Hamiltonian Circuit

Instance: A graph G

Question: Does the graph contains a HC, i.e. an ordered of the vertices $\{v_1, v_2, ..., v_n\}$?

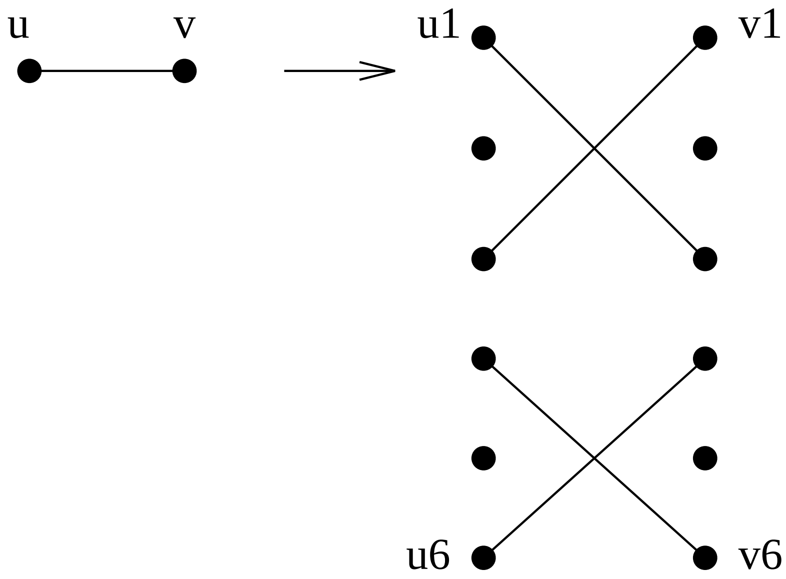
This problem is intimately relates to the Travelling Salesman.

Question: Is there an ordering of the vertices of a weighted graph such that $w(v_1, v_n) + \sum w(v_i, v_{i+1}) \leq k$?

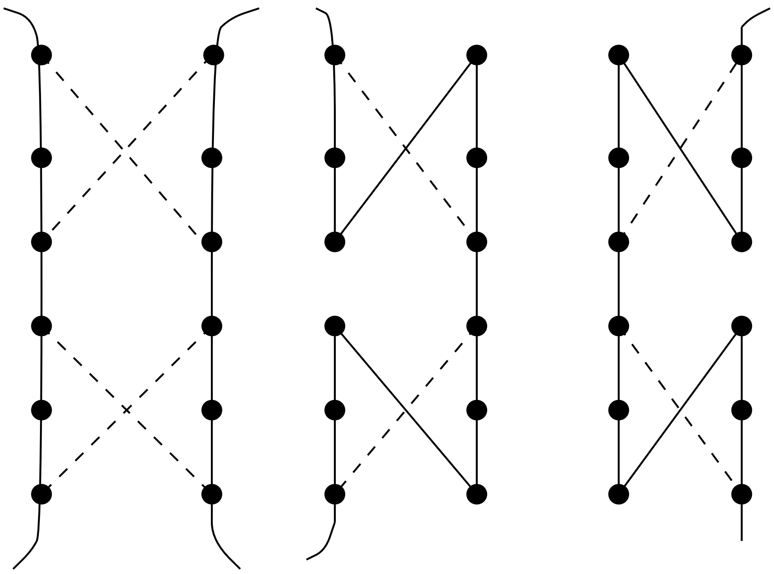
Clearly, $HC \propto TSP$. Assign each edge in G weight 1, any edge not in G weight 2. This new graph has a Travelling Salesman tour of cost n iff the graph is Hamiltonian. Thus TSP is NP-complete if we can show HC is NP-complete.

Theorem: Hamiltonian Circuit is NP-complete

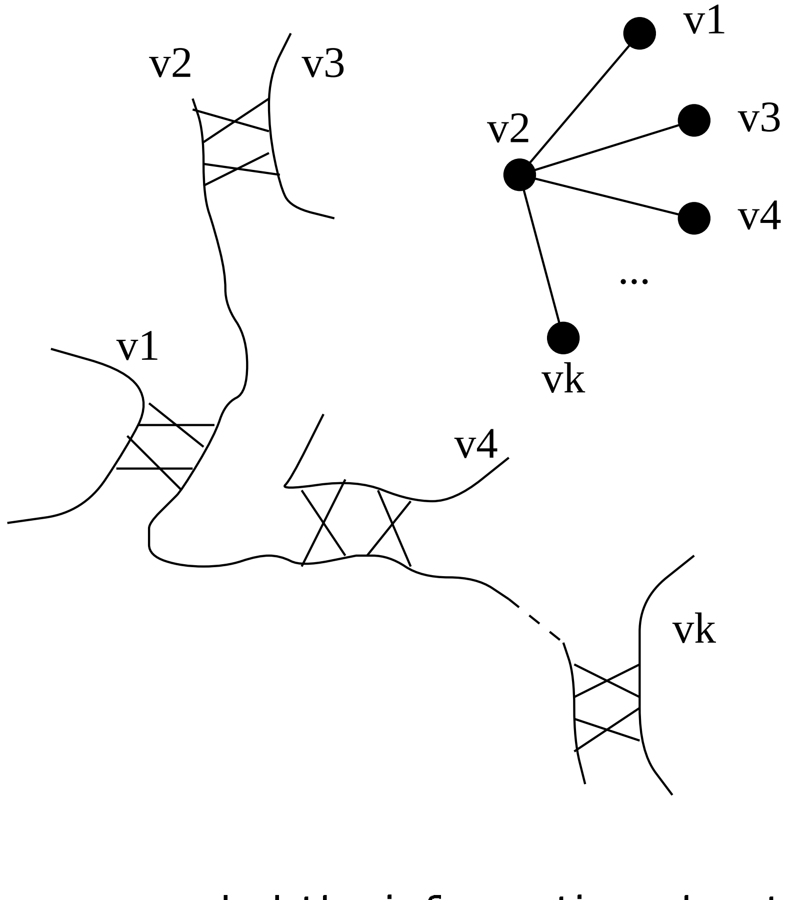
Proof: Clearly HC is in NP-guess a permutation and check it out. To show it is complete, we use vertex cover. A vertex cover instance consists of a graph and a constant k, the minimum size of an acceptable cover. We must construct another graph. Each edge in the initial graph will be represented by the following component:



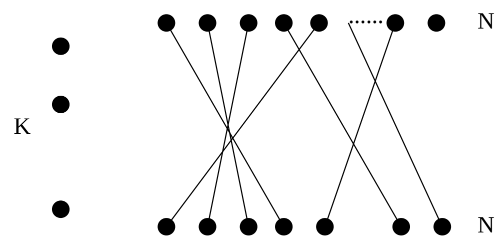
All further connections to this gadget will be through vertices v_1, v_6, u_1 and u_6. The key observation about this gadget is that there are only three ways to traverse all the vertices:



Note that in each case, we exit out the same side we entered. Each side of each edge gadget is associated with a vertex. Assuming some arbitrary order to the edges incident on a particular vertex, we can link successive gadgets by edges forming a chain of gadgets. Doing this for all vertices in the original graph creates n intertwined chains with n entry points and n exits.



Thus we have encoded the information about the initial graph. What about k? We set up k additional vertices and connect each of these to the n start points and n end points of each chain.



Total size of new graph: $GE + K$ vertices and $12E + 2kN + 2E$ edges $\rightarrow$ construction is polynomial in size and time.

We claim this graph has a HC iff G has a VC of size k.

1. Suppose $\{v_1, v_2, ..., v_n\}$ is a HC.

   Assume it starts at one of the k selector vertices. It must then go through one of the chains of gadgets until it reaches a different selector vertex.

   Since the tour is a HC, all gadgets are traversed. The k chains correspond to the vertices in the cover.

   Note that if both vertices associated with an edge are in the cover, the gadget will be traversal in two pieces - otherwise one chain suffices.

   To avoid visiting a vertex more than once, each chain is associated with a selector vertex.

2. Now suppose we have a vertex cover of size $\leq k$.

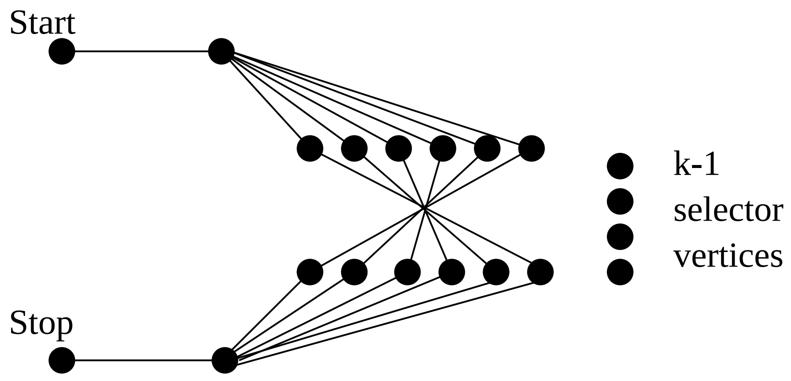
   We can always add more vertices to the cover to bring it up to size k.

   For each vertex in the cover, start traversing the chain. At each entry point to a gadget, check if the other vertex is in the cover and traverse the gadget accordingly.

   Select the selector edges to complete the circuit.

Neat, sweet, and NP-complete.

To show that Longest Path or Hamiltonian Path is *NP*-complete, add start and stop vertices and distinguish the first and last selector vertices.



This has a Hamiltonian path from start to stop iff the original graph has a vertex cover of size k.

Other NP-complete Problems

- Partition - can you partition n integers into two subsets so that the sums of the subset are equal?
- Bin Packing - how many bins of a given size do you need to hold n items of variable size?
- Chromatic Number - how many colors do you need to color a graph?
- $N \times N$ checkers - does black have a forced win from a given position?
- Scheduling, Code Optimization, Permanent Evaluation, Quadratic Programming, etc.

Open: Graph Isomorphism, Composite Number, Minimum Length Triangulation.

Polynomial or Exponential?

Just changing a problem a little can make the difference between it being in P or NP-complete:

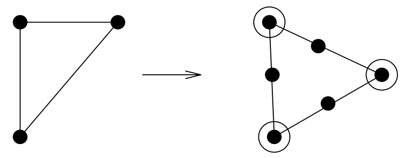
	P	NP-complete
	Shortest Path	Longest Path
	Eulerian Circuit	Hamiltonian Circuit
	Edge Cover	Vertex Cover

The first thing you should do when you suspect a problem might be NP-complete is look in Garey and Johnson, *Computers and Intractability*. It contains a list of several hundred problems known to be NP-complete. Either what you are looking for will be there or you might find a closely related problem to use in a reduction.

Techniques for Proving NP-completeness

1. *Restriction* - Show that a special case of the problem you are interested in is NP-complete. For example, the problem of finding a path of length k is really Hamiltonian Path.

2. *Local Replacement* - Make local changes to the structure. An example is the reduction $SAT \propto 3-SAT$. Another example is showing isomorphism is no easier for bipartite graphs:

   For any graph, replacing an edge with makes it bipartite.

3. *Component Design* - These are the ugly, elaborate constructions

The Art of Proving Hardness

Proving that problems are hard is an skill. Once you get the hang of it, it is surprisingly straightforward and pleasurable to do. Indeed, the dirty little secret of NP-completeness proofs is that they are usually easier to recreate than explain, in the same way that it is usually easier to rewrite old code than the try to understand it.

I offer the following advice to those needing to prove the hardness of a given problem:

- *Make your source problem as simple (i.e. restricted) as possible.*

  Never use the general traveling salesman problem (TSP) as a target problem. Instead, use TSP on instances restricted to the triangle inequality. Better, use Hamiltonian cycle, i.e. where all the weights are 1 or ∞. Even better, use Hamiltonian path instead of cycle. Best of all, use Hamiltonian path on directed, planar graphs where each vertex has total degree 3. All of these problems are equally hard, and the more you can restrict the problem you are reducing, the less work your reduction has to do.

- *Make your target problem as hard as possible.*

  Don't be afraid to add extra constraints or freedoms in order to make your problem more general (at least temporarily).

- *Select the right source problem for the right reason.*

  Selecting the right source problem makes a big difference is how difficult it is to prove a problem hard. This is the first and easiest place to go wrong.

  I usually consider four and only four problems as candidates for my hard source problem. Limiting them to four means that I know a lot about these problems – which variants of these problems are hard and which are soft. My favorites are:

  – 3-Sat – that old reliable... When none of the three problems below seem appropriate, I go back to the source.

  – Integer partition – the one and only choice for problems whose hardness seems to require using large numbers.

  – Vertex cover – for any graph problems whose hardness depends upon *selection*. Chromatic number, clique, and independent set all involve trying to select the correct subset of vertices or edges.

  – Hamiltonian path – for any graph problems whose hardness depends upon *ordering*. If you are trying to route or schedule something, this is likely your lever.

- *Amplify the penalties for making the undesired transition.*

  You are trying to translate one problem into another, while making them stay the same as much as possible. The easiest way to do this is to be bold with your penalties, to punish anyone trying to deviate from your proposed solution. "If you pick this, then you have to pick up this huge set which dooms you to lose." The sharper the consequences for doing what is undesired, the easier it is to prove if and only if.

- *Think strategically at a high level, then build gadgets to enforce tactics.*

  You should be asking these kinds of questions. "How can I force that either A or B but not both are chosen?" "How can I force that A is taken before B?" "How can I clean up the things I did not select?"

- *Alternate between looking for an algorithm or a reduction if you get stuck.*

  Sometimes the reason you cannot prove hardness is that there is an efficient algorithm to solve your problem! When you can't prove hardness, it likely pays to change your thinking at least for a little while to keep you honest.

Now watch me try it!

To demonstrate how one goes about proving a problem hard, I accept the challenge of showing how a proof can be built on the fly.

I need a volunteer to pick a random problem from the 400+ hard problems in the back of Garey and Johnson.

Dealing with NP-complete Problems

Option 1: Algorithm fast in the Average case

Examples are Branch-and-bound for the Traveling Salesman Problem, backtracking algorithms, etc.

Option 2: Heuristics

Heuristics are rules of thumb; fast methods to find a solution with no requirement that it be the best one.

Note that the theory of NP-completeness does not stipulate that it is hard to get close to the answer, only that it is hard to get the optimal answer.

Often, we can prove performance bounds on heuristics, that the resulting answer is within C times that of the optimal one.