CS 467/567: NP-complete problems

Stefan D. Bruda

Winter 2023

- Abstract problem: relation Q over the set I of problem instances and the set S of problem solutions: $Q \subseteq I \times S$
  - Complexity theory deals with decision problems or languages ($S = \{0, 1\}$)
    - Technically a language is a set of strings
    - A problem $Q \subseteq I \times \{0, 1\}$ ca be rewritten as the language (set) $L(Q) = \{w \in I : (w, 1) \in Q\}$
  - Many abstract problems are optimization problems instead; however, we can usually restate an optimization problem as a decision problem which require the same amount of resources to solve

Problems everywhere

- Abstract problem: relation Q over the set I of problem instances and the set S of problem solutions: $Q \subseteq I \times S$
  - Complexity theory deals with decision problems or languages ($S = \{0, 1\}$)
    - Technically a language is a set of strings
    - A problem $Q \subseteq I \times \{0, 1\}$ ca be rewritten as the language (set) $L(Q) = \{w \in I : (w, 1) \in Q\}$
  - Many abstract problems are optimization problems instead; however, we can usually restate an optimization problem as a decision problem which require the same amount of resources to solve
- Concrete problem: an abstract decision problem with $I = \{0, 1\}^*$
  - Abstract problem mapped on concrete problem using an encoding $e : I \rightarrow \{0, 1\}^*$
  - $Q \subseteq I \times \{0, 1\}$ mapped to the concrete problem $e(Q) \subseteq e(I) \times \{0, 1\}$
  - Encodings will not affect the performance of an algorithm as long as they are polynomially related
- An algorithm solves a concrete problem in time $O(T(n))$ whenever the algorithm produces in $O(T(n))$ time a solution for any problem instance i with $|i| = n$

LANGUAGES? PROBLEMS?

- Complexity theory analyzes problems from the perspective of how many resources (e.g., time, storage) are necessary to solve them
  - Given some abstract problem that requires certain resource (time) bounds to solve, it is generally easy to find a language that requires the same resource bounds to decide
  - Sometime (but not always) finding an algorithm for deciding the language immediately implies an algorithm for solving the problem

Languages? Problems?

- Complexity theory analyzes problems from the perspective of how many resources (e.g., time, storage) are necessary to solve them
  - Given some abstract problem that requires certain resource (time) bounds to solve, it is generally easy to find a language that requires the same resource bounds to decide
  - Sometime (but not always) finding an algorithm for deciding the language immediately implies an algorithm for solving the problem
- Traveling salesman (TSP): Given $n \geqslant 2$, a matrix $(d_{ij})_{1 \leqslant i,j \leqslant n}$ with $d_{ij} > 0$ and $d_{ii} = 0$, find a permutation π of $\{1, 2, \ldots, n\}$ such that $c(\pi)$, the cost of π is minimal, where $c(\pi) = d_{\pi_1 \pi_2} + d_{\pi_2 \pi_3} + \cdots + d_{\pi_{n-1} \pi_n} + d_{\pi_n \pi_1}$
  - TSP the language (take 1): $\{((d_{ij})_{1 \leqslant i,j \leqslant n}, B) : n \geqslant 2, B \geqslant 0$, there exists a permutation π such that $c(\pi) \leqslant B\}$
  - TSP the language (take 2), or the Hamiltonian graphs: Exactly all the graphs that have a (Hamiltonian) cycle that goes through all the vertices exactly once
  - Note in passing: A cycle that uses all the edges exactly once is Eulerian; a graph G is Eulerian iff
    1. There is a path between any two vertices that are not isolated, and
    2. Every vertex has an in-degree equal to its out-degree

- Clique: Given an undirected graph $G = (V, E)$, find the maximal set $C \subseteq V$ such that $\forall v_i, v_j \in C : (v_i, v_j) \in E$ (C is a clique of G)
  - Clique, the language: $\{(G = (V, E), K) : K \geqslant 2$: there exists a clique C of V such that $|C| \geqslant K\}$

- Clique: Given an undirected graph $G = (V, E)$, find the maximal set $C \subseteq V$ such that $\forall\, v_i, v_j \in C : (v_i, v_j) \in E$ (C is a clique of G)
  - Clique, the language: $\{(G = (V, E), K) : K \geqslant 2$: there exists a clique C of V such that $|C| \geqslant K\}$
- SAT: Fix a set of variables $X = \{x_1, x_2, \ldots, x_n\}$ and let $\overline{X} = \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$
  - An element of $X \cup \overline{X}$ is called a literal
  - A formula (or set/conjunction of clauses) is $\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_m$ where $\alpha_i = x_{a_1} \vee x_{a_2} \vee \cdots \vee x_{a_k}$, $1 \leqslant i \leqslant m$, and $x_{a_i} \in X \cup \overline{X}$
  - An interpretation (or truth assignment) is a function $I : X \to \{\top, \bot\}$
  - A formula F is satisfiable iff there exists an interpretation under which F evaluates to $\top$.
  - SAT $= \{F : F$ is satisfiable $\}$
- 2-SAT, 3-SAT are variants of SAT (with the number of literals in every clause restricted to a maximum of 2 and 3, respectively)

- Clique: Given an undirected graph $G = (V, E)$, find the maximal set $C \subseteq V$ such that $\forall\, v_i, v_j \in C : (v_i, v_j) \in E$ (C is a clique of G)
  - Clique, the language: $\{(G = (V, E), K) : K \geqslant 2$: there exists a clique C of V such that $|C| \geqslant K\}$
- SAT: Fix a set of variables $X = \{x_1, x_2, \ldots, x_n\}$ and let $\overline{X} = \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$
  - An element of $X \cup \overline{X}$ is called a literal
  - A formula (or set/conjunction of clauses) is $\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_m$ where $\alpha_i = x_{a_1} \vee x_{a_2} \vee \cdots \vee x_{a_k}$, $1 \leqslant i \leqslant m$, and $x_{a_i} \in X \cup \overline{X}$
  - An interpretation (or truth assignment) is a function $I : X \to \{\top, \bot\}$
  - A formula F is satisfiable iff there exists an interpretation under which F evaluates to $\top$.
  - SAT $= \{F : F$ is satisfiable $\}$
- 2-SAT, 3-SAT are variants of SAT (with the number of literals in every clause restricted to a maximum of 2 and 3, respectively)
- Note in passing: Sometimes SAT (2-SAT, 3-SAT) is called CNF (2-CNF, 3-CNF) because the input formulae are written in conjunctive normal form

- Complexity class $\mathcal{P}$: the class of all the concrete problems that are solvable in polynomial time
  - Meaning that for any problem in $\mathcal{P}$ there exists an algorithm that solves it in $O(n^k)$ time for some constant $k \geqslant 0$

THE COMPLEXITY CLASS $\mathcal{P}$

- Complexity class $\mathcal{P}$: the class of all the concrete problems that are solvable in polynomial time
  - Meaning that for any problem in $\mathcal{P}$ there exists an algorithm that solves it in $O(n^k)$ time for some constant $k \geqslant 0$
- For some $f : \mathbb{N} \to \mathbb{N}$, a Turing machine $M = (K, \Sigma, \Delta, s, \{h\})$ is f-time bounded iff for any $w \in \Sigma^*$: there is no configuration C such that $(s, \#w\underline{\#}) \vdash_M^{f(|w|)+1} C$
  - M is polynomially (time) bounded iff M is p-time bounded for some polynomial $p = O(n^k)$
  - Problem p is polynomially solvable iff there exists a deterministic polynomially bounded Turing machine that solves $p \Rightarrow$ complexity class $\mathcal{P}$
- $\mathcal{P}$ (as well as all the other complexity classes) are defined based on worst-case analysis

- Complexity class $\mathcal{NP}$: the class of exactly all the problems solvable by nondeterministic, polynomially bounded Turing machines

The complexity class $\mathcal{NP}$

- Complexity class $\mathcal{NP}$: the class of exactly all the problems solvable by nondeterministic, polynomially bounded Turing machines
- Verification algorithm: An algorithm A with two inputs: an ordinary problem instance x and a certificate y
  - A verifies the input x if there exists a certificate y such that $A(x, y) = 1$
  - The language verified by A is $L = \{x \in \{0,1\}^* : \exists\, y \in \{0,1\}^* : A(x,y) = 1\}$
  - A verifies L if for any string $x \in L$, there exists a certificate y that A can use to prove that $x \in L$; for any string $x \notin L$ there must be no certificate proving that $x \in L$
- Complexity class $\mathcal{NP}$: the class of all the problems verifiable in deterministic polynomial time
  - $L \in \mathcal{NP}$ iff there exists a polynomial verification algorithm A and a constant c such that $L = \{x \in \{0,1\}^* : \exists\, y \in \{0,1\}^* : |y| = O(|x|^c) \wedge A(x,y) = 1\}$
- Complexity class $\mathcal{EXP}$: exactly all the problems solvable by exponentially-bounded, deterministic algorithms
  - $\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{EXP}$

- A problem Q can be reduced to another problem Q' if any instance of Q can be "easily rephrased" as an instance of Q'
  - If Q reduces to Q' then Q is "not harder to solve" than Q'

- A problem Q can be reduced to another problem Q' if any instance of Q can be "easily rephrased" as an instance of Q'
  - If Q reduces to Q' then Q is "not harder to solve" than Q'
- Polynomial reduction: A language L_1 is polynomial-time reducible to a language L_2 ($L_1 \leqslant_P L_2$) iff there exists a polynomial algorithm F that computes the function $f : \{0,1\}^* \to \{0,1\}^*$ such that

$$\forall x \in \{0,1\}^* : x \in L_1 \text{ iff } f(x) \in L_2$$

  - Polynomial reductions show that a problem is not harder to solve than another within a polynomial-time factor

- A problem Q can be reduced to another problem Q' if any instance of Q can be "easily rephrased" as an instance of Q'
  - If Q reduces to Q' then Q is "not harder to solve" than Q'
- Polynomial reduction: A language L_1 is polynomial-time reducible to a language L_2 ($L_1 \leqslant_P L_2$) iff there exists a polynomial algorithm F that computes the function $f : \{0,1\}^* \to \{0,1\}^*$ such that
$$\forall\, x \in \{0,1\}^* : x \in L_1 \text{ iff } f(x) \in L_2$$
  - Polynomial reductions show that a problem is not harder to solve than another within a polynomial-time factor

Lemma

$L_1 \leqslant_P L_2 \wedge L_2 \in P \Rightarrow L_1 \in P$

- A problem Q can be reduced to another problem Q' if any instance of Q can be "easily rephrased" as an instance of Q'
  - If Q reduces to Q' then Q is "not harder to solve" than Q'
- Polynomial reduction: A language L_1 is polynomial-time reducible to a language L_2 ($L_1 \leqslant_P L_2$) iff there exists a polynomial algorithm F that computes the function $f : \{0,1\}^* \to \{0,1\}^*$ such that

$$\forall x \in \{0,1\}^* : x \in L_1 \text{ iff } f(x) \in L_2$$

  - Polynomial reductions show that a problem is not harder to solve than another within a polynomial-time factor

Lemma

$L_1 \leqslant_P L_2 \wedge L_2 \in P \Rightarrow L_1 \in P$

- A problem L is NP-hard iff $\forall L' \in \mathcal{NP} : L' \leqslant_P L$
- A problem L is NP-complete ($L \in \mathcal{NPC}$) iff L is NP-hard and $L \in \mathcal{NP}$

Theorem

Let L be some NP-complete problem; then $\mathcal{P} = \mathcal{NP}$ iff $L \in \mathcal{P}$

- Are there NP-complete problems at all?
  - SAT $\in \mathcal{NPC}$ (Stephen Cook, 1971)
- The first is the hard one: need to show that every problem in $\mathcal{NP}$ reduces to our problem
- Then in order to find other NP-complete problems all we need to do is to find problems such that some problem already known to be NP-complete reduces to them
  - This works because polynomial reductions are closed under composition = are transitive
- Then it is apparently easy to use the theorem stated earlier:

  Let L be some NP-complete problem; then $\mathcal{P} = \mathcal{NP}$ iff $L \in \mathcal{P}$

- Tiling system: $\mathcal{D} = (D, d_0, H, V, s)$
  - D is a finite set of tiles
  - $d_0 \in D$ is the initial corner tile
  - $H, V \in D \times D$ are the horizontal and vertical tiling restrictions
  - $s > 0$ is a constant
- Tiling: $f : \mathbb{N}_s \times \mathbb{N}_s \to D$ such that
  - $f(0,0) = d_0$
  - $\forall\, 0 \leqslant m < s, 0 \leqslant n < s - 1 : (f(m,n), f(m,n+1)) \in V$
  - $\forall\, 0 \leqslant m < s - 1, 0 \leqslant n < s : (f(m,n), f(m+1,n)) \in H$
- The bounded tiling problem:
  - Given a tiling system $\mathcal{D}$, a positive integer s and an initial tiling $f_0 : \mathbb{N}_s \to D$
  - Find whether there exists a tiling function f that extends f_0

- Bounded tiling is in $\mathcal{NP}$ (why?)

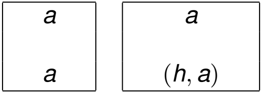
- Need to find reductions from all problems in $\mathcal{NP}$ to bounded tiling

BOUNDED TILING IS NP-COMPLETE

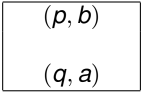
- Need to find reductions from all problems in $\mathcal{NP}$ to bounded tiling
  - The only thing in common to all the $\mathcal{NP}$ problems is that each of them is decided by a nondeterministic, polynomially bounded Turing machine
  - We therefore find a reduction from an arbitrary such a machine to bounded tiling

- Need to find reductions from all problems in $\mathcal{NP}$ to bounded tiling
  - The only thing in common to all the $\mathcal{NP}$ problems is that each of them is decided by a nondeterministic, polynomially bounded Turing machine
  - We therefore find a reduction from an arbitrary such a machine to bounded tiling
- We find a tiling system such that each row in the tiling corresponds to one configuration of the given Turing machine

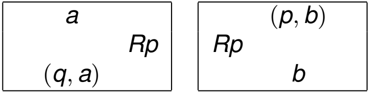
$\forall\, a \in \Sigma$:



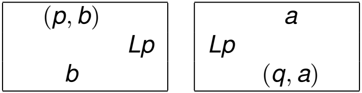
$\forall\, (q, a, p, b) \in \Delta \wedge b \in \Sigma$:



$\forall\, (q, a, p, R) \in \Delta \wedge b \in \Sigma$:



$\forall\, (q, a, p, L) \in \Delta \wedge b \in \Sigma$:



Initial tiling:

$\#$	w_1	w_2	$\ldots$	w_n	$(s, \#)$

SAT IS NP-COMPLETE

1. SAT $\in \mathcal{NP}$
   - Nondeterministically guess an interpretation then check that the interpretation satisfies the formula
   - Both of these take linear time
2. SAT is NP-hard
   - Reduction of bounded tiling to SAT
   - Variables: x_{nmd} standing for "tile d is at position (n, m) in the tiling"
   - Construct clauses such that $x_{nmd} = \top$ iff $f(m, n) = d$
   - First specify that we have a function:
     - Each position has at least one tile: $\forall\, 0 \leqslant m, n \leqslant s : x_{mnd_1} \vee x_{mnd_2} \vee \cdots$
     - No more than one tile in a given position: $\forall\, 0 \leqslant m, n \leqslant s, d \neq d' : \overline{x_{mnd}} \vee \overline{x_{mnd'}}$
   - Then specify the restrictions H and V:
     - $(d, d') \in D^2 \backslash H \Rightarrow \overline{x_{mnd}} \vee \overline{x_{m+1nd'}}$ $\qquad (d, d') \in D^2 \backslash V \Rightarrow \overline{x_{mnd}} \vee \overline{x_{mn+1d'}}$

- 3-SAT is also NP-complete

CLIQUE

- 3-SAT is NP-complete

Clique

- 3-SAT is NP-complete
  - Hint: any clause $x_1 \vee x_2 \vee \cdots x_n$ is logically equivalent with $(x_1 \vee x_2 \vee x_2') \wedge (\overline{x_2'} \vee x_3 \vee x_3') \wedge (\overline{x_3'} \vee x_4 \vee x_4') \wedge \cdots \wedge (\overline{x_{n-2}'} \vee x_{n-1} \vee x_n)$
- Clique $= \{(G = (V, E), k) : k \geqslant 2$: there exists a clique C of V with $|C| = k\}$

  Membership in $\mathcal{NP}$ and 3-SAT $\leqslant_P$ Clique $\Rightarrow$ Clique $\in \mathcal{NPC}$

CLIQUE

- 3-SAT is NP-complete
  - Hint: any clause $x_1 \vee x_2 \vee \cdots x_n$ is logically equivalent with $(x_1 \vee x_2 \vee x_2') \wedge (\overline{x_2'} \vee x_3 \vee x_3') \wedge (\overline{x_3'} \vee x_4 \vee x_4') \wedge \cdots \wedge (\overline{x_{n-2}'} \vee x_{n-1} \vee x_n)$
- CLIQUE $= \{(G = (V, E), k) : k \geqslant 2 :$ there exists a clique C of V with $|C| = k\}$

  Membership in $\mathcal{NP}$ and 3-SAT $\leqslant_P$ CLIQUE $\Rightarrow$ CLIQUE $\in \mathcal{NPC}$
  - Start from $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$, construct $G = (V, E)$
  - Start with $V = \varnothing$ and $E = \varnothing$
  - For each clause $C_r = l_1^r \vee l_2^r \vee l_3^r$ add vertices v_1^r, v_2^r, and v_3^r to V
  - Add (v_i^r, v_j^s) to E whenever $r \neq s$ and l_i^r is not the negation of l_j^s (l_i^r is and l_j^s are consistent)

CLIQUE

- 3-SAT is NP-complete
  - Hint: any clause $x_1 \lor x_2 \lor \cdots x_n$ is logically equivalent with $(x_1 \lor x_2 \lor x'_2) \land (\overline{x'_2} \lor x_3 \lor x'_3) \land (\overline{x'_3} \lor x_4 \lor x'_4) \land \cdots \land (\overline{x'_{n-2}} \lor x_{n-1} \lor x_n)$
- CLIQUE $= \{(G = (V, E), k) : k \geqslant 2$: there exists a clique C of V with $|C| = k\}$

  Membership in $\mathcal{NP}$ and 3-SAT $\leqslant_P$ CLIQUE $\Rightarrow$ CLIQUE $\in \mathcal{NPC}$
  - Start from $\phi = C_1 \land C_2 \land \cdots \land C_k$, construct $G = (V, E)$
  - Start with $V = \emptyset$ and $E = \emptyset$
  - For each clause $C_r = l_1^r \lor l_2^r \lor l_3^r$ add vertices v_1^r, v_2^r, and v_3^r to V
  - Add (v_i^r, v_j^s) to E whenever $r \neq s$ and l_i^r is not the negation of l_j^s (l_i^r is and l_j^s are consistent)
  - Suppose that ϕ is satisfiable; then:
    - The interpretation that makes ϕ true makes at least one literal l_i^r per clause true
    - The vertex v_i^r is connected to all the other vertices v_j^s that make the other clauses true (these are all consistent with each other)
    - So the vertices v_i^r form a clique (of size k) □

- 3-SAT is NP-complete
  - Hint: any clause $x_1 \vee x_2 \vee \cdots x_n$ is logically equivalent with $(x_1 \vee x_2 \vee x_2') \wedge (\overline{x_2'} \vee x_3 \vee x_3') \wedge (\overline{x_3'} \vee x_4 \vee x_4') \wedge \cdots \wedge (\overline{x_{n-2}'} \vee x_{n-1} \vee x_n)$
- CLIQUE $= \{(G = (V, E), k) : k \geqslant 2 :$ there exists a clique C of V with $|C| = k\}$
  Membership in $\mathcal{NP}$ and 3-SAT $\leqslant_P$ CLIQUE $\Rightarrow$ CLIQUE $\in \mathcal{NPC}$
  - Start from $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$, construct $G = (V, E)$
  - Start with $V = \varnothing$ and $E = \varnothing$
  - For each clause $C_r = l_1^r \vee l_2^r \vee l_3^r$ add vertices v_1^r, v_2^r, and v_3^r to V
  - Add (v_i^r, v_j^s) to E whenever $r \neq s$ and l_i^r is not the negation of l_j^s (l_i^r is and l_j^s are consistent)
  - Suppose that ϕ is satisfiable; then:
    - The interpretation that makes ϕ true makes at least one literal l_i^r per clause true
    - The vertex v_i^r is connected to all the other vertices v_j^s that make the other clauses true (these are all consistent with each other)
    - So the vertices v_i^r form a clique (of size k) □
  - Suppose that G has a clique C of size k; then:
    - C contains exactly one vertex per clause
    - Assigning $\top$ to every literal l_i^r for which $v_i^r \in C$ is possible (all are consistent with each other)
    - The assignment makes ϕ true so ϕ is satisfiable □

VERTEX COVER

- A vertex cover of $G = (V, E)$ is a set $V' \subseteq V$ such that $(u, v) \in E \Rightarrow u \in V' \vee v \in V'$
- VERTEX-COVER $= \{(G = (V, E), k) : G$ has a vertex cover of size $k\}$
  Membership in $\mathcal{NP}$ and CLIQUE $\leqslant_P$ VERTEX-COVER $\Rightarrow$ VERTEX-COVER $\in \mathcal{NPC}$

VERTEX COVER

- A vertex cover of $G = (V, E)$ is a set $V' \subseteq V$ such that $(u, v) \in E \Rightarrow u \in V' \vee v \in V'$
- VERTEX-COVER $= \{(G = (V, E), k) : G$ has a vertex cover of size $k\}$
  Membership in $\mathcal{NP}$ and CLIQUE $\leqslant_P$ VERTEX-COVER $\Rightarrow$ VERTEX-COVER $\in \mathcal{NPC}$
  - Start from $(G = (V, E), k) \in$ CLIQUE
  - Compute $\overline{G} = (V, \overline{E})$ where $\overline{E} = (V \times V) \setminus E$ (the complement of G)
  - Then $(G, k) \in$ CLIQUE iff $(\overline{G}, |V| - k) \in$ VERTEX-COVER

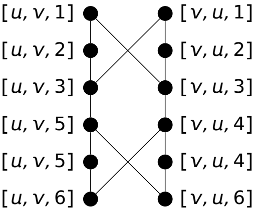
VERTEX COVER

- A vertex cover of $G = (V, E)$ is a set $V' \subseteq V$ such that $(u, v) \in E \Rightarrow u \in V' \vee v \in V'$
- VERTEX-COVER $= \{(G = (V, E), k) : G$ has a vertex cover of size $k\}$
  Membership in $\mathcal{NP}$ and CLIQUE $\leqslant_P$ VERTEX-COVER $\Rightarrow$
  VERTEX-COVER $\in \mathcal{NPC}$
  - Start from $(G = (V, E), k) \in$ CLIQUE
  - Compute $\overline{G} = (V, \overline{E})$ where $\overline{E} = (V \times V) \backslash E$ (the complement of G)
  - Then $(G, k) \in$ CLIQUE iff $(\overline{G}, |V| - k) \in$ VERTEX-COVER
  - Suppose that G has a clique C, $|C| = k$; then:
    - $(u, v) \notin E$ means that u and v cannot be both in C
    - That is, $V \backslash C$ covers every edge $(u, v) \notin E$ that is, every vertex $(u, v) \in \overline{E}$
    - Therefore $V \backslash C$ is a vertex cover for $\overline{G}$ (of size $|V| - k$) □

- A vertex cover of $G = (V, E)$ is a set $V' \subseteq V$ such that $(u, v) \in E \Rightarrow u \in V' \vee v \in V'$
- VERTEX-COVER $= \{(G = (V, E), k) : G \text{ has a vertex cover of size } k\}$
  Membership in $\mathcal{NP}$ and CLIQUE $\leqslant_P$ VERTEX-COVER $\Rightarrow$ VERTEX-COVER $\in \mathcal{NPC}$
  - Start from $(G = (V, E), k) \in$ CLIQUE
  - Compute $\overline{G} = (V, \overline{E})$ where $\overline{E} = (V \times V) \setminus E$ (the complement of G)
  - Then $(G, k) \in$ CLIQUE iff $(\overline{G}, |V| - k) \in$ VERTEX-COVER
  - Suppose that G has a clique C, $|C| = k$; then:
    - $(u, v) \notin E$ means that u and v cannot be both in C
    - That is, $V \setminus C$ covers every edge $(u, v) \notin E$ that is, every vertex $(u, v) \in \overline{E}$
    - Therefore $V \setminus C$ is a vertex cover for $\overline{G}$ (of size $|V| - k$) □
  - Suppose that $\overline{G}$ has a vertex cover V' with $|V'| = |V| - k$; then:
    - $(u, v) \in \overline{E} \Rightarrow u \in V' \vee v \in V'$
    - Contrapositive: $u \notin V' \wedge v \notin V' \Rightarrow (u, v) \notin \overline{E}$
    - That is, $u \in V \setminus V' \wedge v \in V \setminus V' \Rightarrow (u, v) \in E$
    - So $V \setminus V'$ is a clique of G (or size k) □

Hamiltonian cycle

- $\textsc{Hamiltonian-cycle} = \{G = (V, E) : G \text{ is Hamiltonian}\}$
  Membership in $\mathcal{NP}$ and $\textsc{Vertex-cover} \leqslant_P \textsc{Hamiltonian-cycle} \Rightarrow$ $\textsc{Hamiltonian-cycle} \in \mathcal{NPC}$
  - Given $(G = (V, E), k)$ construct $G' = (V', E')$
  - For each $(u, v) \in E$ we use the widget W_{uv} to the right.
  - A widget can only connect to the rest of the graph through $[u, v, 1]$, $[u, v, 6]$, $[v, u, 1]$, and $[v, u, 6]$
  - Thus there are only three ways to traverse a widget as part of a Hamiltonian cycle



  - We also use the selector vertices $s_1, s_2, \ldots, s_k$
  - For each $u \in V$ and all the vertices $u^{(1)}, \ldots, u^{(du)}$ adjacent to u in G we add $([u, u^{(i)}, 6], [u, u^{(i+1)}, 1])$ to G', $1 \leqslant i \leqslant du - 1$
    - These form a "path of widgets" that include all the widgets for the edges incident on u
    - Useful to start such a part from a member of the vertex cover
  - We add the vertices $(s_j, [u, u^{(1)}, 1])$ and $(s_j, [u, u^{du}, 6])$ for all $u \in V$ and $1 \leqslant j \leqslant k$
    - These complete a cycle (combined with the path of widgets) but only for the members of the vertex cover