UCT

$\mathbb{NP}$ and Completeness

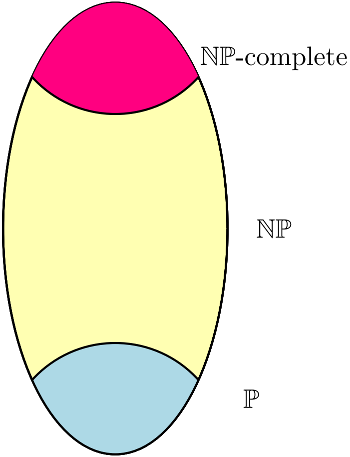
Klaus Sutner

Carnegie Mellon University

Spring 2024

- We have a complexity class $\mathbb{NP}$ that seems to be a proper extension of $\mathbb{P}$.
- We have a natural problem SAT in $\mathbb{NP}$ that works well as a target in numerous reductions.
- We need to show that SAT is indeed $\mathbb{NP}$-complete.

This should help in any attempt to separate the classes: $\mathbb{P} = \mathbb{NP}$ iff SAT is in $\mathbb{P}$.


$\mathbb{NP}$-complete
$\mathbb{NP}$
$\mathbb{P}$

We already have a promising reduction: polynomial time many-one reducibility $A \leq_m^p B$:

$$x \in A \iff f(x) \in B$$

Polynomial time reducibility is is nicely compatible with $\mathbb{P}$ and $\mathbb{NP}$. Log-space reductions are another plausible option.

To produce an $\mathbb{NP}$-complete problem we have two basic options:

- Try to scale down the Halting problem (add polynomial time bounds everywhere and keep all fingers crossed).
- Work on a particular natural problem, in particular SAT.

We can construct an enumeration $(M_e)_e$ of all polynomial time Turing machines: just run a clock and stop the machine after $n^e + e$ steps if it has not halted already.

But then there is a universal, deterministic machine $\mathcal{U}$ that simulates M_e with only a polynomial slowdown. More precisely, $\mathcal{U}$ on input $e \# x$ simulates M_e on x in running time $q(n^e + e)$ where q is some polynomial (even low degree).

Of course, $\mathcal{U}$ itself is not a polynomial time machine, the running time increases for different choices of e.

Likewise, there is a universal, nondeterministic machine $\mathcal{U}_{\mathsf{nd}}$ that simulates nondeterministic polynomial time machines N_e with only a polynomial slowdown in the same sense as above. Again, $\mathcal{U}_{\mathsf{nd}}$ itself is not polynomial time.

So, one has to be a bit careful where the polynomial time bounds should go.

Same old, same old: Let's try to scale down the Halting set.

So we want to use the universal machine $\mathcal{U}_{\mathsf{nd}}$ that can simulate machines in the enumeration (N_e) of nondeterministic, polynomial time Turing machines just mentioned. A first shot would be to define

$$K = \{ e \# x \mid x \text{ accepted by } N_e \}$$

It is easy to see that K is $\mathbb{NP}$-hard, but there is no reason why it should be in $\mathbb{NP}$; simulation of N_e is not a task that can be handled within a fixed polynomial time bound ($\mathcal{U}_{\mathsf{nd}}$ itself is not polynomial time).

Or, in terms of witnesses: we would have to commit to some fixed polynomial to bound the size of witnesses for K, but the witnesses for $\mathcal{L}(N_e)$ could be arbitrarily much larger.

No good.

We can fix the problem by padding [†] the input so that we can compensate for the running time of N_e.

$$\widehat{K} = \{\, 0^t \# e \# x \mid x \text{ accepted by } N_e \text{ in } \leq t = |x|^e + e \text{ steps } \}$$

Proposition

$\widehat{K}$ is in $\mathbb{NP}$.

Proof.

To see this note that the slowdown by $\mathcal{U}_{\mathsf{nd}}$ is polynomial, say, the simulation takes $q(n^e + e)$ steps.

But then $\mathcal{U}_{\mathsf{nd}}$ can test, in time polynomial in $|0^t \# e \# x| = t + |e| + |x| + 2$, whether N_e indeed accepts x in the required time. □

[†] This is similar to the construction that shows full projections move from $\mathbb{P}$ to full semidecidable (whence we use polynomially bounded projections).

Proposition

$\widehat{K}$ *is* $\mathbb{NP}$*-hard.*

Proof.

Consider $A = \mathcal{L}(N_e) \in \mathbb{NP}$ arbitrary. Then the function

$$x \mapsto 0^{|x|^e + |e|} \# e \# x$$

is polynomial time computable and shows that $A \leq_m^p \widehat{K}$.

□

Hence, $\widehat{K}$ is indeed $\mathbb{NP}$-complete.

So we have the desired existence theorem.

Theorem

There is an $\mathbb{NP}$-complete language.

Alas, this result is perfectly useless when it comes to our list of interesting $\mathbb{NP}$ problems: they bear no resemblance whatsoever to $\widehat{K}$.

We have a foothold in the world of $\mathbb{NP}$-completeness, but to show that one of these natural problems is $\mathbb{NP}$-complete we would have to find a reduction from $\widehat{K}$ to, say, Pebbling or Vertex Cover.

Good luck on that.

Our first $\mathbb{NP}$-complete problem $\widehat{K}$ is a bit artificial, but it still helps to bring the separation question for $\mathbb{P}$ and $\mathbb{NP}$ into sharp focus:

Claim: $\mathbb{P} \neq \mathbb{NP}$ iff $\widehat{K} \notin \mathbb{P}$.

This type of claim will be much sharper if we show that a practical problem such as SAT is $\mathbb{NP}$-complete.

Our padded version of Halting

$$\widehat{K} = \{\, 0^t \# e \# x \mid x \text{ accepted by } N_e \text{ in } \leq t = |x|^e + e \text{ steps } \}$$

has a remarkable property: the following intersection problem is undecidable.

Problem:	**Regular Intersection Emptiness (for $\widehat{K}$)**
Instance:	A regular language R.
Question:	Is $R \cap \widehat{K}$ empty?

Exercise

Why should this be difficult?

Recall your favorite decision problems about Boolean formulae:

Problem:	**Satisfiability**
Instance:	A Boolean formula φ.
Question:	Is φ satisfiable?

Problem:	**Tautology**
Instance:	A Boolean formula φ.
Question:	Is φ a tautology?

SAT is clearly in $\mathbb{NP}$, TAUT in co-$\mathbb{NP}$. And, we have seen several examples of reductions from $\mathbb{NP}$ to SAT. One might suspect that SAT is so expressive, it can be used to code up any problem in $\mathbb{NP}$ (recall, it's a watered down version of the Entscheidungsproblem).

This motivates the following result.

Theorem (Cook-Levin 1971/1973)

The Satisfiability Problem is $\mathbb{NP}$-complete.

Membership in $\mathbb{NP}$ is easy using the standard guess-and-verify approach.

But hardness takes work: we need to find an abstract argument that shows that any problem in $\mathbb{NP}$ already can be expressed in terms of SAT.

Note that the following proof is strictly read-once: read it, then throw it away and reconstruct your own proof.

Let A be an arbitrary set in $\mathbb{NP}$. Then there is some polynomial time decidable marked language L such that $A = \text{proj}_P(L)$. We may safely assume that the witness w has length $n' = p(n)$, $n = |x|$.

So there is a deterministic polynomial time Turing machine M such that M accepts $w\#x$ for some $w \in \mathbf{2}^{n'}$ iff $x \in A$.

The idea is to construct a (rather large) Boolean formula Φ_x such that

$$\Phi_x \text{ is satisfiable} \iff M \text{ accepts } w\#x \text{ for some } w \in \mathbf{2}^{n'}.$$

While the formula is fairly long, it can easily be constructed from x and M in time polynomial in n.

As always, we need to be clear about the meaning of the Boolean variables. First off, let $N = q(n)$ be the running time of the machine.

If we have a list of Boolean variables

$$X_0, X_1, \ldots, X_N$$

and a truth assignment σ we can think of $\sigma(X_t)$ as the value of variable X at time t.

We will use Φ_x to pin down the value of X_{t+1} in terms of X_t (and other variables).

$$\bigwedge_{t<N} X_{t+1} \iff \varphi(\ldots, X_t, \ldots)$$

Thus, specifying the value of X_0 for all variables pins down all values.

If we need to code a number r in a certain range, say $1 \leq r \leq s$, we can simply use variables

$$X(1), X(2), \ldots, X(s)$$

plus a stipulation that exactly one of them is true under σ:

$$\mathsf{CNT}_{1,s}\big(X(1), X(2), \ldots, X(s)\big)$$

Here $\mathsf{CNT}_{1,s}(x_1, \ldots, x_s)$ is the counting function that we've encountered before. Note that this formula has size $O(s^2)$, which is OK as long as the number of variables is polynomial in n.

We could use a more concise representation writing r in binary using only $\log s$ bits, but there is no need for this.

Let $m = |Q|$ and $\gamma = |\Gamma|$. Combining these two ideas we can set up polynomially many Boolean variables

states	$S_t(p)$	$0 \leq t \leq N$, $1 \leq p \leq m$
head position	$H_t(i)$	$0 \leq t, i \leq N$
tape inscription	$T_t(i, a)$	$0 \leq t, i \leq N$, $1 \leq a \leq \gamma$

that express, for each time $0 \leq t \leq N$, which state the machine is in, where the head is, and what's on the tape.

The computation has length at most N and we may safely assume that the tape head travels no further (one-way infinite tape): at time 0 we use $p(n) + n + 1$ cells, and we can adjust q accordingly. Hence $H_t(i)$, $i \leq N$ suffices.

Here are some of the conjuncts making up Φ_x:

$$\Phi_1 = \bigwedge_{t \leq N} \mathsf{CNT}_{1,m}(S_t(1), \ldots, S_t(m))$$

$$\Phi_2 = \bigwedge_{t \leq N} \mathsf{CNT}_{1,N}(H_t(0), \ldots, H_t(N))$$

$$\Phi_3 = \bigwedge_{i,t \leq N} \mathsf{CNT}_{1,\gamma}(T_t(i,1), \ldots, T_t(i,\gamma))$$

Clearly, any satisfying truth assignment fixes, for each time t, a unique state, head position, and tape inscription (exactly one symbol from the tape alphabet in each tape cell).

We need to express the constraint that the variables change from time t to time $t+1$ only in accordance with the transition function of the Turing machine.

More pieces of Φ_x:

$$\Phi_4 = \bigwedge_{\substack{t<N \\ \delta(p,a)=(q,b,\Delta)}} S_t(p) \wedge H_t(i) \wedge T_t(i,a) \Rightarrow S_{t+1}(q) \wedge H_{t+1}(i+\Delta) \wedge T_{t+1}(i,b)$$

$$\Phi_5 = \bigwedge_{t,i<N} \neg H_t(i) \Rightarrow \bigwedge_a T_{t+1}(i,a) = T_t(i,a)$$

Initially we are in state q_0, the head is at cell 0; at the end we accept:

$$\Phi_6 = H_0(0) \wedge S_0(q_0) \wedge S_N(q_Y).$$

The last part, Φ_7, specifies the initial tape contents like so. The following are all true

$$\begin{aligned} T_0(0, \text{\textvisiblespace}) &\qquad \text{blank for tapehead} \\ T_0(n'+1, \#) &\qquad \text{separator} \\ T_0(n'+2, x_1), \ldots, T_0(n'+n+2, x_n) &\qquad \text{actual input} \\ T_0(n'+n+3, \text{\textvisiblespace}), \ldots, T_0(N, \text{\textvisiblespace}) &\qquad \text{blank tape} \end{aligned}$$

whereas the following are unspecified

$$T_0(1, a), \ldots, T_0(n', a) \qquad \text{space for witness}$$

Finally, assemble the pieces in a big conjunction:

$$\Phi_x = \Phi_1 \wedge \ldots \wedge \Phi_7$$

Claim

The whole formula Φ_x has size polynomial in n.

Proof. Count. □

Now suppose Φ_x is satisfied by truth assignment σ.

By Φ_1, Φ_2, Φ_3 assignment σ defines, for each time t,

- a unique state p_t,
- a unique head positioned h_t,
- a unique tape inscription C_t,

By Φ_7, inscription C_0 looks like

$$\llcorner w_1 \ldots w_{n'} \# x_1 \ldots x_n \llcorner \ldots \llcorner.$$

By Φ_4, Φ_5 and Φ_6, the sequence $(p_t, h_t, C_t)_{t \leq N}$ correctly describes an accepting computation of M on $w\#x$.

Conversely, every witness plus corresponding accepting computation can be translated into a satisfying truth assignment σ.

That's it. □

Exercise

Determine the size of Φ_x more precisely. Convince yourself that the formula can be constructed in logarithmic space.

Exercise

Our construction simulates a deterministic Turing machine on a marked language. Instead, use a nondeterministic machine directly (this requires only minor modifications in the formula).

General Principle: after each hardness argument, try to understand what limitations can be placed on the instances without ruining hardness.

For example, for Halting we don't need $\{e\}(x)$, just $\{e\}(e)$ is enough. In fact, even $\{e\}()$ is fine.

This is often important for future hardness arguments: it is easier to deal with a limited set of instances.

Corollary

The Satisfiability Problem is $\mathbb{NP}$-complete for formulae in 3-CNF.

Our original argument constructs a Boolean formula Φ_x without any particular regard for a normal form.

But: closer inspection shows that the overall structure is one big conjunction.

It is not difficult to rewrite all the subformulae into disjunctions of literals.

This may slightly increase the size of the formula, but only by a polynomial amount.

In other words, we can easily force Φ_x to be in CNF. From there, it is straightforward to get to 3-CNF by splitting long clauses.

A better solution is to think more structurally and to show that any formula Φ can be associated with another formula Φ' in CNF that is equisatisfiable and only polynomially larger than Φ.

The problem here is the following: The formula

$$\varphi = (p_{10} \land p_{11}) \lor (p_{20} \land p_{21}) \lor \ldots \lor (p_{n0} \land p_{n1})$$

is in DNF, but conversion to CNF using the standard rewrite rules produces the exponentially larger formula

$$\varphi \equiv \bigwedge_{f:[n]\to\mathbf{2}} \bigvee_{i\in[n]} p_{if(i)}$$

Note that there appear to be no short-cuts: the 2^n disjunctions of length n must all appear.

Consider the formula

$$\varphi_n = p_1 \oplus p_2 \oplus \ldots \oplus p_n$$

where $\oplus$ is exclusive or.

Here both DNF and CNF have 2^{n-1} terms of length n each.

Exercise

Figure out what these normal forms look like.
Try to reason why they are in fact minimal.

Thinking outside of the Box: For equisatisfiability, we don't need to stick to the original set of variables: we could add a few.

For a propositional variable p we let $q_p = p$. For the whole formula φ we introduce a clause $\{q_\varphi\}$. Otherwise we introduce clauses for all subformulae of Φ as follows:

$$q_{\neg\psi} : \{q_\psi, q_{\neg\psi}\}, \{\neg q_\psi, \neg q_{\neg\psi}\}$$

$$q_{\psi\vee\varphi} : \{\neg q_\psi, q_{\psi\vee\varphi}\}, \{\neg q_\varphi, q_{\psi\vee\varphi}\}, \{q_\varphi, q_\psi, \neg q_{\psi\vee\varphi}\}$$

$$q_{\psi\wedge\varphi} : \{q_\psi, \neg q_{\psi\wedge\varphi}\}, \{q_\varphi, \neg q_{\psi\wedge\varphi}\}, \{\neg q_\psi, \neg q_\varphi, q_{\psi\wedge\varphi}\}$$

The intended meaning of q_ψ is pinned down by these clauses, e.g.

$$q_{\psi\vee\varphi} \equiv q_\psi \vee q_\varphi$$

Consider again the formula

$$\varphi = (p_{10} \wedge p_{11}) \vee (p_{20} \wedge p_{21}) \vee \ldots \vee (p_{n0} \wedge p_{n1})$$

Set $B_k = (p_{k0} \wedge p_{k1})$ and $A_k = B_k \vee B_{k+1} \vee \ldots \vee B_n$ for $k = 1, \ldots, n-1$. Thus, $\varphi = A_1$ and all the subformulae other than variables are of the form A_k or B_k.

The clauses in the Tseitin form of φ are as follows (we ignore the variables):

- q_{A_k}: $\{q_{B_k}, \neg q_{B_k \wedge A_{k-1}}\}, \{q_{A_{k-1}}, \neg q_{B_k \wedge A_{k-1}}\}, \{\neg q_{B_k}, \neg q_{A_{k-1}}, q_{B_k \wedge A_{k-1}}\}$
- q_{B_k}: $\{p_{k0}, \neg q_{B_k}\}, \{p_{k1}, \neg q_{B_k}\}, \{\neg p_{k1}, \neg p_{k0}, q_{B_k}\}$

Exercise

Make sure you understand in the example how any satisfying assignment to φ extends to a satisfying assignment of the Tseitin CNF, and conversely.

Theorem

Let Γ be the set of clauses in Tseitin CNF for formula ϕ. Then Γ and ϕ are equisatisfiable. Moreover, C can be constructed in time linear in the size of ϕ.

Proof.

$\Rightarrow$: Suppose that $\sigma \models \Gamma$.

An easy induction shows that for any subformula ψ we have $[\![\psi]\!]_\sigma = [\![q_\psi]\!]_\sigma$. Hence $[\![\phi]\!]_\sigma = [\![q_\phi]\!]_\sigma = 1$ since $\{q_\phi\}$ is a clause in C.

$\Leftarrow$: Assume that $\sigma \models \phi$.

Define a new valuation τ by $\tau(q_\psi) = [\![\psi]\!]_\sigma$ for all subformulae ψ. It is easy to check that $\tau \models \Gamma$.

$\square$

A closer look at the construction of Φ_x reveals that $O(\log n)$ memory is sufficient. For example, to generate

$$\bigwedge_{t \leq N} \mathsf{CNT}_{1,m}(S_t(1), \ldots, S_t(m))$$

it suffices to have a counter for t, several auxiliary counters that spell out the counting formula and a few pointers into the formula (depending on coding details).

Since we do not charge for the output tape, this requires no more than logarithmic memory. Hence we actually have a logarithmic space reduction.

Recall our simple arithmetic programming language:

initialize	$x = 0$
assignments	$x = y$
increment	$x = x + 1$
sequential composition	$P; Q$
control	do x : P od

We are interested in Inequivalence for loop programs of depth 1. As mentioned, membership in $\mathbb{NP}$ is quite difficult to establish, but hardness is now quite straightforward.

This lay of the land is unusual, but it does occur occasionally.

For hardness we show how to reduce 3SAT to Inequivalence.

Suppose we have Boolean variables $x_1, x_2, \ldots, x_n$ and clauses $C_1, C_2, \ldots, C_m$.

Now consider a clause C_i, say, $C_i = x \vee \overline{y} \vee z$. We compute the truth value $c_i \in \mathbf{2}$ of C_i as follows:

```
c = 0;
if( x == 1 ) c = 1;
if( y == 0 ) c = 1;
if( z == 1 ) c = 1;
```

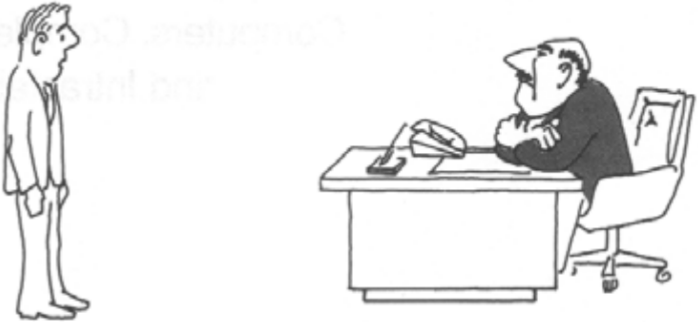
Lastly, we compute $\min(c_1, \ldots, c_m)$, the truth value of the whole formula.

The corresponding program is easily Loop(1) (it operates solely on Boolean values and does not begin to exploit the possibilities of arithmetic) and inequivalent to $\mathbf{0}$, the constant-zero program, iff the formula is satisfiable.

Satisfiability is a tremendously important practical problem, but if this were the only relevant $\mathbb{NP}$-complete problem the whole notion would still be somewhat academic.

But as Richard Karp realized after reading Cook's paper, there are dozens (actually: thousands) of combinatorial problems that all turn out to be $\mathbb{NP}$-complete. So none of them will admit a polynomial time solution unless $\mathbb{P} = \mathbb{NP}$.

The proof method is interesting: some problems are proven hard by direct reduction from SAT, then these are used to show other problems are hard, and so on ... By transitivity one could, in principle, produce a direct reduction from SAT, but in reality these direct reductions are often very hard to find.



"I can't find an efficient algorithm, I guess I'm just too dumb."

"I can't find an efficient algorithm, because no such algorithm is possible!"

"I can't find an efficient algorithm, but neither can all these famous people."

To find a reduction from SAT to some combinatorial problem it is usually quite a bit easier to deal with just 3-SAT (which we also know to be $\mathbb{NP}$-complete).

It now suffices to find a polynomial time computable function

$$f : \text{3-CNF} \longrightarrow \text{Graphs} \times \text{Integers}$$

such that φ in 3-CNF is satisfiable iff for $f(\varphi) = (G, k)$ the graph G has a vertex cover of size k.

Again, by transitivity one could, in principle, produce a direct reduction, but that may be significantly more difficult.

Theorem

Vertex Cover is $\mathbb{NP}$-complete.

Proof.

Suppose we have a 3-CNF formula $\Phi = \Phi_1 \wedge \Phi_2 \wedge \ldots \wedge \Phi_m$ where $\Phi_i = \{z_{i,1}, z_{i,2}, z_{i,3}\}$.

The Boolean variables are $x_1, \ldots, x_n$.

We start with a graph G' on $2n + 3m$ vertices.

- Vertices: x_i, $\overline{x}_i$ for $i = 1, \ldots, n$ and $u_{i,1}$, $u_{i,2}$, $u_{i,3}$ for $i = 1, \ldots, m$.
- Edges: one edge between x_i and $\overline{x}_i$, and three edges that turn $u_{i,1}$, $u_{i,2}$, $u_{i,3}$ into a triangle.

These are truth-setting edges and clause-edges, respectively.

It is easy to see that every vertex cover of G' must have at least $n + 2m$ vertices: one for each truth-setting edge, and two of the clause-edges (which form a triangle).

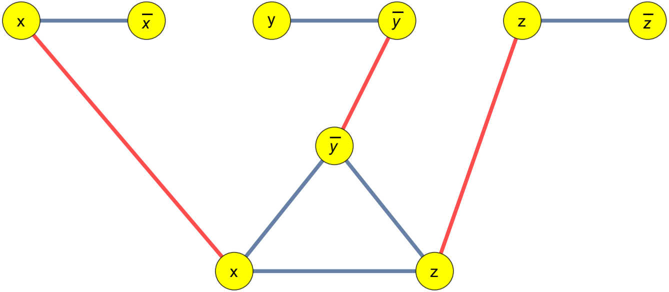
These choices are arbitrary, so there are lots of these covers.

So far we have only used n and m, but not the formula itself.

Let G be the graph obtained by adding $3m$ more link-edges to G':

- if $z_{i,j} = x_s$ connect $u_{i,j}$ to x_s
- if $z_{i,j} = \neg x_s$ connect $u_{i,j}$ to $\overline{x}_s$

Lastly, set the bound to $k = n + 2m$.


x
$\overline{x}$
y
$\overline{y}$
z
$\overline{z}$
$\overline{y}$
x
z

Claim

G has a cover of size $k = n + 2m$ iff the formula is satisfiable.

To see this, note that any cover C defines an assignment σ:

$$\sigma(x_i) = \begin{cases} 1 & \text{if } x_i \in C, \\ 0 & \text{otherwise.} \end{cases}$$

Then σ satisfies the formula: one vertex in each clause triangle is not in C; its link-edge must be covered from the other end. Hence the corresponding literal is true by construction.

Conversely, every satisfying assignment translates into a cover.

□

For this construction to work we need two crucial ingredients. Suppose $f(\Phi) = (G, k)$.

- The graph G and the bound k can be computed from Φ in polynomial time.
- G has a vertex cover of size k **if, and only if,** Φ is satisfiable.

Many other completeness proofs look very similar: it is trivial to see that the problem is in $\mathbb{NP}$, and it requires work (sometimes a lot of it) to produce hardness.

Corollary

Independent Set and Clique are $\mathbb{NP}$-complete.

There is no need to prove this from scratch, instead we can exploit the logical connection between vertex covers, independent sets and cliques.

Exercise

Prove that Independent Set and Clique are $\mathbb{NP}$-complete.

Theorem

Hamiltonian Cycle is $\mathbb{NP}$*-complete.*

Note that this is a clean decision problem, taken directly from graph theory. There is no artificial bound to force things into this format.

A similar problem is Hamiltonian Path: we are looking for a path that touches every vertex exactly once (but need not form a cycle).

Membership is obvious, for hardness reduce from Vertex Cover.

Let $G = \langle V, E \rangle$ be a ugraph, and k a bound, $1 \leq k \leq n = |V|$. We may assume wlog that all vertices have degree at least 2 (why?).

For each v, fix an enumeration u_i^v, $i \in [\deg(v)]$, of all its neighbors.

Define a new graph H as follows:

Vertices
- anchor vertices $a_1, \ldots, a_k$
- box vertices $\boxed{e, v, i}$ for all $v \in e \in E$, $i \in [6]$.

Edges
- chain edges (connecting anchors and boxes)
- box edges (inside a box)

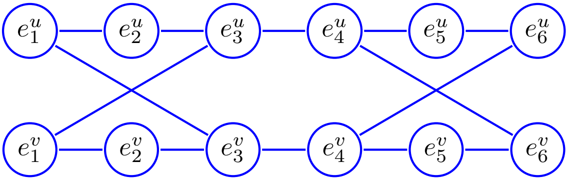
Chain Edges:

- $\{a_j, \boxed{e, u^v_1, 1}\}$ and $\{a_j, \boxed{e, u^v_{\deg(v)}, 6}\}$ for all $j \in [k]$.
  These edges connect the anchor points to the first and last box on the v-chain. e is understood to be $\{v, u^v_i\}$.
- $\{\boxed{e, u^v_i, 6}, \boxed{e, u^v_{i+1}, 1}\}$ for $1 \leq i < \deg(v)$,
  These edges connect two consecutive boxes on the v-chain.

Box Edges:

$\{\boxed{e, v, i}, \boxed{e, v, i+1}\}$, $\{\boxed{e, v, 1}, \boxed{e, u, 3}\}$ and $\{\boxed{e, v, 4}, \boxed{e, u, 6}\}$ for all $\{u, v\} = e \in E$.

These edges form a 12-point gadget, a box that appears on the v-chain.



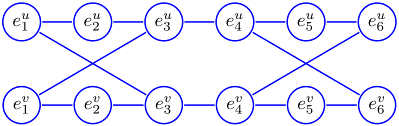
The box representing edge $e = \{u, v\}$.

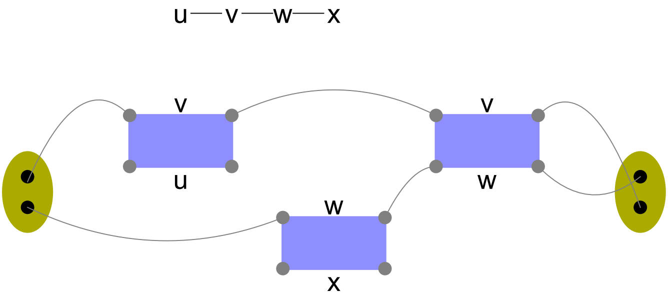
It is connected to the rest of the graph only at the 4 corners (to form a chain, and connect to the anchor vertices).

Now assume that P is a Hamiltonian cycle in G.

Claim 1: P must enter and exit each box at the same side. P can pass through the e-box in exactly one of two ways: type full (covers all vertices) or type half (covers only the points on the side where it entered).

To see this, take a pen and try to traverse the box. There simply are no other possibilities.





u—v—w—x
v
u
v
w
w
x

Claim 2: Without loss of generality, P consists of blocks

$$a_i, \boxed{e, u_1^v, 1}, \ldots, \boxed{e, u_{d(v)}^v, 6}, a_{i+1}$$

going from an anchor point to another, and containing the whole v-chain.

Claim 3: G has a vertex cover of size k.

To see this define $C = \{\, v \in V \mid P \text{ uses the } v\text{-chain } \}$. Now let $e = \{u, v\} \in E$. As P passes through the e-box it must use the u-chain or v-chain. Thus C covers e.

Suppose G has a vertex cover of size k.

Claim 4: Then H has a Hamiltonian cycle.

Construct P as follows: P has k blocks $a_i, \boxed{e, u_1^v, 1}, \ldots, \boxed{e, u_{d(v)}^v, 6}, a_{i+1}$ where v is in C.

The way P passes through the e-box on the v-chain is determined by whether $u \in C \wedge v \in C$ (type half) or $u \in C \oplus v \in C$ (type full).

Thus G has a VC of size k iff H is Hamiltonian. Clearly, G can be constructed in polynomial time.

$\square$

Corollary

Traveling Salesman is $\mathbb{NP}$-complete.

Corollary

Hamiltonian Path is $\mathbb{NP}$-complete.

Corollary

Longest Path is $\mathbb{NP}$-complete.

Exercise

Verify all these claims.