Recall that for a class of languages $C$, we had $A \subseteq C^*$ is $C$-HARD if for every $B \in C$, $B \leq^m A$.

Then if $A \subseteq C$ as well, $A$ is $C$-COMPLETE.

There is a snag, however. There is a second requirement we should place on completeness. For example, if our $C = \text{NP}$, then certainly any reduction from all $B \in \text{NP}$ to $A$ will suffice to show $A$ is hard for $\text{NP}$. The worse the required reduction, the harder $A$ must be.

(Actually this isn’t true, b/c a reduction could solve the problem and ask some trivial yes to a simple TM)

But for $A$ to be complete for $\text{NP}$ we want to be able to solve every problem in $\text{NP}$ using $A$. This means we need a poly time reduction from everything to $A$ (this can be nondeterministic poly time in theory but in practice never is). We denote this by $\leq^p_{\text{NP}}$ (or sometimes $\leq_p$). Let’s formalize this.

A function $f: \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function if some polynomial time TM $M$ exists so that, on input $w \in \Sigma^*$, halts with just $f(w)$ on its tape.

A language $A$ is polynomial time mapping reducible to a language $B$, written $A \leq^p \text{m} B$, if there is a polynomial time computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that $\forall w \in \Sigma^*$, $w \in A \iff f(w) \in B$.

The function $f$ is called the polynomial time reduction of $A$ to $B$.

Why does this matter? Well, let’s look at a useful theorem.
If \( A \leq^P B \) and \( B \in P \), then \( A \in P \).

**Proof.** Suppose \( A \leq^P B \) and \( B \in P \).

Then there is a polytime reduction \( f \) from \( A \) to \( B \) and a polytime TM \( M_B \) that decides \( B \). Consider the following TM \( M_A \):

\[
M_A = \text{"On input } w, \text{ do:}
\begin{align*}
   &1. \text{ Compute } f(w) \\
   &2. \text{ Run } M_B(f(w)) \\
   &3. \text{ Accept if } M_B \text{ does, reject if not.} 
\end{align*}
\]

Clearly, \( L(M_A) = A \) and \( M_A \) decides \( A \) in polytime.

The same argument gives us a theorem about \( NP \).

If \( A \leq^P B \) and \( B \in NP \), then \( A \in NP \).

We can get negative results with small resource bounds too.

Before we had \( A \leq^P B \) and \( A \in RE \Rightarrow B \in RE \)

**Cor.** If \( A \leq^P B \) and \( A \in P \), then \( B \in P \).

**Proof.** If \( B \in P \), then there is a polytime decision \( D \) for \( B \). There's also a polytime reduction \( f \) from \( A \) to \( B \). So \( D(f) \) decides \( A \) in polytime, which is nonsense, so no such \( D \) exists.

**Cor.** If \( A \leq^P B \) and \( A \in NP \), then \( B \in NP \).

**Proof.** Identical to the prior proof but with \( NP \).

You can also get results in the opposite direction.
If \( A \leq^p_m B \) and \( A \in \text{AEP-HARD} \), then \( B \in \text{NP-HARD} \).

**Proof:** Since \( A \in \text{AEP-HARD} \), \( \exists C \in \text{CNP} \) such that \( C \leq_m A \). But \( A \leq^p_m B \), so \( \exists C \in \text{CNP} \), it must be the case that \( C \leq^p_m B \).

Thus \( B \in \text{NP-HARD} \).

**Corollary:** If \( A \leq^p_m B \) and \( B \in \text{NP-HARD} \), then \( A \not\in \text{NP-HARD} \).

**Proof:** Since \( B \in \text{NP-HARD} \), \( \exists C \in \text{CNP} \) such that \( C \not\leq^p_m B \). If \( A \in \text{AEP-HARD} \), then \( C \leq^p_m A \leq^p_m B \), so \( C \leq^p_m B \), which is nonsense, so \( A \not\in \text{NP-HARD} \).}

Remember that these reductions can be read as:

- \( A \leq^p_m B \) = "\( A \) is no harder than \( B \)"
- \( A \leq^p_m B \) = "\( B \) is at least as hard as \( A \)"

To show a language \( A \) is \text{NP-COMPLETE}, there are two ways to do so conveniently:

- Show \( A \in \text{NP} \) and \( B \leq^p_m A \) for some \( B \in \text{NP-HARD} \).
- Show for some \( B \in \text{NP-COMPLETE} \) that \( A \leq^p_m B \) and \( B \leq^p_m A \).

Sometimes the reduction \( B \leq^p_m A \) is basically the same in reverse, so getting \( A \leq^p_m B \) is quick and easy.

The key point of all this is that \text{NP} problems are "probably hard," but \text{NPC} problems are "almost certainly hard." We don't know which of the following cases are true, so we don't know how hard these problems actually are.

\[ \text{NPC} \subseteq \text{NP} \]

\[ \text{P} \subseteq \text{NP} \]

\[ \text{P} \neq \text{NP} \]

\[ \text{NPC} \subseteq \text{NP} \]

\[ \text{P} \subseteq \text{NPC} \]
If we show any \( \text{BENPC} \) and \( \text{BEP} \), then we have \( \text{P} = \text{NP} \).

Similarly, if we could show any \( \text{BENP} \) and \( \text{BEP} \), then we would know \( \text{P} \cap \text{NP} = \emptyset \).

Obviously, we need a first \( \text{NPC} \) problem to do anything useful with these definitions. (Technically knowing \( \text{A\#NP} \) and \( \text{A\#NP}-\text{HARD} \) is also useful, but let's move on). The problem we pick is \( \text{SAT} \).

\[
\text{SAT} = \{ \phi \mid \phi \text{ is a satisfiable Boolean formula} \}
\]

Proving \( \text{SAT} \in \text{NPC} \) is hard. The theorem is known as the Cook-Levin Theorem. We will prove this if time allows, but let's first just accept it as true and look at a bunch of \( \text{NPC} \) problems.

A literal of a Boolean formula is a variable or its negation.

A clause is a disjunction of literals (i.e. \( x_1 \lor x_2 \lor x_4 \)).

A Boolean function in conjunctive normal form (\( \text{CNF} \)) is a conjunction of clauses (i.e. \( (x_1 \lor x_2) \land (x_3 \lor x_4) \)). This is also called a \( \text{CNF-formula} \).

A 3\text{CNF-formula} is a \( \text{CNF-formula} \) such that each clause has exactly 3 literals.

\[
\text{3SAT} = \{ \phi \mid \phi \text{ is a satisfiable } 3\text{CNF-formula} \}
\]

That \( \text{3SAT} \in \text{NPC} \) follows from a modification of the Cook-Levin theorem as it is easier than a reduction \( \text{SAT} \leq^p \text{3SAT} \).

It's often the case that \( \text{3SAT} \) is easier to work with.
Given an undirected graph $G=(V,E)$, a clique is a $V' \subseteq V$ such that $orall u, v \in V'$ (with $u \neq v$), $(u, v) \in E$. A $k$-clique is a clique of size $k$.

Ex)

We define the decision problem for clique as

$$\text{CLIQUE} = \{ \langle G, k \rangle | \text{G is an undirected graph with a k-clique} \}.$$

Thm) CLIQUE $\in$ NPC

Pf) We show CLIQUE $\in$ NP by giving a verifier $V$.

$V =$ "On input $\langle G, k, c \rangle$,

1) Check that $c = V' \subseteq V$, and reject if not
2) If $|V'| \neq k$, reject
3) For each $u, v \in V'$ with $u \neq v$,
   a) If $(u, v) \notin E$, reject
   4) Accept".

We now show CLIQUE $\in$ NP-hard by showing $3\text{SAT} \leq_m \text{CLIQUE}$. Let $\phi = (a \lor b, v_1) \land \ldots \land (a_k \lor b, v_k)$ be a 3cnf formula with $k$ clauses.

We'll construct a graph $G' = (V', E)$ out of $\phi$.  

The vertices of $G_i$ are placed into $K$ groups of $3$, $6_1, \ldots, 6_K$. Each $6_i$ corresponds to the $i^{th}$ clause of $\phi_i$ and each vertex of $6_i$ corresponds to its literal.

We now connect every distinct pair of vertices in $V$ except that with the same group or with contradictory values.

$$\phi(x, x_2, x_3) = (x, \overline{x_2}, \overline{x_3}) \land (x, \overline{x_2}, x_3)$$

Any $2$-clique (i.e., a connected pair of vertices) will satisfy $\phi$.

We claim $f(\phi) = (G_k, k)$ is a reduction of 3SAT to CLIQUE.

Suppose $\phi$ has a satisfying assignment. To build a $K$-clique, we must pick one vertex from each $6_i$ since the $6_i$'s have no internal edges and there are precisely $k$ of them. In fact, from $6_i$, we pick whichever of $a_i$, $b_i$, or $c_i$ is true in the satisfying assignment of $\phi$ (if more than one is, pick one at random). None of these choices can correspond to a logical contradiction, so by construction, there is an edge between each of them. As such, we have a $K$-clique.
In the reverse direction, suppose we have a $K$-clique. Again, there are $k$ $t_i$'s with no internal edge, so we must have one vertex from each. By construction, these correspond to non-conflicting assignments to $\phi$'s variables. If not every variable has been assigned, just make the remainders all true. Thus $\phi$ is satisfiable.

Given an undirected graph $G = (V, E)$, an independent set is a $V' \subseteq V$ such that $\forall u, v \in V'$, $(u, v) \notin E$. A $K$-independent-set is an independent set of size $k$.

The decision problem for this is

$$IS = \{ \langle G, k \rangle \mid G \text{ is an undirected graph with a } K\text{-independent-set} \}$$

**Thm** $IS \in$ NPC

**Pr** We'll prove $IS \leq P_{\text{CLIQUE}}$ and $\text{CLIQUE} \leq P_{\text{IS}}$.

In either case, the reduction is the same.

$$f(\langle G = (V, E), k \rangle) = \langle (V, E), k \rangle$$

Notice that $f(f(\langle G, k \rangle)) = \langle G, k \rangle$, so it suffices to show that $(V, E)$ contains a $K$-clique iff $(V, E)$ contains a $K$-IS.

Suppose $G$ contains a $K$-clique $V'$. Then $\forall u, v \in V'$, $(u, v) \in E$ we have $\exists v \in V \iff \forall u, v \in V' \ (u, v) \in E \iff (V, E)$ contains a $K$-IS $V'$.

If we wanted to show the reverse reduction, then we do the following.

Suppose $G$ contains a $K$-IS $V'$, then $\forall u, v \in V' \ (u, v) \in E \iff \exists v \in V \iff (V, E)$ contains a $K$-clique $V'$. [\qed]
We have 3SAT and a couple graph problems. Let's pick up a numerical problem next.

The subset sum problem is as follows. Given a finite collection of numbers \( S \) and a target value \( t \), determine if there is an \( S' \subseteq S \) such that \( \sum_{x \in S'} x = t \).

The language for this problem is \( \text{SUBSET-SUM} = \{ (S, t) | S \text{ is a finite collection of numbers with an } S' \subseteq S \text{ such that } \sum_{x \in S'} x = t \} \).

**Thm.** \( \text{SUBSET-SUM} \in \text{NPC} \)

We first give a verifier \( V \) for \( \text{SUBSET-SUM} \).

\[
V = \text{"On input } (S, t, c), \\
1) \text{ Check that } c \leq S \\
2) \text{ Check that } \sum_{x \in c} x = t \\
3) \text{ If both tests pass, accept; reject otherwise"}
\]

Obviously, \( V \) runs in polytime and verifies \( \text{SUBSET-SUM} \).

We now show \( \text{3SAT} \leq^P \text{SUBSET-SUM} \).

Let \( \varphi \) be a 3CNF formula with variables \( x_1, \ldots, x_n \) and clauses \( C_1, \ldots, C_m \). We map \( x_i \) to a number \( y_i \) and \( \bar{x}_i \) to a number \( z_i \); we'll pick \( y_i, z_i, \) and \( t \) such that we can only pick one of \( y_i \) and \( z_i \).

Here's how we do that. The \( i \)th digit of \( y_i \) and \( z_i \) are both 1, as is the \( i \)th digit of \( t \). Every other number we pick will have the \( i \)th digit be 0 so that the only possible way to get 1 in the \( i \)th digit is to pick exactly one of \( y_i \) and \( z_i \).
We also want digit $\ell + j$ of $y_i$ to be 1 if $x_i$ satisfies clause \( c_j \). Similarly, digit $\ell + j$ of $z_i$ is 1 if $\overline{x_i}$ satisfies clause \( c_j \). Both are 0 otherwise. In total, both $y_i$ and $z_i$ should have $\ell + k$ digits. Further, digit $\ell + j$ of $t$ should be 3 for each $j$. This is because a clause can have up to 3 1's from $y_i$, $\overline{y_i}$, or $z_i$. We also add a $y_i$ and $h_i$ that have digit $\ell + i$ be 1 and every other digit be 0. This allows us to fill in up to 2 unsatisfied literals of a clause. We give a full example of this construction.

\[
\phi(x_1, x_2, x_3, x_4) = (x_1 \lor x_2 \lor x_3) \land (\overline{x_1} \lor x_4 \lor x_2) \land (x_2 \lor \overline{x_3} \lor x_4)
\]

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\overline{y_1}$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$\overline{x_1}$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$\overline{x_2}$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$Y_1$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$\overline{Y_1}$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$Y_2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$\overline{Y_2}$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$Y_4$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\overline{Y_4}$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$g_1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$h_1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$g_2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$h_2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$g_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$h_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Pick $x_1, x_3, x_4$ (and $x_2$) or $y_1, y_2, y_3, y_4, g_1, g_2, g_3$.

\[
\begin{align*}
1000 & 100 & y_1 \\
0100 & 101 & y_2 \\
0010 & 110 & y_3 \\
0001 & 011 & y_4 \\
\end{align*}
\]

\[
\begin{align*}
1111 & 13 & 22 \\
0000 & 010 & g_2 \\
0000 & 001 & g_3 \\
\end{align*}
\]

\[
1111 \quad 3 \quad 3 \quad 3 = 6
\]
We'll formally argue this reduction works now. Suppose $\phi$ is satisfiable. Then there is some assignment of $x_1, \ldots, xQ$ that satisfies $\phi$, then selecting the corresponding $y$'s and $z$'s give us that the first $L$ digits are all 1 and the last $K$ digits are all at least 1 since each clause is satisfied, but no more than 3 by construction. We can fill in any of these last $K$ digits that are not 3 with $g$'s and $h$'s.

Now suppose there is a selection of numbers that adds up to $T$. By construction, we must select exactly 1 of each $y_i$ or $z_i$, and for each $j$, we must have at least one $i$ for which a selection of $y_i$ or $z_i$ has its $l+j$th digit as 1. This corresponds to a satisfying assignment, so $\phi$ is satisfiable.

Lastly, the reduction must be done in polytime. The table has size $2(k+L)^2$, and each entry can be determined in linear time at worst, so the runtime is $O(n^3)$.

We now consider the Hamiltonian path problem. A Hamiltonian path in a directed graph $G=(V,E)$ is a path that visits each vertex exactly once. We define the relevant language as

$$\text{HAM-PATH} = \{ (G,s,t) \mid G \text{ is an undirected graph with a Hamiltonian path from } s \text{ to } t \}$$

We will show HAM-PATH is NPC via a reduction from 3SAT.
We first give a polynomial verifier for HAM-PATH to show it's in NP.

\( V = \text{"On input } \langle \langle G,s,t \rangle, c \rangle, \text{"} \)

1) Check that \( c \) is a path \( p \)
2) Check that \( p \) starts with \( s \) and ends at \( t \)
3) Check that \( p \) visits each \( u \in V \) exactly once
4) If any check fails, reject
5) Accept"

We now reduce 3SAT \( \leq_p \) HAM-PATH to get that HAM-PATH is in NP-HARD (and thus NPC).

We'll create a gadget that mimics a variable assignment and attach it to clause nodes, indicating the clause is satisfied.

Bienstock is in the directionality.

Given a boolean formula of \( n \) variables,

\[ \phi = (a_1, v, b_1, v, c_1) \land \ldots \land (a_k, v, b_k, v, c_k), \]

we make a vertex for each \( c_i \); and a gadget for each \( x_i \);

\[ c_i = (x_i, v, ?), \text{ add } x_i \text{ to } V \]

\[ \rightarrow \text{ Assign } x_i, T \]
\[ \leftarrow \text{ Assign } x_i, F \]

\[ \rightarrow \text{ Satisfies } c_i, \text{ which contains } x_i, \text{ and we can include } c_i, \text{ the vertex on a Hamiltonian path but not } \]

\[ \rightarrow \text{ not } \]
The entire construction looks like this

\[ C_i = a_i \lor b_i \lor c_i \]

Then if \( C_i \) is satisfied, then at least one of \( a_i, b_i, \) and \( c_i \) must be a true literal. Then in the corresponding \( x_j \) for that literal we can include the middle vertices exactly once by going exclusively left or exclusively right. Right means \( x_j = \text{True} \) and left means \( x_j = \text{False} \) \((\bar{x}_j = \text{true})\). The \( C \) vertex can be detoured to without revisiting a vertex only when the variable is assigned correctly.

\[ x_j \text{ must be true} \]
\[ x_j \text{ must be false} \]
\[ x_j \text{ can be true or false} \]
\[ C_i = x_j \lor \bar{x}_j \lor ? \]
In the opposite direction, if there's a Hamiltonian path, it must deter to each $C_i$. Where it does so corresponds to a variable assignment, and such a path can only go one direction through that variable, so we can't generate a contradictory assignment. Both of these arguments generalize, so the construction works as a (poly time) reduction.

The **undirected Hamiltonian Path problem** is the same except now you have an undirected graph.

$$\text{UHAM-PATH} = \{ (G, s, t) \mid G \text{ is a graph with a Hamiltonian path from } s \text{ to } t \}.$$ 

**Cor.** UHAM-PATH ∈ NPC  

**PF.** We can transform an undirected graph into a directed graph $G' = (V, E')$ by defining $E' = \{ (u, v) \in V \times V \mid \{u, v\} \in E \}$. 

Suppose $p = v_1 \ldots v_n$ is a Hamiltonian path in $G$. Then $v_i$, $(v_i, v_{i+1}) \in E'$, hence $v_i$, $(v_i, v_{i+1}) \in E'$. Thus $p$ is a Hamiltonian path in $G$. 

Suppose $p = v_1 \ldots v_n$ is a Hamiltonian path in $G$. Then $v_i$, $(v_i, v_{i+1}) \in E'$, hence $v_i$, $(v_i, v_{i+1}) \in E$. Thus $p$ is a Hamiltonian path in $G$.

This shows UHAM-PATH ∈ HAM-PATH, so UHAM-PATH ∈ NPC.

To get the reverse reduction to complete the proof, we need to simulate directionality. Except for $s$ and $t$, we transform each $u \in V$ in $G = (V, E)$.
That is we create 3 vertices $U_{in}$, $U_{mid}$, and $U_{out}$ for each $u \in V \setminus S_t \cup S_s$. There is an edge from $U_{out}$ ($v \in V$) to $U_{in}$ if $(v,u) \in E$, an edge between $U_{in}$ and $U_{mid}$, the same for $U_{mid}$ and $U_{out}$, and an edge from $U_{out}$ to $U_{in}$ ($v \in V$) if $(v,u) \in E$. $s$ only gets $s_{out}$ and it only gets $t_{in}$.

To get a $\text{UHAM-PATH}$, you can only go through the $n$ vertices one way, and since ins are only connected to outs (and vice versa), you can only go one direction.

Example:

![Diagram](image)

Notice the edge $(t,A)$ is lost but that it's useless anyway. The same is true of edge $(t,s)$.

Let this construction be $S((g,s,t)) = (v',E'), s_{out}, t_{in}$.

Suppose $G$ has an $s-t$ Ham-path $u_1 ... u_k t$. Then clearly

$s_{out} u_1 ... u_{i-1} mid u_i ... u_{k-1} u_{k} u_{k+1} ... s_{in}$ is a Ham-path in $G'$.

Now suppose we have an $s_{out}-t_{in}$ Hampath in $G'$. We observed earlier that an out must go to an in which must then go to the corresponding out (via the mid) between since otherwise the mid is lost. This repeats from $s_{out}$ to $t_{in}$, which corresponds to an $s-t$ Ham-path in $G$. 

\[ \]
A Hamiltonian cycle is a cycle that visits each vertex exactly once, but not a path.

A Hamiltonian cycle can be similarly defined.

The modified version can be similarly defined.

The language associated is:

**HAM-CYCLE** = \( \{ G \mid G \text{ is a directed graph containing a Hamiltonian cycle} \} \).

A Hamiltonian cycle is a Hamilton path plus an edge, so obviously its in PSPACE.

We can reduce **HAM-PATH** to **HAM-CYCLE** easily.
If there is an $s-t$ Hamilton path in $G$, specified by edges $e_1, \ldots, e_k$, then clearly $e_1, \ldots, e_k (t, u) (u, s)$ is a Hamilton cycle in $G'$.

Now suppose $e_1, \ldots, e_{k+2}$ is a Hamilton cycle in $G$. Then it must be the case that $\exists s$ such that $e_s = (t, u)$ and $e_{s+1} = (u, s)$. WLOG, assume $i = k+1$. Then $e_1, \ldots, e_k$ is an $s-t$ Hamilton path in $G$. □

Next on our list of problems is vertex cover. This is a minimization problem that we will reduce the maximization problem Clique to. Beyond that it will introduce the notion of cover problems.

Given a graph $G = (V, E)$, a vertex cover of $G$ is a $V' \subseteq V$ such that for every edge $(a, b) \in E$, $a \in V'$ or $b \in V'$.

Example:

\begin{enumerate}
\item $\{A, B, C\}$ is a vertex cover of this graph.
\item It is also minimal.
\end{enumerate}

The corresponding language is:

\[ VC = \{G, k \mid G \text{ is a graph with a vertex cover of size } k \} \]

**Thm** $VC \in \text{NP-Complete}$.

**Pf** We will do the usual thing and show $VC \in \text{NP}$ by providing a poly time verifier. Afterwards, we will show $VC \in \text{NP-Hard}$ by showing $\text{CLIQUE} \leq_p VC$. 

\[ VC \in \text{NP} \]
On input \( \langle \langle G, k \rangle, c \rangle \),
1) Check that \( c = V' \subseteq V \).
2) Check that \( |V'| = k \).
3) Check that \( \forall (a,b) \in E, a \in V' \land b \in V' \).
4) If any check fails, reject.
5) Accept.

The reduction from \( \text{CLIQUE} \leq_k \text{VC} \) is fairly simple.
Let \( G = (V,E) \) be a graph. We construct the edge set complement
\( \overline{E} = \{ \{ u,v \} \mid u \notin v \} \) as we did with the reduction to \( \text{IS} \).
We then claim that \( f(\langle \langle V', E \rangle, k \rangle) = \langle \langle V', E \rangle, |V| - k \rangle \) is a reduction
(and is obviously poly time). The intuition is that \( E \rightarrow \overline{E} \) turns cliques into independent sets, and we need vertices from outside the
\( \text{ISs} \).

\[ \begin{align*}
\text{3 Cliques} & \rightarrow 3 \text{ IS} \setminus \{A,B,D\} \quad 2 \text{ VC} \setminus \{C,E\}.
\end{align*} \]

Suppose \( G = (V,E) \) has a \( k \)-clique \( V' \). Then we know \( V' \)
is a \( k \)-\( \text{IS} \) in \( \overline{G} = (V, \overline{E}) \). By definition, \( \forall u,v \in V' \), \((u,v) \notin \overline{E} \).
In other words, if \((u,v) \in E \), then either \( u \notin V' \) or \( v \notin V' \).
Let \( V'' = V \setminus V' \). Then this means \((u,v) \notin E \Rightarrow u \in V'' \) or \( v \in V'' \).
But this is the definition of a vertex cover, so \( V'' \) is a \((|V| - k)\)-\( \text{VC} \).

Now suppose \( G = (V,E) \) and \( \overline{G} = (V, \overline{E}) \) and \( \overline{G} \) has a \((|V| - k)\)-\( \text{VC} V' \subseteq V \). Then by definition, \( \forall u,v \in V', (u,v) \notin \overline{E} \). But then
if \( V'' = V \setminus V' \), \( V'' \) is a \((|V| - (|V| - k)) = k - \text{IS} \), which we know
is a $k$-clique in $G$.

Thus we have CLIQUE $\leq_p$ VC, so VC $\in$ NP-HARD.

Since VC $\in$ NP as well, we have VC $\in$ NP-COMPLETE.