23 More NP-Hard Problems (April 26)

In this lecture, I'll describe some more NP-hardness reductions, mostly involving graphs.

23.1 Independent Set (from Clique)

An independent set is a collection of vertices in a graph with no edges between them. The \textsc{IndependentSet} problem is to find the largest independent set in a given graph.

There is an easy proof that \textsc{IndependentSet} is NP-hard, using a reduction from \textsc{Clique}. Any graph $G$ has a complement $\overline{G}$ with the same vertices, but with exactly the opposite set of edges—$(u,v)$ is an edge in $\overline{G}$ if and only if it is not an edge in $G$. A set of vertices forms a clique in $G$ if and only if the same vertices are an independent set in $\overline{G}$. Thus, we can compute the largest clique in a graph simply by computing the largest independent set in the complement of the graph.

\[
\begin{array}{c}
\text{graph } G \\
\mapsto
\text{complement graph } \overline{G} \\
\downarrow \text{\textsc{IndependentSet}}
\end{array}
\]

\[
\begin{array}{c}
\text{largest clique} \\
\mapsto
\text{largest independent set}
\end{array}
\]

23.2 Vertex Cover (from Independent Set)

A vertex cover of a graph is a set of vertices that touches every edge in the graph. The \textsc{VertexCover} problem is to find the smallest vertex cover in a given graph.

Again, the proof of NP-hardness is simple, and relies on just one fact: If $I$ is an independent set in a graph $G = (V, E)$, then $V \setminus I$ is a vertex cover. Thus, to find the largest independent set, we just need to find the vertices that aren't in the smallest vertex cover of the same graph.

\[
\begin{array}{c}
\text{graph } G = (V, E) \\
\mapsto
\text{trivial} \\
\downarrow \text{\textsc{VertexCover}}
\end{array}
\]

\[
\begin{array}{c}
\text{largest independent set } V \setminus S \\
\mapsto
\text{smallest vertex cover } S
\end{array}
\]

23.3 Graph Coloring (from 3SAT)

A c-coloring of a graph is a map $C : V \to \{1, 2, \ldots, c\}$ that assigns one of $c$ 'colors' to each vertex, so that every edge has two different colors at its endpoints. The graph coloring problem is to find the smallest possible number of colors in a legal coloring. To show that this problem is NP-hard, it's enough to consider the special case 3\textsc{Colorable}: Given a graph, does it have a 3-coloring?

To prove that 3\textsc{Colorable} is NP-hard, we use a reduction from 3\textsc{Sat}. Given a 3CNF formula, we produce a graph as follows. The graph consists of a truth gadget, one variable gadget for each variable in the formula, and one clause gadget for each clause in the formula.

The truth gadget is just a triangle with three vertices $T$, $F$, and $X$, which intuitively stand for TRUE, FALSE, and OTHER. Since these vertices are all connected, they must have different colors in any 3-coloring. For the sake of convenience, we will name those colors TRUE, FALSE, and OTHER. Thus, when we say that a node is colored TRUE, all we mean is that it must be colored the same as the node $T$. 1
The variable gadget for a variable \( \alpha \) is also a triangle joining two new nodes labeled \( \alpha \) and \( \overline{\alpha} \) to node \( X \) in the truth gadget. Node \( \alpha \) must be colored either TRUE or FALSE, and so node \( \overline{\alpha} \) must be colored either FALSE or TRUE, respectively.

Finally, each clause gadget joins three literal nodes to node \( T \) in the truth gadget using five new unlabeled nodes and ten edges; see the figure below. If all three literal nodes in the clause gadget are colored FALSE, then the rightmost vertex in the gadget cannot have one of the three colors. Since the variable gadgets force each literal node to be colored either TRUE or FALSE, in any valid 3-coloring, at least one of the three literal nodes is colored TRUE. I need to emphasize here that the final graph contains only one node \( T \), only one node \( F \), only one node \( \overline{T} \) for each variable \( \alpha \), and so on.

For example, the formula \((a \lor b \lor c) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor c \lor d) \land (a \lor \overline{b} \lor \overline{d})\) that I used to illustrate the MAXCLIQUE reduction would be transformed into the following graph. The 3-coloring is one of several that correspond to the satisfying assignment \( a = c = \text{TRUE}, b = d = \text{FALSE} \).

The proof of correctness is just brute force. If the graph is 3-colorable, then we can extract a satisfying assignment from any 3-coloring—at least one of the three literal nodes in every clause gadget is colored TRUE. Conversely, if the formula is satisfiable, then we can color the graph according to any satisfying assignment.
We can easily verify that a graph has been correctly 3-colored in linear time: just compare the endpoints of every edge. Thus, 3COLORING is in NP, and therefore NP-complete. Moreover, since 3COLORING is a special case of the more general graph coloring problem—What is the minimum number of colors?—the more problem is also NP-hard, but not NP-complete, because it's not a yes/no problem.

### 23.4 Hamiltonian Cycle (from Vertex Cover)

A Hamiltonian cycle is a graph is a cycle that visits every vertex exactly once. This is very different from an Eulerian cycle, which is actually a closed walk that traverses every edge exactly once. Eulerian cycles are easy to find and construct in linear time using a variant of depth-first search. Finding Hamiltonian cycles, on the other hand, is NP-hard.

To prove this, we use a reduction from the vertex cover problem. Given a graph \( G \) and an integer \( k \), we need to transform it into another graph \( G' \), such that \( G' \) has a Hamiltonian cycle if and only if \( G \) has a vertex cover of size \( k \). As usual, our transformation consists of putting together several gadgets.

- For each edge \((u, v)\) in \( G \), we have an edge gadget in \( G' \) consisting of twelve vertices and fourteen edges, as shown below. The four corner vertices \((u, v, 1)\), \((u, v, 6)\), \((v, u, 1)\), and \((v, u, 6)\) each have an edge leaving the gadget. A Hamiltonian cycle can only pass through an edge gadget in one of three ways. Eventually, these will correspond to one or both of the vertices \( u \) and \( v \) being in the vertex cover.

- \( G' \) also contains \( k \) cover vertices, simply numbered 1 through \( k \).

- Finally, for each vertex \( u \) in \( G \), we string together all the edge gadgets for edges \((u, v)\) into a single vertex chain, and then connect the ends of the chain to all the cover vertices. Specifically, suppose \( u \) has \( d \) neighbors \( v_1, v_2, \ldots, v_d \). Then \( G' \) has \( d - 1 \) edges between \((u, v_1, 6)\) and \((u, v_{i+1}, 1)\), plus \( k \) edges between the cover vertices and \((u, v_1, 1)\), and finally \( k \) edges between the cover vertices and \((u, v_d, 6)\).
It's not hard to prove that if \( \{v_1, v_2, \ldots, v_k\} \) is a vertex cover of \( G \), then \( G' \) has a Hamiltonian cycle—start at cover vertex 1, through traverse the vertex chain for \( v_1 \), then visit cover vertex 2, then traverse the vertex chain for \( v_2 \), and so forth, eventually returning to cover vertex 1. Conversely, any Hamiltonian cycle in \( G' \) alternates between cover vertices and vertex chains, and the vertex chains correspond to the \( k \) vertices in a vertex cover of \( G \). (This is a little harder to prove.) Thus, \( G \) has a vertex cover of size \( k \) if and only if \( G' \) has a Hamiltonian cycle.

The transformation from \( G \) to \( G' \) takes at most \( O(n^2) \) time, so the Hamiltonian cycle problem is NP-hard. Moreover, since we can easily verify a Hamiltonian cycle in linear time, the Hamiltonian cycle problem is in NP, and therefore NP-complete.

A closely related problem to Hamiltonian cycles is the famous \textit{traveling salesman problem}—Given a \textit{weighted} graph \( G \), find the shortest cycle that visits every vertex. Finding the shortest cycle is obviously harder than determining if a cycle exists at all, so the traveling salesman problem is also NP-hard.
23.5 Minesweeper (from Circuit SAT)

In 1999, Richard Kaye proved that the solitaire game Minesweeper is NP-complete, using a reduction from the original circuit satisfiability problem.\footnote{Minesweeper is NP-complete. Mathematical Intelligencer 22(2):9–15, 2000.} The reduction involves setting up gadgets for every possible feature of a boolean circuit: wires, AND gates, OR gates, NOT gates, wire crossings, and so forth. For all the gory details, see http://www.mat.bham.ac.uk/R.W.Kaye/minesw/minesw.pdf!