Main topics for lecture include NP-hardness.

Clique and Vertex Cover

- Let's define a couple more problems. A clique is another name for a complete graph. The MaxClique problem asks for the number of vertices in the largest complete subgraph of G.
- A vertex cover is a set of vertices that touch every edge in the graph. MinVertexCover asks for the size of the smallest vertex cover in the graph.
- Below, we have a clique to the left and an vertex cover to the right.

\[\text{MaxClique} \text{ and } \text{MinVertexCover} \text{ are both NP-hard.}\]

- For MaxClique, we define the edge-complement \(\overline{G}\) of G as the graph with the same vertices but the opposite set of edges so \(uv\) is an edge in \(\overline{G}\) if and only if it wasn’t an edge in G.
- A set of vertices is independent in G if and only if it is a clique in \(\overline{G}\). So we can solve MaxIndSet by solving MaxClique in the complement!

\[\text{For MinVertexCover, observe that } I \text{ is an independent set in } G = (V, E) \text{ if and only if } V \setminus I \text{ is a vertex cover. So the largest independent set in } G \text{ is the complement of the smallest vertex cover. If the smallest vertex cover has size } k, \text{ the largest independent set has size } n - k.\]

- Like before, the decisions versions of these problems are also hard. Given G and an
integer k, the problems of deciding if there is a clique of size k and if there is a vertex cover of size k are both NP-complete.

**Graph Coloring**

- Let’s look at yet another graph problem that’s a bit different.
- A *proper k-coloring* of $G = (V, E)$ is a function $C : V \to \{1, 2, \ldots, k\}$ assigning one of k “colors” to each vertex so each edge has distinct colors at its endpoints.
- The graph coloring problem is to find the smallest possible number of colors to get a proper k-coloring.
- It’s directly used for certain applications like compiler design. Can I store all my local variables for this function using only a few registers?
- 3Color is the “easier” problem where we simply ask, given a graph, does it have a 3-coloring?
- Claim: 3Color is NP-complete.
- It’s in NP, because you can just tell me the colors and I can verify its a proper coloring in polynomial time.
- For hardness, we’ll do a reduction from 3SAT. Really, this is just an example of a greater truth: If all else fails, try 3SAT.
- Suppose we’re given a 3CNF formula $\Phi$.
- In many reductions, we build an input to the new problem by combining together a collection of useful subgraphs called *gadgets*. There are three types of gadget for 3Color:
  - A *truth gadget*: A triangle with vertices T, F, and X standing for True, False, and Other. These three vertices have to have different colors in a proper 3-coloring. For convenience, we’ll refer to the colors they get as True, False, and Other respectively.
  - For each variable $a$, a *variable gadget*: a triangle joining two vertices $a$ and $\neg a$ to the same vertex X used in the truth gadget. Vertex $a$ must be colored True or False in proper 3-coloring, implying $\neg a$ gets False or True, respectively.
  - For each clause in $\Phi$, a *clause gadget*: We join the three literal vertices for the clause to vertex T using five new vertex and ten new edges. Tedium case analysis implies that if a 3-coloring gives all three literals the color False, then there is a monochromatic edge in the clause gadget. However, it is always possible to color at least one literal True and the rest False and then extend those colors to a proper 3-coloring of the whole clause gadget.
Here's a whole example. Clearly, we can build it in polynomial time.

I claim the graph is 3-colorable if and only if $\Phi$ is satisfiable.

- If the graph is 3-colorable, exactly one of $a$ or $\sim a$ is assigned True for each variable. And as I already argued, at least one literal per clause is assigned True. So we can give the literals with color True an assignment of True to satisfy $\Phi$.
- If $\Phi$ is satisfiable, then we color each True literal with the color True and each False literal with the color False. We extend these choices to a proper 3-coloring of each clause gadget to get a proper 3-coloring of the whole graph.

Here's the whole algorithm:

- $3$Color is NP-hard since we got a reduction, so it's NP-complete.
- And as you might expect, for any constant $k > 3$, the analogously defined problem $k$Color
is NP-complete. We can even reduce from 3Color.

- And the more general optimization version of graph coloring “how many colors do I need” must be NP-hard as well, since it naturally solves 3Color.

**Hamiltonian Cycle**

- A Hamiltonian cycle is a cycle in a graph that visits every vertex exactly once. (Visiting every edge once is an Eulerian tour.)
- The Hamiltonian cycle problem is given a directed graph \( H = (V, E) \), does \( H \) contain a Hamiltonian cycle?
- Claim: Hamiltonian cycle is NP-complete.
- (Incidentally, the Eulerian tour problem is in P.)
- I can tell you the order of vertices in the cycle so it must be in NP.
- We’ll reduce from the decision version of VertexCover: is there a vertex cover of size \( k \)?
- I’ll be a bit lighter on the details here so we can get to one last example.
- Let’s start with the box algorithm this time. We want to know if undirected graph \( G \) has a vertex cover of size \( k \), so we’ll build a directed graph \( H \) and ask if it has a Hamiltonian cycle.

![Diagram](image.png)

- Now let’s build some gadgets.
- Each edge \( uv \) in \( G \) becomes four vertices \((u, v, in), (u, v, out), (v, u, in), \) and \((v, u, out)\) in \( H \) plus six directed edges
  
  \[
  (u, v, in)\rightarrow(u, v, out) \quad (u, v, in)\rightarrow(v, u, in) \quad (v, u, in)\rightarrow(u, v, in) \\
  (v, u, in)\rightarrow(v, u, out) \quad (u, v, out)\rightarrow(v, u, out) \quad (v, u, out)\rightarrow(u, v, out)
  \]

- Eventually, we’ll have one edge going out from each out vertex and one edge going in to each in vertex as shown below. There’s exactly three routes a Hamiltonian cycle can take through the four vertices. Each will correspond to a choice of \( u \) and \( v \), only \( u \), or only \( v \) being used in a vertex cover of \( G \).
For each vertex $u$ in $G$, we connect all the edge gadgets for edges $uv$ in a directed path called the *vertex chain*. Specifically, if $u$ has $d$ neighbors $v_1, v_2, \ldots, v_d$, we add $d - 1$ edges $(u, v_i, \text{out}) \rightarrow (u, v_{i+1}, \text{in})$ for each $i$ from 1 to $d - 1$.

And we add $k$ additional *cover vertices* $x_1, x_2, \ldots, x_k$. Each has an edge to the first in vertex in each chain and an edge from the last out vertex in each chain. Here’s an example:

- Now, suppose there is a vertex cover $u_1, u_2, \ldots, u_k$ in $G$. We can find a Hamiltonian cycle in $H$. For each $i$ from 1 to $k$, we go from $x_i$, through $u_i$’s chain, and to $x_{i+1}$. We go through each of $u_i$’s edge gadgets as described above, detouring through the $(v, u_i)$ vertices for any edge $uv$ where $v$ is not in the cover.

Conversely, suppose there is a Hamiltonian cycle. It must contain an edge from every cover vertex to the start of some vertex chain. If you start at some edge $(u, v, \text{in})$ you must leave $(u, v, \text{out})$, so we’ll touch every edge gadget in the chain. Every edge gadget for $uv$ must be entered in at least one of its two entrances $(u, v, \text{in})$ or $(v, u, \text{in})$, so we’ll go through enough vertex chains to cover all edges in $G$. And we go through exactly $k$ vertex chains, so we have a vertex cover of size $k$.

- There are many variants of HamiltonianCycle. HamiltonianPath asks if there is a path containing each vertex exactly once. It’s also NP-hard. These problems remain NP-hard in undirected graphs as well.

And if you’re interested, Erickson and CLRS also show reductions from 3Sat directly to Hamiltonian cycle.
Subset Sum

- Let's finish with one last problem that doesn't involve booleans or graphs.
- SubsetSum: Given a set $X$ of positive integers and an integer $T$, does $X$ have a subset whose elements sum to $T$?
- We'll again reduce from VertexCover. Suppose we have a graph $G = (V, E)$ and an integer $k$. Is there a vertex cover of size $k$?
- We need to make a set $X$ of integers and a target value $T$ so there's a subset sum of $T$ if and only if there's a vertex cover of size $k$.
- We're still going to use gadgets, but now the gadgets will be very large numbers.
- Number the edges of $G$ from 0 to $|E| - 1$. $X$ contains $b_i := 4^i$ for each edge $i$.
- For each vertex $v$, it also contains

$$a_v := 4^E + \sum_{i \in \Delta(v)} 4^i$$

where $\Delta(v)$ is the set of edges incident to $v$.
- We can also imagine each integer as written in base 4 using $(E + 1)$-digits. The $|E|$th digit (quit?) is 1 if the integer represents a vertex and is 0 otherwise.
- For each $i < |E|$, the $i$th digit is 1 if the integer represents edge $i$ or one of its endpoints, and is 0 otherwise.
- Finally, set

$$T := k \cdot 4^E + \sum_{i=0}^{E-1} 2 \cdot 4^i.$$

- It only takes polynomial time to write out these numbers in base 4 or even binary.
- Here's the SubsetSum instance for that graph we were using for HamiltonianCycle.

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

\[
\begin{align*}
a_u := 111000_4 &= 1344 & b_{uv} := 010000_4 &= 256 \\
ad := 110110_4 &= 1300 & b_{uw} := 001000_4 &= 64 \\
a_w := 101101_4 &= 1105 & b_{vw} := 000100_4 &= 16 \\
ax := 100011_4 &= 1029 & b_{vx} := 000010_4 &= 4 \\
avx := 100011_4 &= 1029 & b_{wx} := 000001_4 &= 1
\end{align*}
\]

\[T := 222222_4 = 2730\]

- And here's a good example of why we need to do the full proof correctness, because it's still not clear how a subset summing to $T$ corresponds to a vertex cover!
- Suppose there is a vertex cover $C$ of size $k$. 
Let $X_C$ be the subset of integers that contains $a_v$ for every vertex $v$ in $C$ and $b_i$ for every edge $i$ covered exactly once.

The sum of the integers, written in base 4, has a 2 in each of the first $|E|$ digits.

And we’re summing $k$ $4^{|E|}$’s for our $k$ vertices.

So the sum is exactly $T$.

Suppose there is a subset $X'$ of $X$ that sums to $T$.

Let $V'$ be the subset of vertices whose vertex numbers we chose.

There are no carries in the first $E$ digits, because for each $i$ there are only three numbers whose $i$th digit is 1. Each edge number $b_i$ contributes a single 1 to the $i$th digit, so we need one of the vertex numbers for its endpoints. In other words, each edge is covered by $V'$.

And every vertex number is at least $4^{|E|}$, so we can only afford to use $k$ of them. $|V'| \leq k$.

Now, if you were reading carefully, you might have noticed there’s an $O(nT)$ time algorithm for SubsetSum earlier in Erickson. Doesn’t that imply $P = NP$?

That’s another example of a pseudo-polynomial time algorithm, like Ford-Fulkerson with arbitrary integer capacities. The worst-case running time is exponential in the actual input size, because we only used $O(\log T)$ bits to write out these numbers.

NP-hard problems with such algorithms are called weakly NP-hard. If the problem is still NP-hard even when writing down any numbers in unary (so in space proportional to their value), then the problem is called strongly NP-hard. SubsetSum is the only example we’ve seen of a weakly NP-hard problem. Another famous one is the Partition problem, where you try to partition a set of integers into two pieces with equal sums.

And that’s it! I’ll send a reminder, but I think instead of meeting Tuesday, I’ll give you a chance to take a practice QE exam. We can discuss it on Thursday. And I’ll do another regular review session sometime before your final on Thursday May 13th.

And in case I forget to ask later, please fill out evaluations for the course at eval.utdallas.edu. I do take your comments seriously when thinking about how I can improve this and other courses. Comments and scores you give affect things like raises and tenure, so whether or not you liked the course or my instruction, I think it’s in yours and others’ interest to give honest evaluations.