A Hamiltonian Cycle in a graph is a cycle that visits every vertex exactly once (note that this is very different from an Eulerian cycle which visits every edge exactly once).

The Hamiltonian Cycle problem is to determine if a given graph $G$ has a Hamiltonian Cycle.

We will show that this problem is NP-Hard by a reduction from the vertex cover problem.

To do the reduction, we need to show that we can solve Vertex Cover in polynomial time if we have a polynomial time solution to Hamiltonian Cycle.

Given a graph $G$ and an integer $k$, we will create another graph $G'$ such that $G'$ has a Hamiltonian cycle iff $G$ has a vertex cover of size $k$.

As for the last reduction, our transformation will consist of putting together several "gadgets."
**Edge Gadget and Cover Vertices**

- For each edge \((u, v)\) in \(G\), we have an *edge gadget* in \(G'\) consisting of twelve vertices and fourteen edges, as shown below.

\[(u,v,1)\quad (u,v,2)\quad (u,v,3)\quad (u,v,4)\quad (u,v,5)\quad (u,v,6)\quad...

### An edge gadget for \((u, v)\) and the only possible Hamiltonian paths through it.

**Cover Vertices**

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

**Edge Gadget**

- 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 the three ways shown in the figure.
- These paths through the edge gadget will correspond to one or both of the vertices \(u\) and \(v\) being in the vertex cover.

**Vertex Chains**

- 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 the following edges:
  - \(d - 1\) edges between \((u, v_i, 6)\) and \((u, v_{i+1}, 1)\) (for all \(i\) between 1 and \(d - 1\))
  - \(k\) edges between the cover vertices and \((u, v_1, 1)\)
  - \(k\) edges between the cover vertices and \((u, v_d, 6)\)
### The Reduction

- 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
- To get this Hamiltonian cycle, we start at cover vertex 1, traverse through the vertex chain for \( v_1 \), then visit cover vertex 2, then traverse the vertex chain for \( v_2 \) and so forth, until we eventually return to cover vertex 1
- Conversely, one can prove that any Hamiltonian cycle in \( G' \) alternates between cover vertices and vertex chains, and that the vertex chains correspond to the \( k \) vertices in a vertex cover of \( G \)

Thus, \( G \) has a vertex cover of size \( k \) iff \( G' \) has a Hamiltonian cycle

### Example

The original graph \( G \) with vertex cover \( \{v, w\} \), and the transformed graph \( G' \) with a corresponding Hamiltonian cycle (bold edges).

Vertex chains are colored to match their corresponding vertices.
• A problem closely related to Hamiltonian cycle is the famous Traveling Salesperson Problem (TSP).
• The TSP problem is: “Given a 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 since Hamiltonian Cycle is NP-hard, TSP is also NP-hard!

• In 1999, Richard Kaye proved that the solitaire game Minesweeper is NP-Hard, using a reduction from Circuit Satifiability.
• Also in the last few years, Eric Demaine, et. al., proved that the game Tetris is NP-Hard.

---

Consider the optimization version of, say, the graph coloring problem: “Given a graph $G$, what is the smallest number of colors needed to color the graph?” (Note that unlike the decision version of this problem, this is not a yes/no question).

• Show that the optimization version of graph coloring is also NP-Hard by a reduction from the decision version of graph coloring.
• Is the optimization version of graph coloring also NP-Complete?

Consider the problem 4Sat which is: “Is there any assignment of variables to a 4CNF formula that makes the formula evaluate to true?”

• Is this problem NP-Hard? If so, give a reduction from 3Sat that shows this. If not, give a polynomial time algorithm which solves it.
Challenge Problem

• Consider the following problem: “Does there exist a clique of size 5 in some input graph $G$?”
• Is this problem NP-Hard? If so, prove it by giving a reduction from some known NP-Hard problem. If not, give a polynomial time algorithm which solves it.

Vertex Cover

• A vertex cover of a graph is a set of vertices that touches every edge in the graph
• The decision version of Vertex Cover is: “Does there exist a vertex cover of size $k$ in a graph $G$?”
• We’ve proven this problem is NP-Hard by an easy reduction from Independent Set
• The optimization version of Vertex Cover is: “What is the minimum size vertex cover of a graph $G$?”
• We can prove this problem is NP-Hard by a reduction from the decision version of Vertex Cover (left as an exercise).

Approximating Vertex Cover

• Even though the optimization version of Vertex Cover is NP-Hard, it’s possible to approximate the answer efficiently
• In particular, in polynomial time, we can find a vertex cover which is no more than 2 times as large as the minimal vertex cover

Approximation Algorithm

• The approximation algorithm does the following until $G$ has no more edges:
• It chooses an arbitrary edge $(u, v)$ in $G$ and includes both $u$ and $v$ in the cover
• It then removes from $G$ all edges which are incident to either $u$ or $v$
Approximation Algorithm

Approx-Vertex-Cover(G){
    C = {};
    E' = Edges of G;
    while(E' is not empty){
        let (u,v) be an arbitrary edge in E';
        add both u and v to C;
        remove from E' every edge incident to u or v;
    }
    return C;
}

Analysis

• If we implement the graph with adjacency lists, each edge need be touched at most once
• Hence the run time of the algorithm will be $O(|V| + |E|)$, which is polynomial time
• First, note that this algorithm does in fact return a vertex cover since it ensures that every edge in $G$ is incident to some vertex in $C$
• Q: Is the vertex cover actually no more than twice the optimal size?

TSP

• An optimization version of the TSP problem is: “Given a weighted graph $G$, what is the shortest Hamiltonian Cycle of $G$?”
• This problem is NP-Hard by a reduction from Hamiltonian Cycle
• However, there is a 2-approximation algorithm for this problem if the edge weights obey the triangle inequality

Analysis

• Let $A$ be the set of edges which are chosen in the first line of the while loop
• Note that no two edges of $A$ share an endpoint
• Thus, any vertex cover must contain at least one endpoint of each edge in $A$
• Thus if $C^*$ is an optimal cover then we can say that $|C^*| \geq |A|$
• Further, we know that $|C| = 2|A|$
• This implies that $|C| \leq 2|C^*|$

Which means that the vertex cover found by the algorithm is no more than twice the size of an optimal vertex cover.
Triangle Inequality

- In many practical problems, it’s reasonable to make the assumption that the weights, $c$, of the edges obey the triangle inequality.
- The triangle inequality says that for all vertices $u, v, w \in V$:
  \[ c(u, w) \leq c(u, v) + c(v, w) \]
- In other words, the cheapest way to get from $u$ to $w$ is always to just take the edge $(u, w)$.
- In the real world, this is usually a pretty natural assumption. For example it holds if the vertices are points in a plane and the cost of traveling between two vertices is just the euclidean distance between them.

Approximation Algorithm

- Given a weighted graph $G$, the algorithm first computes a MST for $G$, $T$, and then arbitrarily selects a root node $r$ of $T$.
- It then lets $L$ be the list of the vertices visited in a depth first traversal of $T$ starting at $r$.
- Finally, it returns the Hamiltonian Cycle, $H$, that visits the vertices in the order $L$.

Approx-TSP(G) {
    T = MST(G);
    L = the list of vertices visited in a depth first traversal of T, starting at some arbitrary node in T;
    H = the Hamiltonian Cycle that visits the vertices in the order L;
    return H;
}

Example Run

The top left figure shows the graph $G$ (edge weights are just the Euclidean distances between vertices); the top right figure shows the MST $T$. The bottom left figure shows the depth first walk on $T$, $W = (a, b, c, h, b, a, d, e, f, e, g, e, c, d, a)$; the bottom right figure shows the Hamiltonian cycle $H$ obtained by deleting repeat visits from $W$, $H = (a, b, c, h, d, e, f, g)$. 
Analysis

- The first step of the algorithm takes $O(|E| + |V| \log |V|)$ (if we use Prim's algorithm).
- The second step is $O(|V|)$.
- The third step is $O(|V|)$.
- Hence the run time of the entire algorithm is polynomial.

Analysis

An important fact about this algorithm is that: the cost of the MST is less than the cost of the shortest Hamiltonian cycle.

- To see this, let $T$ be the MST and let $H^*$ be the shortest Hamiltonian cycle.
- Note that if we remove one edge from $H^*$, we have a spanning tree, $T'$.
- Finally, note that $w(H^*) \geq w(T') \geq w(T)$.
- Hence $w(H^*) \geq w(T)$.

Unfortunately, $W$ is not a Hamiltonian cycle since it visits some vertices more than once.
- However, we can delete a visit to any vertex and the cost will not increase because of the triangle inequality. (The path without an intermediate vertex can only be shorter.)
- By repeatedly applying this operation, we can remove from $W$ all but the first visit to each vertex, without increasing the cost of $W$.
- In our example, this will give us the ordering $H = (a, b, c, h, d, e, f, g)$.
Analysis

- By the last slide, $c(H) \leq c(W)$.
- So $c(H) \leq c(W) = 2c(T) \leq 2c(H^*)$
- Thus, $c(H) \leq 2c(H^*)$
- In other words, the Hamiltonian cycle found by the algorithm has cost no more than twice the shortest Hamiltonian cycle.

Take Away

- Many real-world problems can be shown to not have an efficient solution unless $P = NP$ (these are the NP-Hard problems)
- However, if a problem is shown to be NP-Hard, all hope is not lost!
- In many cases, we can come up with a provably good approximation algorithm for the NP-Hard problem.