E(T)| = |V(T)| - 1$ because $T$ is a tree on $V(T)$.

Let $C$ be a vertex cover in $G$. Then as a subset of the complete graph part of $H$, there is a tree on $C$ with $|C| - 1$ edges. Add to this tree an edge to each $e \in S = E(G)$, using as few edges as possible. This is possible because $C$ is a cover, and it gives a Steiner tree for $S$. It has weight $w(T) = |C| - 1 + |S|$. □
**Theorem 9.4.** **Directed Hamilton Cycle** is $\mathcal{NP}$-hard (so also $\mathcal{NP}$-complete).

*Proof.* We reduce $k$-Vertex Cover to Directed Hamilton Cycle. Let $G$ be a graph for which we want to know if there is a Vertex Cover with $\leq k$ vertices. Given $uv \in E(G)$, we’ll use the following building block $H_{uv}$:

For each vertex $u \in V(G)$, we connect the $H_{uv}$ into a chain as follows. Let $\delta(u) = \{uv_1, \ldots, uv_m\}$, and for each $i$, connect vertex $b_u$ of $H_{uv_i}$ to vertex $a_u$ of $H_{uv_{i+1}}$. So for each vertex we have a chain of $H_{uv}$, and each $H_{uv}$ is in two different chains, one for $u$ and one for $v$.

Next we add a set $K$ of $k$ extra vertices, with for each vertex $u$ an edge from each $x \in K$ to each first vertex $a_u$ of the chain for $u$, and an edge from each last vertex $b_u$ of the chain for $u$. We call the resulting graph $H$.

Now suppose there is a Hamilton cycle in $H$. It must pass through each $H_{uv}$, and it can do that in 3 ways: $a_u a_v b_u b_v$, $a_u a_v b_u b_v$, or separately through $a_u b_u$ and $a_v b_v$ (if it went any other way, there would be vertices left that it can never visit). Either way, if it enters at $a_u$ then it leaves at $b_u$, so it must stay within the chain for $u$. It must enter the chain from some $x \in K$, and leave to some other $y \in K$.

Let $C$ be the set of $k$ vertices whose chain the Hamilton cycle goes through entirely. Then $C$ is a vertex cover: Given an edge $uv \in E(G)$, the Hamilton cycle must pass through $H_{uv}$ somehow, so it must use the chain for $u$ or $v$ (or both), so at least one of those is in $C$.

Conversely, if $C$ is a vertex cover in $G$, then we can construct a Hamilton cycle in $H$ from it. Write $C = \{u_1, \ldots, u_k\}$ and $K = \{x_1, \ldots, x_k\}$. From $x_1$ we go through the chain for $u_1$, out of the chain we go to $x_2$, then through the chain for $u_2$, etc, until we go through the chain for $u_k$, out of which we go back to $x_1$. For each $H_{uv}$, we go $a_u b_u$ if $v$ is also in the cover, and $a_u a_v b_v b_u$ if not. 

\[ \square \]
Hamilton Cycle: Given an undirected graph $G$, decide whether it has a Hamilton cycle, i.e. a cycle $C$ such that $V(C) = V(G)$.

**Theorem 9.5.** Hamilton Cycle is $\mathcal{NP}$-hard (so also $\mathcal{NP}$-complete).

**Proof.** We reduce Directed Hamilton Cycle to Hamilton Cycle. Given $D$ in which to decide on the existence of a directed Hamilton cycle, for each $v \in V(D)$ take 3 vertices $v_1, v_2, v_3$, with 2 edges $v_1v_2$ and $v_2v_3$. Then for each directed edge $uv \in E(D)$, take an undirected edge $u_3v_1$. Call this undirected graph $G$. Now a directed Hamilton cycle in $D$ corresponds exactly to a Hamilton cycle in $G$ (possibly in reverse).

Travelling Salesman Problem: Given an undirected complete graph $G$, with weights $w : E(G) \to \mathbb{R}_{\geq 0}$, find a Hamilton cycle of minimum weight.

**Theorem 9.6.** The Travelling Salesman Problem is $\mathcal{NP}$-hard.

**Proof.** We reduce Hamilton Cycle to Travelling Salesman Problem. Given a graph $G$ in which to decide if there is a Hamilton cycle, give all edges weight 1, then add edges of weight 10 between any unconnected vertices. Then there is a Hamilton cycle in $G$ if and only if the minimum weight Hamilton cycle in the new weighted graph has weight $|V(G)|$, since then it used only edge from $G$. 
