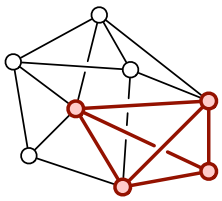
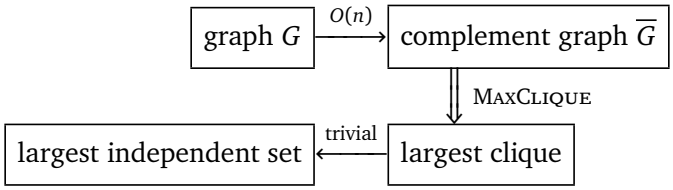


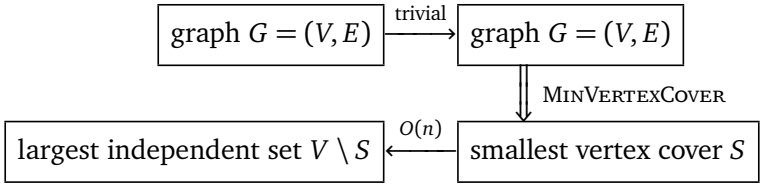
A graph with maximum clique size 4.



30.9 Vertex Cover (from Independent Set)

A *vertex cover* of a graph is a set of vertices that touches every edge in the graph. The MinVertexCover 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.



30.10 Graph Coloring (from 3SAT)

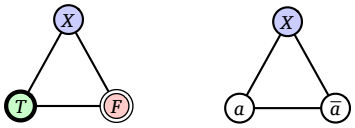
A *k-coloring* of a graph is a map $C : V \to \{1, 2, \ldots, k\}$ that assigns one of k 'colors' to each vertex, so that every edge has two different colors at its endpoints. The graph coloring problem is to find the smallest possible number of colors in a legal coloring. To show that this problem is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

To prove that 3Colorable is NP-hard, we use a reduction from 3SAT. Given a 3CNF formula Φ, we produce a graph G_Φ 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.

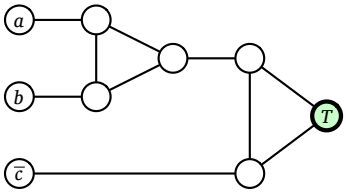
- The variable gadget for a variable a is also a triangle joining two new nodes labeled a and $\overline{a}$ to node X in the truth gadget. Node a must be colored either True or False, and so node $\overline{a}$ must be colored either False or True, respectively.

- 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. A straightforward case analysis



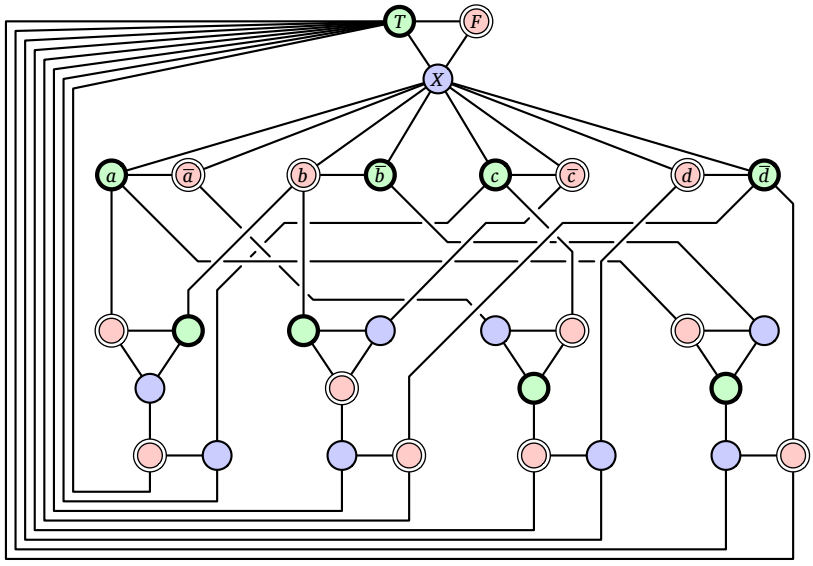
The truth gadget and a variable gadget for a.

implies that if all three literal nodes in the clause gadget are colored FALSE, then some edge in the gadget must be monochromatic. 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. On the other hand, for any coloring of the literal nodes where at least one literal node is colored TRUE, there is a valid 3-coloring of the clause gadget.



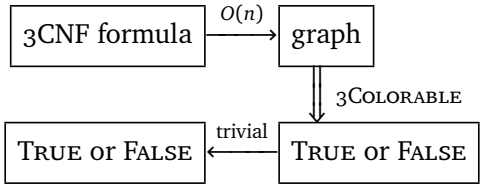
A clause gadget for $(a \vee b \vee \bar{c})$.

The final graph G_Φ contains exactly *one* node T, exactly *one* node F, and exactly *two* nodes a and $\bar{a}$ for each variable. For example, the formula $(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$ that I used to illustrate the MAXCLIQUE reduction would be transformed into the graph shown on the next page. The 3-coloring is one of several that correspond to the satisfying assignment $a = c = \text{TRUE}$, $b = d = \text{FALSE}$.



A 3-colorable graph derived from the satisfiable 3CNF formula $(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$

Now the proof of correctness is just brute force case analysis. 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 decision problem.

30.11 Hamiltonian Cycle (from Vertex Cover)

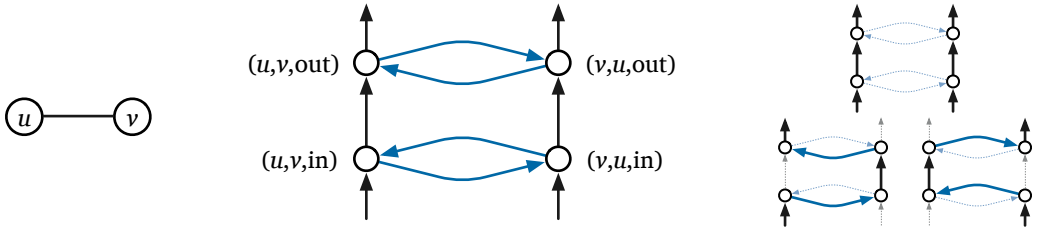
A ***Hamiltonian cycle*** in a graph is a cycle that visits every vertex exactly once. This is very different from an *Eulerian cycle*, which is actually a closed *walk* that traverses every *edge* exactly once. Eulerian cycles are easy to find and construct in linear time using a variant of depth-first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

To prove that finding a Hamiltonian cycle in a directed graph is NP-hard, we describe a reduction from the vertex cover problem. Given an undirected graph G and an integer k, we need to transform it into another graph H, such that H has a Hamiltonian cycle if and only if G has a vertex cover of size k. As usual, our transformation uses several gadgets.

- For each undirected edge uv in G, the directed graph H contains an *edge gadget* consisting of four vertices $(u, v, \text{in}), (u, v, \text{out}), (v, u, \text{in}), (v, u, \text{out})$ and six directed edges

$$(u,v,\text{in})\to(u,v,\text{out}) \qquad (u,v,\text{in})\to(v,u,\text{in}) \qquad (v,u,\text{in})\to(u,v,\text{in})$$
$$(v,u,\text{in})\to(v,u,\text{out}) \qquad (u,v,\text{out})\to(v,u,\text{out}) \qquad (v,u,\text{out})\to(u,v,\text{out})$$

  as shown on the next page. Each "in" vertex has an additional incoming edge, and each "out" vertex has an additional outgoing edge. A Hamiltonian cycle must pass through an edge gadget in one of three ways—either straight through on both sides, or with a detour from one side to the other and back. Eventually, these options will correspond to both u and v, only u, or only v belonging to some vertex cover.

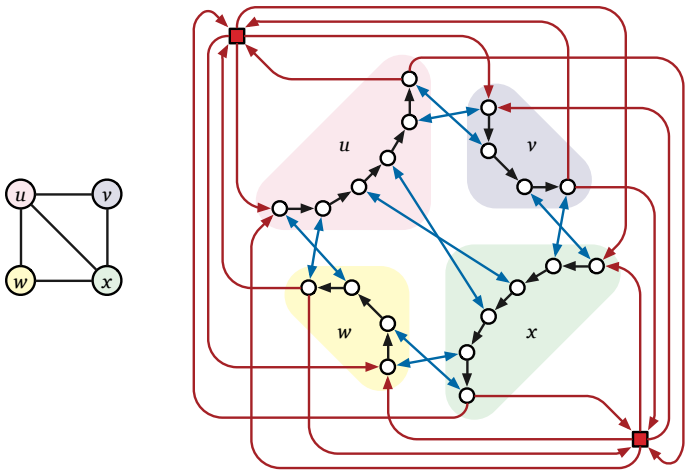


An edge gadget for uv and its only possible intersections with a Hamiltonian cycle.

- For each vertex u in G, all the edge gadgets for incident edges uv are connected in H into a single directed path, which we call a *vertex chain*. Specifically, suppose vertex u has d neighbors $v_1, v_2, \ldots, v_d$. Then H has $d-1$ additional edges $(u, v_i, \text{out})\to(u, v_{i+1}, \text{in})$ for each i.

- Finally, H also contains k *cover vertices*, simply numbered 1 through k. Each cover vertex has a directed edge to the first vertex in each vertex chain, and a directed edge from the last vertex in each vertex chain.

An example of our complete transformation is shown below.

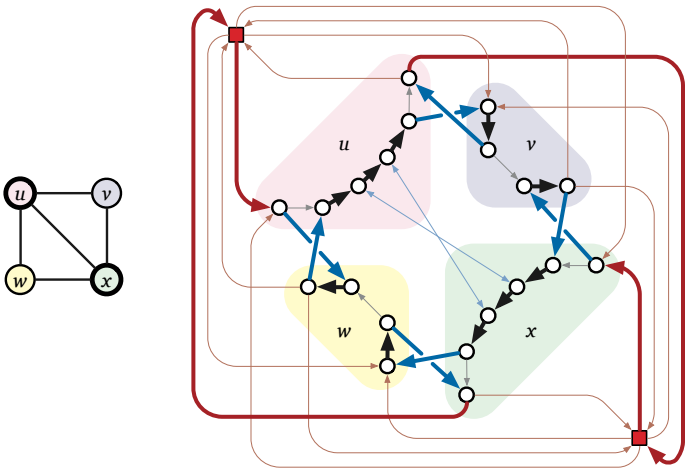


The original graph G and the transformed graph H, where $k = 2$.

Now suppose $C = \{u_1, u_2, \ldots, u_k\}$ is a vertex cover of G. Then H contains a Hamiltonian cycle, constructed as follows. Start at cover vertex 1, through traverse the vertex chain for vu_1, then visit cover vertex 2, then traverse the vertex chain for vu_2, and so forth, eventually returning to cover vertex 1. As we traverse the vertex chain for any vertex u_i, we have a choice for how to proceed when we reach any node (u_i, v, in).

- If $v \in C$, follow the edge $(u_i, v, \text{in}) \to (u_i, v, \text{out})$.
- If $v \notin C$, detour through the path $(u_i, v, \text{in}) \to (v, u_i, \text{in}) \to (v, u_i, \text{out}) \to (u_i, v, \text{out})$.

Thus, for each edge uv of G, the Hamiltonian cycle visits (u, v, in) and (u, v, out) as part of u's vertex chain if $u \in C$ and as part of v's vertex chain otherwise.

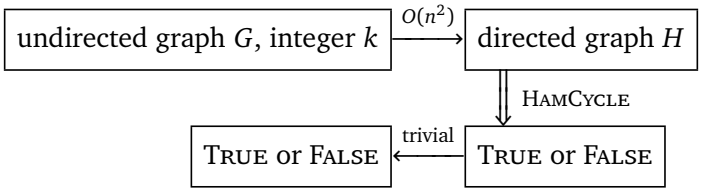


A vertex cover $\{u, x\}$ in G and the corresponding Hamiltonian cycle in H.

Now suppose H contains a Hamiltonian cycle C. This cycle must contain an edge from each cover vertex to the start of some vertex chain. Our case analysis of edge gadgets inductively implies that after C enters the vertex chain for some vertex u, it must traverse the entire vertex chain. Specifically, at each vertex (u, v, in), the cycle must contain either the single edge $(u, v, \text{in}) \to (u, v, \text{out})$ or the detour path $(u, v, \text{in}) \to (v, u, \text{in}) \to (v, u, \text{out}) \to (u, v, \text{out})$, followed by an edge to the next edge gadget in u's vertex chain, or to a cover vertex if this is the last such edge gadget. In particular, if C contains the detour edge $(u, v, \text{in}) \to (v, u, \text{in})$, it does not contain edges between any cover vertex and v's vertex chain. It follows that C traverses exactly k vertex chains. Moreover, these vertex chains describe a vertex cover of the original graph G, because C visits the vertex (u, v, in) for every edge uv in G.

We conclude that G contains a vertex cover of size k if and only if H contains a Hamiltonian cycle.

The transformation from G to H takes at most $O(n^2)$ time; we conclude that 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 is NP-complete.



A closely related problem to Hamiltonian cycles is the famous *traveling salesman problem*—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 the traveling salesman problem is also NP-hard.

Finally, we can prove prove that finding Hamiltonian cycles in *undirected* graphs is NP-hard using a simple reduction from the same problem in *directed* graphs. I'll leave the details of this reduction as an entertaining exercise.

30.12 Subset Sum (from Vertex Cover)

The next problem that we prove NP-hard is the SubsetSum problem considered in the very first lecture on recursion: Given a set X of positive integers and an integer t, determine whether X has a subset whose elements sum to t.

To prove this problem is NP-hard, we once again reduce from VertexCover. Given a graph G and an integer k, we compute a set X of integer and an integer t, such that X has a subset that sums to t if and only if G has an vertex cover of size k. Our transformation uses just two 'gadgets', which are *integers* representing vertices and edges in G.

Number the *edges* of G arbitrarily from 0 to $m-1$. Our set X contains the integer $b_i := 4^i$ for each edge i, and the integer

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

for each vertex v, where $\Delta(v)$ is the set of edges that have v has an endpoint. Alternately, we can think of each integer in X as an $(m+1)$-digit number written in base 4. The mth digit is 1 if the integer represents a vertex, and 0 otherwise; and for each $i < m$, the ith digit is 1 if the

integer represents edge i or one of its endpoints, and 0 otherwise. Finally, we set the target sum

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

Now let's prove that the reduction is correct. First, suppose there is a vertex cover of size k in the original graph G. Consider the subset $X_C \subseteq X$ that includes a_v for every vertex v in the vertex cover, and b_i for every edge i that has *exactly one* vertex in the cover. The sum of these integers, written in base 4, has a 2 in each of the first m digits; in the most significant digit, we are summing exactly k 1's. Thus, the sum of the elements of X_C is exactly t.

On the other hand, suppose there is a subset $X' \subseteq X$ that sums to t. Specifically, we must have

$$\sum_{v \in V'} a_v + \sum_{i \in E'} b_i = t$$

for some subsets $V' \subseteq V$ and $E' \subseteq E$. Again, if we sum these base-4 numbers, there are no carries in the first m digits, because for each i there are only three numbers in X whose ith digit is 1. Each edge number b_i contributes only one 1 to the ith digit of the sum, but the ith digit of t is 2. Thus, for each edge in G, at least one of its endpoints must be in V'. In other words, V is a vertex cover. On the other hand, only vertex numbers are larger than 4^m, and $\lfloor t/4^m \rfloor = k$, so V' has at most k elements. (In fact, it's not hard to see that V' has *exactly* k elements.)

For example, given the four-vertex graph used on the previous page to illustrate the reduction to Hamiltonian cycle, our set X might contain the following base-4 integers:

$$\begin{array}{ll} a_u := 111000_4 = 1344 & b_{uv} := 010000_4 = 256 \\ a_v := 110110_4 = 1300 & b_{uw} := 001000_4 = 64 \\ a_w := 101101_4 = 1105 & b_{vw} := 000100_4 = 16 \\ a_x := 100011_4 = 1029 & b_{vx} := 000010_4 = 4 \\ & b_{wx} := 000001_4 = 1 \end{array}$$

If we are looking for a vertex cover of size 2, our target sum would be $t := 222222_4 = 2730$. Indeed, the vertex cover $\{v, w\}$ corresponds to the subset $\{a_v, a_w, b_{uv}, b_{uw}, b_{vx}, b_{wx}\}$, whose sum is $1300 + 1105 + 256 + 64 + 4 + 1 = 2730$.

The reduction can clearly be performed in polynomial time. Since VERTEXCOVER is NP-hard, it follows that SUBSETSUM is NP-hard.

There is one subtle point that needs to be emphasized here. Way back at the beginning of the semester, we developed a dynamic programming algorithm to solve SUBSETSUM in time $O(nt)$. Isn't this a polynomial-time algorithm? idn't we just prove that P=NP? Hey, where's our million dollars? Alas, life is not so simple. True, the running time is polynomial in n and t, but in order to qualify as a true polynomial-time algorithm, the running time must be a polynomial function *of the size of the input*. The *values* of the elements of X and the target sum t could be exponentially larger than the number of input bits. Indeed, the reduction we just described produces a value of t that is exponentially larger than the size of our original input graph, which would force our dynamic programming algorithm to run in exponential time.

Algorithms like this are said to run in ***pseudo-polynomial time***, and any NP-hard problem with such an algorithm is called ***weakly NP-hard***. Equivalently, a weakly NP-hard problem is one that can be solved in polynomial time when all input numbers are represented in *unary* (as a sum of 1s), but becomes NP-hard when all input numbers are represented in *binary*. If a problem is NP-hard even when all the input numbers are represented in unary, we say that the problem is ***strongly NP-hard***.