1 Introduction

We now want to use our new-found knowledge of:

Theorem 1.1 (Cook, 1971)**.** SAT *is* **NP***-complete*

to prove that a lot of other problems are **NP**-complete.

2 Small clauses are enough

First of all we're going to show that we can reduce clauses in SAT to contain at most three literals, while still retaining **NP**-completeness. To that end, let's define:

k-SATISFIABILITY
Instance A SAT instance with exactly k literals per clause.
Problem Is it satisfiable?

Proposition 2.1. *For $k \geqslant 3$, k-SAT is* **NP***-complete. For $k < 3$ it is in* **P***.*

Proof. Let's start with the easy bits. Certainly 1-SAT is in **P**, since all we need to do is check whether both a literal and its negation occur as clauses (if so, we're unsatisfiable, it not, we're satisfiable).

For 2-SAT, things are a little more interesting. Suppose that we have an instance of 2-SAT using n variables $x_1, x_2, \ldots, x_n$. I'm going to show how to reduce it in polynomial time to an instance in $(n-1)$ variables which is satisfiable if and only if the original one was. Eventually that's all we need since once we get to one variable the situation will be clear!

Consider the variable x_n and all the clauses $z \vee x_n$ in our instance (where z is another literal from among x_1 through x_{n-1} and their negations). Let the set of these literals be P ("positive for x_n"). If, in P we have an occurrence of both a variable and its negation, then in any satisfying assignment, x_n would have to be true (otherwise, one of those two clauses would be false). In that case, we just set $x_n = \mathsf{t}$, include all the individual literals

that occur in clauses $y \vee \neg x_n$ as new one-literal clauses[1] and we have our $(n-1)$-variable instance. We can do a similar thing with the literals N occurring in the clauses $z \vee \neg x_n$.

If neither of these hold, there's no reason to suppose in advance that x_n would have to be either t or f. However, if $x_n = \mathsf{t}$ in a satisfying assignment then we must have

$$\bigwedge_{w \in N} w = \mathsf{t},$$

while, if $x_n = \mathsf{f}$ in a satisfying assignment then we must have

$$\bigwedge_{z \in P} z = \mathsf{t}.$$

So, we must have:

$$\left(\bigwedge_{w \in N} w \right) \bigvee \left(\bigwedge_{z \in P} z \right) = \mathsf{t}$$

for there to be the possibility of a satisfying assignment (and the rest of the clauses must also be satisfied). This is not in CNF but using the distributive law of $\vee$ over $\wedge$ it's equivalent to a conjunction of $|N||P|$ clauses each of size 2 in the remaining variables.

Thus, in any case we're down to an instance of 2-SAT in fewer variables.

Now, what about the case where $k \geqslant 3$? Obviously, it's enough to show that 3-SAT is **NP**-complete since 4-SAT is more general than 3-SAT etc. etc. (we can get to 4-SAT from 3-SAT just by duplicating an element in each clause – even if you don't want to allow duplicated literals in clauses there are easy work-arounds – see footnote 2 below).

Small clauses are no problem since we can just add duplicated literals if need be[2].

[1] If you're worried that our clauses are supposed to have exactly two literals, note that in this case we can actually "resolve" these clauses, i.e., the truth value of those literals y is forced so we can actually eliminate (recursively if need be) more than one variable.

[2] Again, if this offends your sense of what a "3-literal clause" should be, we can be more clever, e.g., replace the clause x by four clauses $x \vee x_1 \vee x_2$, $x \vee \neg x_1 \vee x_2$, $x \vee x_1 \vee \neg x_2$, $x \vee \neg x_1 \vee \neg x_2$ where x_1 and x_2 are new variables. This family of clauses can only be satisfied by setting $x = \mathsf{t}$ so serves the same purpose as the original. Similarly to boost a 2-literal clause to 3 literals takes only one new variable and two new clauses

The key idea here is to show that for every "big" clause

$$x_1 \vee x_2 \vee \cdots \vee x_n$$

where $n \geqslant 4$ there is a (polynomially-sized) family of clauses of size 3 that is satisfiable only for truth assignments of the x_i that satisfy the original clause. Let's introduce some new variables (not occurring in this, or in any other clauses) z_3 through z_{n-1} and consider the family of clauses:

$$\begin{gathered} x_1 \vee x_2 \vee z_3 \\ \neg z_3 \vee x_3 \vee z_4 \\ \neg z_4 \vee x_4 \vee z_5 \\ \cdots \\ \neg z_{n-2} \vee x_{n-2} \vee z_{n-1} \\ \neg z_{n-1} \vee x_{n-1} \vee x_n. \end{gathered}$$

Suppose we have a satisfying assignment for this. Can all the x_i be false? If so, we need z_3 through z_{n-1} all t in order to satisfy all but the last clause. But then the last clause is not satisfied. Conversely, if any $x_i =$ t for i from 3 to $n-2$ we can set z_t to be t for $t \leqslant i$ and f for $i < t$ which satisfies all the clauses. If x_1 or x_2 is t we can make all the zs be f and if x_{n-1} or x_n is t we can make all the zs be t.

So the satisfying assignments for this new set of clauses are, restricted to the variables occurring in the literals x_i, exactly the same as the satisfying assignments for the original single clause.

Therefore, replacing each clause in an instance of SAT by a family of clauses of size 3 in this way gives a polynomial-time reduction from SAT to 3-SAT and therefore establishes that 3-SAT is **NP**-complete. □

Using 3-SAT rather than SAT in reductions is frequently convenient because of the fixed clause size.

3 Reducing 3-SAT to INDEPENDENT-SET

Recall:
INDEPENDENT-SET
Instance: A graph G and a positive integer k
Problem: Is there an independent set in G having (at least) k elements?

We will show a polynomial-time reduction of 3-SAT to INDEPENDENT-SET, thereby verifying that the latter is **NP**-complete. What does finding such a reduction entail?

INPUT: A set of clauses each containing exactly three literals
OUTPUT: A graph G and a parameter k such that the clauses are satisfiable if and only if G has a k-element independent set.

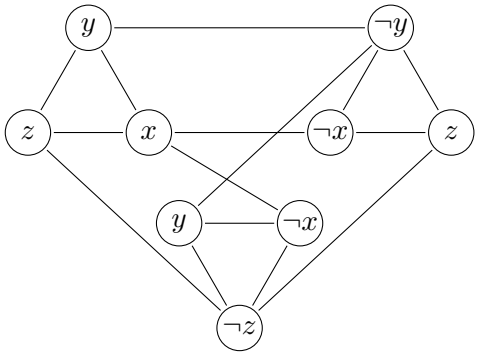
Of course crucially, the reduction has to be done in polynomial-time. In this case the idea is a very simple (but elegant) one. Let k be the number of clauses in the 3-SAT instance. For each clause $x \vee y \vee z$, include a triangle in G, giving a total of $3k$ vertices (label the vertices of each triangle with the corresponding literals). Then connect with an edge each pair of vertices representing contradictory literals (i.e., a variable and its negation).

Suppose first that the clauses have a satisfying assignment. Take one, and choose one vertex from each triangle that is t for the assignment. Since we can never select contradictory literals in this way and since we're only selecting one vertex per triangle, we get a k-element independent set.

Conversely, suppose that G has a k-element independent set. This must include exactly one vertex per triangle (since it can't have two vertices in any triangle). Form the truth assignment that arises from setting the literals corresponding to the independent set to be t. This is possible, since we can never be required to set both a literal and its negation to t as all such pairs are connected by edges. There may be some unassigned variables – set them arbitrarily. The resulting truth assignment has at least one true literal per clause, and so the 3-SAT instance was satisfiable.

The graph below exhibits the construction for the clauses:

$$x \vee y \vee z$$
$$\neg x \vee \neg y \vee z$$
$$\neg x \vee y \vee \neg z.$$



The satisfying assignment with all of x, y and z set to t corresponds to the independent sets using any one of the three in the upper left triangle, the z in the upper right triangle, and the y in the lower triangle. Of course other satisfying assignments exist, but each corresponds to one or more independent sets of size three. For instance, we could take $x = \mathsf{f}$, $y = \mathsf{f}$, $z = \mathsf{t}$ which would correspond to the z in the upper left, $\neg x$ or $\neg y$ in the upper right, and $\neg x$ in the lower triangle.

4 3-SAT to 3-COLOURING

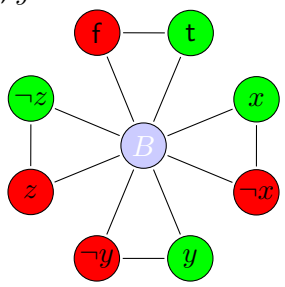
Again recall:
3-COLOURING
Instance: A graph G
Problem: Is there a 3-colouring of G?

This time to give the reduction we'll follow a slightly different strategy:

- First we'll define a part of the graph, call it the *core*, corresponding to the individual variables and their negations that occur in (any of) the clauses in such a way that each 3-colouring of that part corresponds to a unique truth assignment to the variables, then
- for each clause, we'll add a *gadget* that connects the corresponding literals (and possibly some other vertices) in such a way that the gadget can be three-coloured if and only if one of its literals is t.

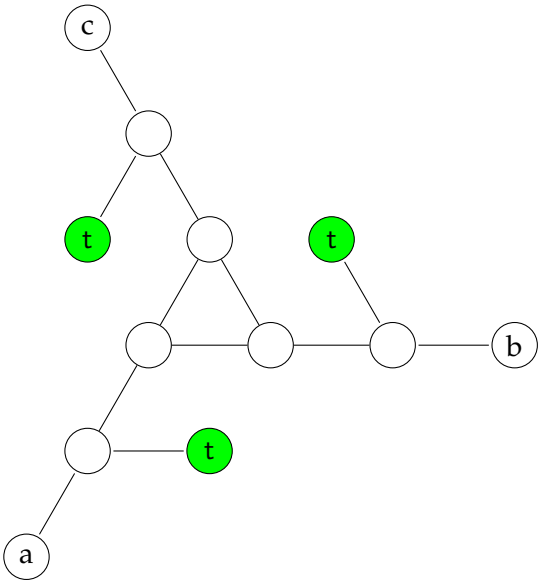
If there are a total of k distinct variables that occur in literals of the clauses (say x_1 through x_k), then the core will consist of $k + 1$ triangles, sharing a common base vertex. One triangle will have its other two vertices labelled t and f.

We illustrate the core below for the three variable case again together with the three colouring corresponding to $x =$ t, $y =$ t and $z =$ f:



What do the three-colourings of the core look like? The base vertex gets some colour (call it blue). The t-vertex also gets a colour, call it green, and the f-vertex gets the third colour, red. Then we are free to choose for each variable whether to colour it green and its negation red or vice versa. That is, each colouring of the core corresponds precisely to a truth assignment of the variables where all the variables sharing a colour with the t-vertex are assigned the value t while those sharing their colour with the f-vertex are assigned the value f.

Now consider any clause $a \vee b \vee c$ where each of a, b and c is some literal. Define the following gadget that uses the vertices of the core corresponding to a, b, c and t, together with six new vertices. Note that for the purposes of symmetry, three copies of the vertex t have been drawn – but they all stand for the same vertex of the core.



If a, b and c are all f, i.e., red and we try to 3-colour the gadget then each of the vertices adjacent to the corner of the triangles must be coloured blue. But then each of the vertices of the triangle would have to be red or green and two would be the same colour, so we cannot properly colour the gadget in that case.

However, if even one of a, b or c is t, then the adjacent vertex can be blue or red, and the corresponding vertex of the triangle could be any colour. Thus, in this case we can three colour the gadget.

If the 3-SAT instance is satisfiable then the corresponding three-colouring of the core can be extended to a three-colouring of the entire graph consisting of the core and all the gadgets. Conversely, if we have a three-colouring of the entire graph then, because the gadgets are properly three-coloured, each clause is satisfied in the truth assignment that corresponds to the colouring of the core.

So, we have achieved a reduction of 3-SAT to 3-COLOURING.

5 3-SAT to HAMILTON-CYCLE

It turns out to be more convenient to do this one via an intermediate problem:

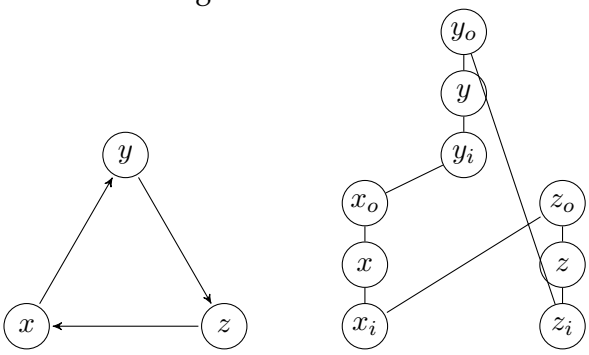
DIRECTED HAMILTON-CYCLE
Instance: A directed graph G
Problem: Is there a cycle in G that visits every vertex?

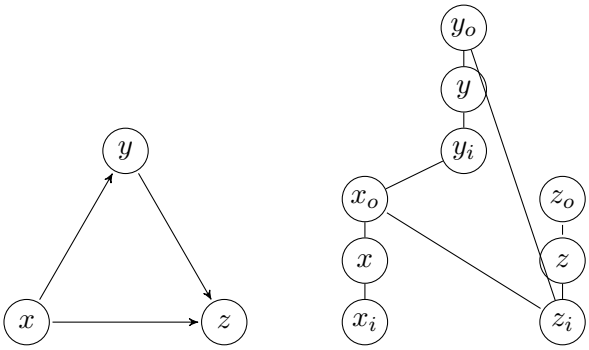
So, what's a *directed* graph? That's easy - edges are now ordered pairs of vertices and an edge (v, w) represents an edge from v to w. We still disallow multiple copies of the same edge (i.e., the edges are a set) but having both edges (v, w) and (w, v) are allowed. A cycle in a directed graph must use the edges in the "correct" direction. That is, if (v, w) is an edge but (w, v) is not then a cycle can have the form $\cdots vw \cdots$ but not $\cdots wv \cdots$.

We have to define two reductions: 3-SAT to DIRECTED HAMILTON-CYCLE and DIRECTED HAMILTON-CYCLE to HAMILTON-CYCLE. Let's deal with the second one first since it's an easy idea.

Given a directed graph G_d (i.e., an instance of DIRECTED HAMILTON-CYCLE) we'll construct an undirected graph, G_u with three times as many vertices. Each vertex v of G_d gets replaced by three vertices v_i ("v-in"), v, and v_o ("v-out") connected by edges $\{v_i, v\}$ and $\{v, v_o\}$. Every directed edge (v, w) defines an edge between v_o and w_i in G_u. It's clear we can construct G_u in polynomial time from G_d.

In the pictures below we show the undirected graph obtained from a triangle directed as a cycle, and one where two of the edges clash.





Suppose that G_d had a directed Hamilton cycle. Take a Hamilton cycle $v^1 v^2 \dots v^n$ of G_d (so $v^n = v^1$ and each of (v^i, v^{i+1}) is an edge of G_d for $1 \leqslant i < n$). Then:

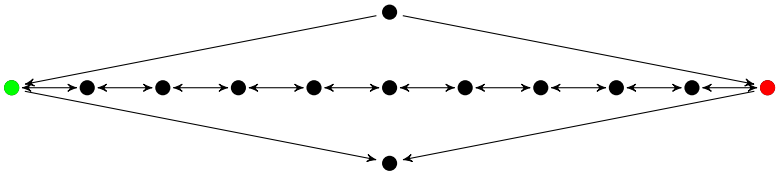
$$v_i^1\ v^1\ v_o^1\ v_i^2\ v^2\ v_o^2\ v_i^3\ \cdots\ v_i^{n-1}\ v^{n-1}\ v_o^{n-1}\ v_i^1$$

is a Hamilton cycle of G_u.

Conversely, if G_u has a Hamilton cycle choose some arbitrary vertex v as the start point. If the next vertex of the cycle is v_o then proceed as below. Otherwise it is some vertex v_i. In that case read the cycle in the opposite order (its only two neighbours are v_i and v_o so one of these holds, and in an undirected graph the reverse of a Hamilton cycle is a Hamilton cycle).

Now the full cycle has to take the form "go from some 'out' vertex to an 'in' vertex corresponding to an edge in G_d, from there to the corresponding 'original' vertex (or you'll miss it), then to the 'out' vertex for it ..." In other words if we just throw away all the 'in' and 'out' vertices in the undirected cycle we'll see a Hamilton cycle of the directed graph.

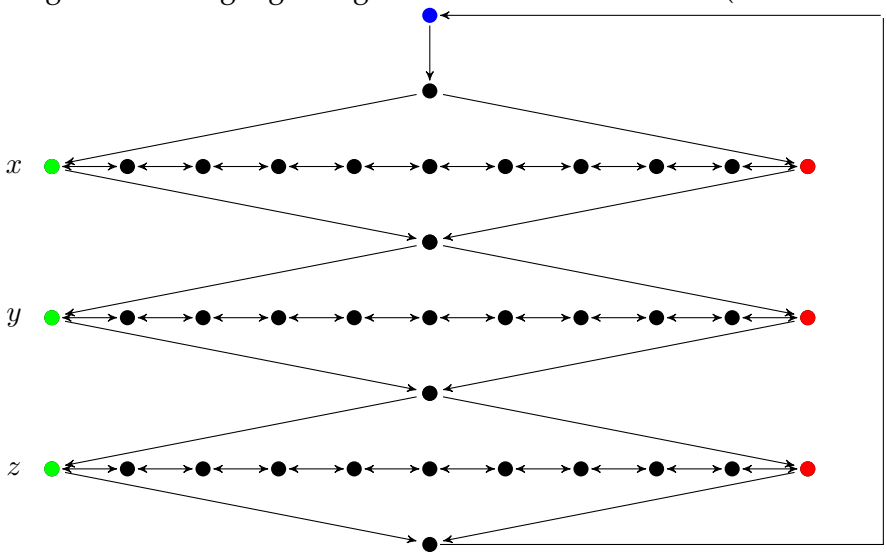
So now we need a reduction from 3-SAT to DIRECTED HAMILTON-CYCLE. The first part of the construction is to build a graph that has one possible directed Hamilton cycle for each assignment of truth values to some sequence of variables. For this purpose we introduce an individual "variable gadget" that looks like this:



The size of the midline should be three times the number of clauses we plan to include (so this gadget would be suitable for three clauses).

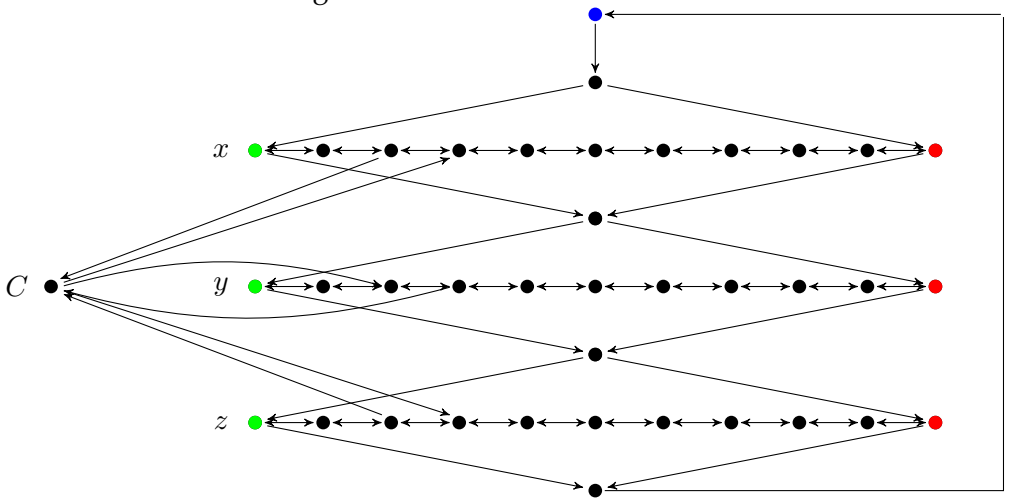
Notice the "two-way" arrows across the middle - they are really two one-way arrows but it's easier to draw them like this. If a Hamilton cycle in some graph containing this variable gadget enters at the top then it must go left (which we'll call "true") or right ("false"). That's why we coloured the left vertex green and the right one red. Then, in order not to miss the vertices across the middle it has to run across to the opposite side, before proceeding to the bottom.

Now we can glue variable gadgets together with one more vertex (coloured blue)

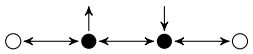


There are eight directed Hamilton cycles in this graph – one corresponding to each possible truth assignment for the three variables.

Now we need to add a little more for the clauses. Each clause will add just one new vertex. This will be joined in to the midline of the variable gadget in such a way that if the truth setting satisfies the clause then we can detour briefly to visit the clause vertex as we cross the midline of an appropriate variable. For instance suppose that our first clause is $C = x \vee \neg y \vee z$. Choose the first two vertices (on the left) inside the midline of each gadget. When the variable occurs positively, connect the first of these to the vertex for C, and C to the second one. When it occurs negatively, do the reverse. For this clause and our three variables we'd get:



Notice that each pair of vertices in a midline that we're planning to connect into a clause vertex are guarded on either side by a vertex that won't be connected that way. The local situation is (symmetric to):



Here the solid vertices are the ones connected to the clause vertex and the hollow ones are the "guards". Suppose that in a Hamilton cycle we use the upwards pointing edge to visit the clause vertex. How could that happen? If we arrived at the solid vertex that is its source from the right hand side then we can't use the edge connecting to the leftmost guard vertex. But that vertex has only one other incoming and outgoing edge and they're from the same vertex (the next to the left along the midline) so it couldn't be part of the

Hamilton cycle. So we must arrive from the left. Now we visit the clause vertex. Suppose that we don't immediately return to this midline. In that case the second solid vertex is cut off from the cycle – we aren't using the edge to the clause vertex, nor the edges to and from the preceding vertex on the midline – so again it has only one incoming and outgoing edge remaining and they have the same other endpoint meaning we can't create a cycle.

In other words, when we visit a clause vertex as part of a Hamilton cycle we must immediately return to the same midline or we'd have a contradiction. Ignoring our visits to the clause vertices gives us a Hamilton cycle of the original graph that glued the variable gadgets together, and therefore corresponds to a truth assignment of the variables. And, the mere ability to have visited a clause vertex from the midline of some variable gadget means this truth assignment must have been satisfying for each clause, and hence for the whole SAT-instance.

So if the big graph has a directed Hamilton cycle then the SAT-instance was satisfiable. Conversely, if we have a satisfying assignment then just take the corresponding Hamilton cycle in the variable gadget graph, detouring to visit each clause at some point when you're crossing the midline of a variable gadget in an appropriate direction.

6 Tutorial problems