Lecture 24 Still More NP-Complete Problems

In this lecture we use the basic NP-complete problems given in previous lectures, which may have appeared contrived, to show that several very natural and important decision problems are NP-complete.

We first consider a collection of problems with many applications in operations research and industrial engineering.

Definition 24.1 (Knapsack) Given a finite set S, integer weight function $w : S \to \mathcal{N}$, benefit function $b : S \to \mathcal{N}$, weight limit $W \in \mathcal{N}$, and desired benefit $B \in \mathcal{N}$, determine whether there exists a subset $S' \subseteq S$ such that

$$\begin{aligned} \sum_{a \in S'} w(a) &\leq W \\ \sum_{a \in S'} b(a) &\geq B \ . \end{aligned}$$

□

The name is derived from the problem of trying to decide what you really need to take with you on your camping trip. For another example: you are the coach of a crew team, and you wish to select a starting squad of rowers with a combined weight not exceeding W and combined strength at least B.

Definition 24.2 (Subset Sum) Given a finite set S, integer weight function $w : S \to \mathcal{N}$, and target integer B, does there exist a subset $S' \subseteq S$ such that

$$\sum_{a \in S'} w(a) \;=\; B \;?$$

□

Definition 24.3 (Partition) Given a finite set S and integer weight function $w : S \to \mathcal{N}$, does there exist a subset $S' \subseteq S$ such that

$$\sum_{a \in S'} w(a) \;=\; \sum_{a \in S - S'} w(a) \;?$$

□

Trivially, Partition reduces to Subset Sum by taking

$$B \;=\; \frac{1}{2} \sum_{a \in S} w(a) \;.$$

Also, Subset Sum reduces to Partition by introducing two new elements of weight $N - B$ and $N - (\Sigma - B)$, respectively, where

$$\Sigma \;=\; \sum_{a \in S} w(a)$$

and N is a sufficiently large number (actually $N > \Sigma$ will do). The number N is chosen large enough so that both new elements cannot go in the same partition element, because together they outweigh all the other elements. Now we ask whether this new set of elements can be partitioned into two sets of equal weight (which must be N). By leaving out the new elements, this gives a partition of the original set into two sets of weight B and $\Sigma - B$.

Both Subset Sum and Partition reduce to Knapsack. To reduce Partition to Knapsack, take $b = w$ and $W = B = \frac{1}{2}\Sigma$.

We show that these three problems are as hard as Exact Cover by reducing Exact Cover to Subset Sum. Assume that $X = \{0, 1, \ldots, m-1\}$ in the given instance (X, S) of Exact Cover. For $x \in X$, define

$$\#x \;=\; |\{A \in S \mid x \in A\}| \;,$$

the number of elements of S containing x. Let p be a number exceeding all $\#x$, $0 \le x \le m-1$. Encode $A \in S$ as the number

$$w(A) \;=\; \sum_{x \in A} p^x$$

and take

$$B \;=\; \sum_{x=0}^{m-1} p^x \;=\; \frac{p^m - 1}{p - 1} \;.$$

In p-ary notation, $w(A)$ looks like a string of 0's and 1's with a 1 in position x for each $x \in A$ and 0 elsewhere. The number B in p-ary notation looks like a string of 1's of length m. Adding the numbers $w(A)$ simulates the union of the sets A. The number p was chosen big enough so that we do not get into trouble with carries. Asking whether there is a subset sum that gives B is the same as asking for an exact cover of X.

The *bin packing problem* is an important problem that comes up in industrial engineering and computer memory management.

Definition 24.4 (Bin Packing) Given a finite set S, volumes $w : S \to \mathcal{N}$, and bin size $B \in \mathcal{N}$, what is the minimum number of bins needed to contain all the elements of S? Expressed as a decision problem, given the above data and a natural number k, does there exist a packing into k or fewer bins? □

We can easily reduce Partition to Bin Packing by taking B to be half the total weight of all elements of S and $k = 2$.

An extremely important and general problem in operations research is the integer programming problem.

Definition 24.5 (Integer Programming) Given rational numbers a_{ij}, c_j, and b_i, $1 \le i \le m$, $1 \le j \le n$, find integers $x_1, x_2, \ldots, x_n$ that maximize the linear function

$$\sum_{j=1}^{n} c_j x_j$$

subject to the linear constraints

$$\sum_{j=1}^{n} a_{ij} x_j \quad \le \quad b_i \ , \quad 1 \le i \le m \ . \tag{33}$$

The corresponding decision problem is to test whether there exists a point with integer coordinates in a region defined by the intersection of half-spaces: given a_{ij} and b_i, $1 \le i \le m$, $1 \le j \le n$, test whether there exists an integer point $x_1, \ldots, x_n$ in the region (33). □

In *linear programming*, the x_i's are not constrained to be integers, but may be real. The linear programming problem was shown to be solvable in polynomial time in 1980 by Khachian [60] using a method that has become known as the *ellipsoid method*. In 1984, a more efficient polynomial time algorithm was given by Karmarkar [56]; his method has become known as the *interior point method*. Since that time, several refinements have appeared [90, 102]. The older *simplex method*, originally due to Dantzig (see [19]), is used successfully in practice but is known to be exponential in the worse case.

The integer programming problem is *NP*-hard, as the following reduction from Subset Sum shows: the instance of Subset Sum consisting of a set S with

weights $w : S \to \mathcal{N}$ and threshold B has a positive solution iff the Integer Programming instance

$$0 \leq x_a \leq 1\,, \quad a \in S$$

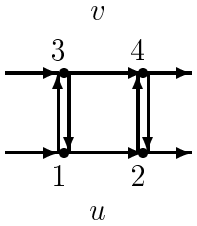
$$\sum_{a \in S} w(a) x_a = B$$

has an integer solution. It is also possible to show that Integer Programming is in *NP* by showing that if there exists an integer solution, then there exists one with only polynomially many bits as a function of the size of the input (n, m, and number of bits in the a_{ij}, b_i, and c_j) [16]. The integer solution can then be guessed and verified in polynomial time.

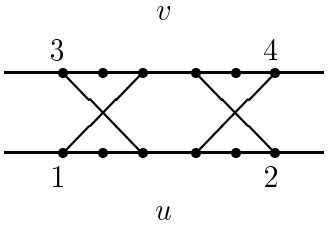
Definition 24.6 (Hamiltonian Circuit) A *Hamiltonian circuit* in a directed or undirected graph $G = (V, E)$ is a circuit that visits each vertex in the graph exactly once. It is like an Euler circuit, except the constraint is on vertices rather than edges. The *Hamiltonian circuit problem* is to determine for a given graph G whether a Hamiltonian circuit exists. □

We reduce Vertex Cover to Hamiltonian Circuit. Recall that 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.

We will build a graph H which will have a Hamiltonian circuit iff G has a vertex cover of size k. The main building block of H for the directed Hamiltonian circuit problem is a widget consisting of four vertices connected as shown.



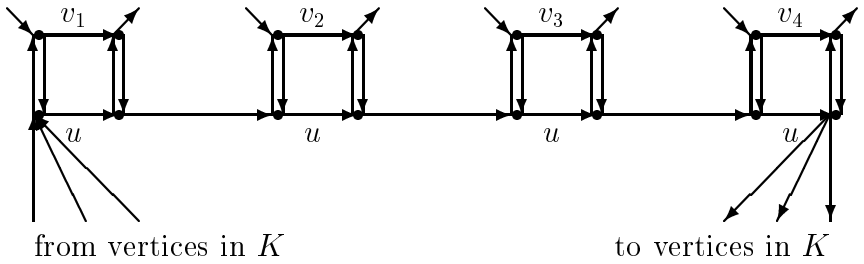
For the undirected version, we use an undirected widget with twelve vertices:



There is one widget corresponding to each edge $(u, v) \in E$. In the widget corresponding to the edge (u, v), one side corresponds to the vertex u and the other to the vertex v.

These widgets have the following interesting property: any Hamiltonian circuit that enters at vertex 1 must leave at vertex 2, and there are only two ways to pass through, either straight through or in a zigzag pattern that crosses to the other side and back. If it goes straight through, then all the vertices on the u side and none of the vertices on the v side are visited. If it crosses to the other side and back, then all the vertices on both sides are visited. Any other path through the widget leaves some vertex stranded, so the path could not be a part of a Hamiltonian circuit. Thus any Hamiltonian circuit that enters at 1 either picks up the vertices in the widget all at once using the zigzag path, or goes straight through and picks up only the vertices on one side, then re-enters at 3 later on to pick up the vertices on the other side.

The graph H is formed as follows. For each vertex u, we string together end-to-end all the u sides of all the widgets corresponding to edges in E incident to u. Call this the *u loop*. In addition, H has a set K of k extra vertices, where k is the parameter of the given instance of Vertex Cover denoting the size of the vertex cover we are looking for. There is an edge from each vertex in K to the first vertex in the u loop, and an edge from the last vertex in the u loop to each vertex in K.



We now show that there is a vertex cover of size k in G iff H has a Hamiltonian circuit. Suppose there is a vertex cover $\{u_1, \ldots, u_k\}$ of G of size k. Then H has a Hamiltonian circuit: starting from the first vertex of K, go through the u_1 loop. When passing through the widget corresponding to an edge (u_1, v) of G, take the straight path if v is in the vertex cover, *i.e.* if $v = u_j$ for some j (the other side of the widget will be picked up later when we traverse the u_j loop), and take the zigzag path if v is not in the vertex cover. When leaving the u_1 loop, go to the second vertex of K, then through the u_2 loop, and so on, all the way around and back to the first vertex of K.

Conversely, if H has a Hamiltonian circuit, the number of u loops traversed must be exactly k, and that set of vertices u forms a vertex cover of G.

This argument holds for both the directed and undirected case. Thus, determining the existence of a Hamiltonian circuit in a directed or undirected

graph is *NP*-hard. It is also in *NP*, since a Hamiltonian circuit can be guessed and verified in polynomial time.

Finally, we consider the *Traveling Salesman Problem (TSP)*. The optimization version of this problem asks for a tour through a set of cities minimizing the total distance. There are several versions of TSP, depending on the properties of the graph and distance function and the type of tour desired. We consider here a quite general formulation.

Definition 24.7 (Traveling Salesman (TSP)) Given a number $k \geq 0$ and a directed graph $G = (V, E)$ with nonnegative edge weights $w : E \to \mathcal{N}$, does there exist a tour of total weight at most k visiting every vertex at least once and returning home? □

Garey and Johnson [39] use a slightly more restricted version which asks for a tour visiting each vertex exactly once. We prefer the more general version above, since to get anywhere from Ithaca and back usually involves at least two stops in Pittsburgh.

TSP is in *NP* provided we can argue that optimal tours are short enough that they can be guessed and verified in polynomial time. Each vertex can be visited at most n times in an optimal tour, because otherwise we could cut out a loop and still visit all vertices. We can thus guess a tour of length at most n^2 and verify that its total weight is at most k.

TSP is *NP*-hard, since there is a straightforward reduction from Hamiltonian Circuit: give all edges unit weight and ask for a TSP tour of weight n.

Combining arguments from the last several lectures, we have:

Theorem 24.8 *The CNF Satisfiability problem reduces via* $\leq_{\mathrm{m}}^{\mathrm{p}}$ *to all the following problems: Knapsack, Partition, Subset Sum, Exact Cover, Bin Packing, Integer Programming, directed and undirected Hamiltonian Circuit, and Traveling Salesman.*