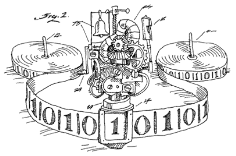
EECS 376: Foundations of Computer Science

Lecture 16 - Introduction to NP-Completeness



Today's Agenda

- Recap
- Prove more NP-completeness
  - 3SAT
  - Clique
  - Vertex-Cover

Recap

- **P** is the set of efficiently decidable decision problems
- **NP** is the set of efficiently verifiable decision problems

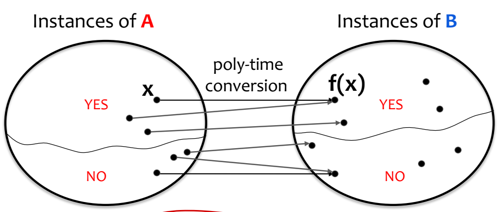
- **Major open problem**: is **P** = **NP**?
  - Is every easy-to-verify problem also easy to solve?

- **Common belief: P $\neq$ NP**
  - Some easy-to-verify problems are not easy to solve.
  - I treat it like a law of physics: Not proven. But full of evidence.

Polynomial-time mapping reduction from **A** to **B** (denoted **A** $\leq_p$ **B**)

Defn: A $\leq_p$ **B** if there is a poly-time-computable function **f** where

x is a yes-instance of **A** $\Leftrightarrow$ **f(x)** is a yes-instance of **B**.



Remark: f need not be injective nor surjective.

NP-Hardness and **NP**-Completeness

A problem **L** is NP-hard if
- for EVERY problem **X** in NP, **X** $\leq_p$ **L**.

A problem **L** is NP-complete if
- **L** $\in$ **NP**
- **L** is **NP**-hard

Exercise

Suppose **A** $\leq_p$ **B.**
1. If **B** $\in$ **P**, then **A** $\in$ **P.**
2. If **A** is NP-hard, then **B** is NP-hard.
3. Suppose L is NP-complete. Then, L $\in$ **P** iff **P** = **NP**
   - Suppose L $\notin$ **P**. As L $\in$ **NP**, then **P** $\neq$ **NP**.
   - Suppose L $\in$ **P**.
     - As L is **NP-hard**, every NP-problem X $\leq_p$ L.
     - So, X is in **P** by (1).
     - So, **NP** $\subseteq$ **P.**

Terminology on Formulas

A **Boolean formula** Φ is made up of:
- "literals": variables and their negations (e.g. x, y, z, ¬x, ¬y, ¬z)
- OR: ∨
- AND: ∧

Example:

$$\Phi 1 = (x \vee y) \wedge (\neg y \vee x \vee \neg z) \wedge (\neg x \vee (y \wedge \neg z))$$

Φ is ***satisfiable*** if
- ∃ a true/false assignment **A** to the variables so that Φ(**A**) = true
- For example, Φ1 is satisfiable.
  - Assign x = F, y = T, z = F

Satisfiability (SAT)

Input: A Boolean formula Φ
Output: Is Φ *satisfiable*?

SAT is NP-complete

Given that SAT is NP-complete,
Today: prove that other problems are NP-complete via **reductions**

3SAT is NP-complete

A version of SAT called **3SAT** is also NP-complete
(proof is in course notes)

A 3SAT instance:

$$\underbrace{(x_1 \vee \overline{x}_2 \vee x_{42})}_{\text{Clause 1}} \wedge \underbrace{(x_2 \vee x_3 \vee \overline{x}_{17})}_{\text{Clause 2}} \wedge \cdots \wedge \underbrace{(\overline{x}_3 \vee x_5 \vee x_{17})}_{\text{Clause m}}$$

- Each clause contains the OR (disjunction) of exactly three literals (a literal is a variable or its negation)
- The clauses are ANDed to form a single boolean expression

this type of formula is called a "3-CNF" ("conjunctive normal form")

Input: A 3-CNF formula Φ
Output: Is the formula Φ satisfiable?

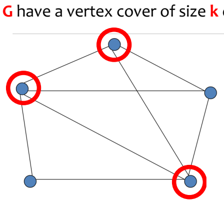
Notation: ¬x and $\overline{x}$ both mean "not x" (you can use either)

Vertex Cover is NP-complete

Vertex Cover (VC)
("Coffee Shop Problem")

Put coffee shops on street corners so that every street has a shop on at least one of its two corners.

- A ***vertex cover*** of a graph is a set **S** of vertices such that
  - for every edge, at least one of its two endpoints is in **S** (i.e. every edge is "covered")
- **Vertex cover problem:**
  - Given a graph **G** and a budget **k**,
  - does **G** have a vertex cover of size **k** or less?



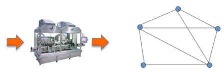
When k = 3 ⇒ Answer: Yes
When k = 4 ⇒ Answer: Yes
When k = 2 ⇒ Answer: No

Showing **NP**-Completeness via reductions

To show that **VC** is **NP**-Complete:

- **VC** is in **NP.** (Why?)
  - Certificate: a vertex set **S** of size at most k
  - Verifier: check that, for all edges (u, v), either u in **S** or v in **S**
- **VC** is NP-hard by showing $\mathbf{3SAT} \le_p \boldsymbol{VC}$

$(x_1 \vee \overline{x}_2 \vee x_4) \wedge$
$(x_2 \vee x_3 \vee \overline{x}_{42})$



, 3

3SAT instance — fast conversion machine — Vertex Cover Instance

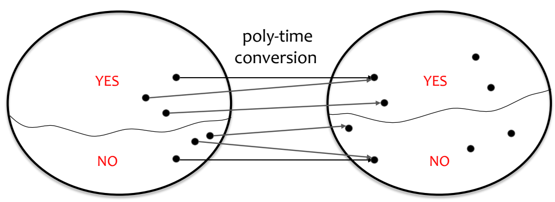
Showing **NP**-Completeness via reductions

To show that **VC** is **NP**-Complete:

- **VC** is in **NP.** (Why?)
  - Certificate: a vertex set **S** of size at most k
  - Verifier: check that, for all edges (u, v), either u in **S** or v in **S**
- **VC** is NP-hard by showing $\mathbf{3SAT} \le_p \boldsymbol{VC}$
  1. Show a mapping f from instances of **3SAT** to instances of **VC**
  2. x is a yes-instance for **3SAT** ⟺ $f(x)$ is a yes-instance of **VC** (both directions!)
  3. $f(x)$ runs in poly($|x|$) time

$\mathbf{3SAT} \le_p \boldsymbol{VC}$

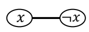
Instances of 3SAT — Instances of VC



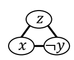
$\mathrm{3SAT} \le_p \mathrm{VC}$

Goal: "*translate*" φ to (G_φ, k_φ) st:
- φ sat ⇒ G_φ has some k_φ-VC
- φ unsat ⇒ G_φ has no k_φ-VC

- **Claim:** $\mathrm{3SAT} \le_p \mathrm{VC}$
- **Proof idea:**
  - Given a 3CNF formula ϕ with n variables, m clauses:
  - Make subgraphs (***"gadgets"***) that represent variables and clauses.
  - Connect the gadgets together in the right way.



Variable gadget for x



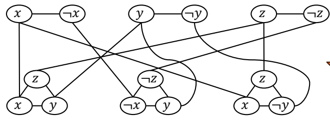
Clause gadget for $(x \vee \neg y \vee z)$

Goal: "<u>translate</u>" φ to (G_φ, k_φ) st:
- φ sat $\Rightarrow G_\varphi$ has some k_φ-VC
- φ unsat $\Rightarrow G_\varphi$ has no k_φ-VC

$3\text{SAT} \le_p \text{VC}$

* **Construction of G_φ:**
  * **add** *variable gadgets* and *clause gadgets (for every variable and clause)*
  * **add** edge $\{u, v\}$ if
    * u is in a <u>variable gadget</u> and
    * v is in a <u>clause gadget and</u>
    * u and v are <u>labeled the same</u>
* **Set k_φ** to $n + 2m$ (n – number of variables, m – number of clauses)
* Concrete example:

$\varphi = (x \vee y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z)$



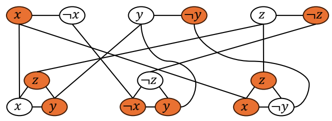
Where is a VC of size n+2m in this graph

Goal: "<u>translate</u>" φ to (G_φ, k_φ) st:
- φ sat $\Rightarrow G_\varphi$ has some k_φ-VC
- φ unsat $\Rightarrow G_\varphi$ has no k_φ-VC

$3\text{SAT} \le_p \text{VC}$

* **Construction of G_φ:**
  * **add** *variable gadgets* and *clause gadgets (for every variable and clause)*
  * **add** edge $\{u, v\}$ if
    * u is in a <u>variable gadget</u> and
    * v is in a <u>clause gadget and</u>
    * u and v are <u>labeled the same</u>
* **Set k_φ** to $n + 2m$ (n – number of variables, m – number of clauses)
* Concrete example:

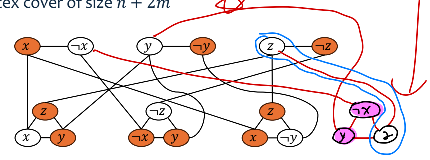
$\varphi = (x \vee y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z)$



如果一个clause里没有一个variable true, 那么在其中任意选 n-1个variable,
⇒ 最后一个variable都会
形成一个不被
cover的edge

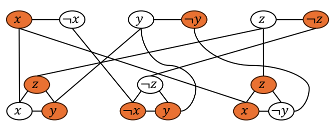
ϕ satisfiable $\Rightarrow (n + 2m)$-VC

* Concrete example:
  * $\varphi = (x \vee y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z)$
  * $A = (1,0,0)$ is a satisfying assignment
* Given a satisfying assignment A:
  * For each variable gadget,
    * pick x into vertex-cover if $A_x = 1$ and $\neg x$ otherwise
  * **Claim:** can cover all other edges by picking 2 vertices per clause gadget.
    * For example, if x =true, pick other two literals.
* Get a vertex cover of size $n + 2m$



$(n + 2m)$-VC $\Rightarrow \phi$ satisfiable

* **Claim:** In a $(n + 2m)$-VC of G_φ,
  * each variable gadget has exactly one vertex in cover.
  * each clause gadget has exactly two vertices in cover.
* For each variable x,
  * If x is in cover $\Rightarrow$ set $A_x = 1$. Else, set $A_x = 0$
* A is a satisfying assignment!
  * For any clause gadget, for example,
    if "$\neg z$" is not picked $\Rightarrow$ "$\neg z$" must be picked in variable gadget.
  * So, each clause is satisfied.



$3\text{SAT} \le_p \text{VC}$

* **Construction of G_φ:**
  * **create** *variable gadgets* and *clause gadgets (for every variable and clause)*
  * **add** edge $\{u, v\}$ if
    * u is in a <u>variable gadget</u> and
    * v is in a <u>clause gadget and</u>
    * u and v are <u>labeled the same</u>
* **Set k_φ** to $n + 2m$ (n – number of variables, m – number of clauses)

Goal: "<u>translate</u>" φ to (G_φ, k_φ) st:
- φ sat $\Rightarrow G_\varphi$ has some k_φ-VC
- φ unsat $\Rightarrow G_\varphi$ has no k_φ-VC

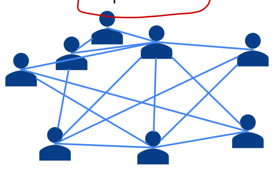
* **Last check:** (G_φ, k_φ) can be constructed in poly($|\varphi|$) = poly(n,m) time

Clique is NP-complete

Clique Problem

(Friendship problem)

* Given a graph, a ***clique*** is a set **S** of vertices so that
  * *every* pair of vertices in **S** has an edge between them.
* **Clique decision problem:**
  * Given a graph **G** and a budget **k**,
  * does **G** have a clique of size **k** or more?



, 4

We will show: 3SAT ≤p CLIQUE

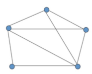
Showing **NP**-Completeness via reductions

To show that **CLIQUE** is **NP**-Complete:

* **CLIQUE** is in **NP.** (Why?)
  * Certificate: a vertex set **S** of size at least k
  * Verifier: check that, for all $u, v \in$ **S**, (u, v) is an edge
* **CLIQUE** is NP-hard by showing **3SAT** $\le_p$ **CLIQUE**

$(x_1 \vee \overline{x}_2 \vee x_4) \wedge$
$(x_2 \vee x_3 \vee \overline{x}_{42})$

fast conversion machine



, 3

3SAT instance

Clique Instance

$3\mathrm{SAT} \leq_p \mathrm{CLIQUE}$

* Need to "*translate*" a 3SAT formula φ into (G_φ, k_φ) such that:
  * φ is satisfiable $\Rightarrow G_\varphi$ has k_φ-clique (clique: "yes")
  * φ is *not* satisfiable $\Rightarrow G_\varphi$ *doesn't* have k_φ-clique (clique: "no")
* Given φ, (G_φ, k_φ) can be constructed in polynomial time

Example

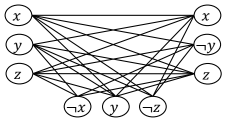
Goal: "*translate*" 3CNF formula φ into (G_φ, k_φ) such that:
* φ satisfiable $\Rightarrow G_\varphi$ has k_φ-clique (clique: "yes")
* φ not satisfiable $\Rightarrow G_\varphi$ *doesn't* have k_φ-clique (clique: "no")

Construction of G_φ:
* For each clause, make a vertex for each literal
* Add an edge between two literals in *different clauses* only if they're "compatible"
  * They refer to different variables (e.g. x and ¬y) **or**
  * They are the same (e.g. z and z)

Set k_φ to m --- number of clauses

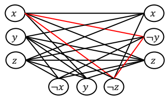
Concrete example:

$$\varphi = (x \lor y \lor z) \land (\neg x \lor y \lor \neg z) \land (x \lor \neg y \lor z)$$



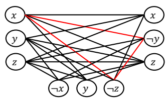
Observations:
1) Any clique can have at most one vertex from a clause ($k_\varphi \leq 3$)
2) A clique can have several x or $\neg x$, but not both

φ is satisfiable $\Rightarrow G_\varphi$ has an m-clique



* Consider any satisfying assignment A of φ
* Since φ is satisfied by A, for $1 \leq i \leq m$,
  * For each C_i (e.g., $x \lor y \lor z$), select a literal ℓ_i that A sets to true (pick any if there are several choices)
* **Claim:** $\{\ell_1, \ell_2, \ldots, \ell_m\}$ is an m-clique in G_φ
  * Consider any two literals ℓ_i and ℓ_j, $i \neq j$
  * If $\ell_i = \ell_j$, then there's an edge between them in G_φ.
  * Otherwise, ℓ_i and ℓ_j must refer to *different variables*! (why?)
  * Hence, they also have an edge between them.

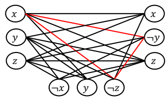
G_φ has an m-clique $\Rightarrow \varphi$ is satisfiable



* Suppose that $\{\ell_1, \ell_2, \ldots, \ell_m\}$ is an m-clique in G_φ
  * **It corresponds to one literal per clause**
* Define an *assignment* A of φ
  * For each literal ℓ_i, A sets ℓ_i to true (if any variables are unset at the end, set them arbitrarily)
* $\{\ell_1, \ell_2, \ldots, \ell_m\}$ is a clique in $G_\varphi \Rightarrow$ *no conflicts* in setting the variables this way
  * For each ℓ_i and ℓ_j, they are compatible
* Since A satisfies each clause of φ, it satisfies φ!

Runtime Analysis

Construction of G_φ:
* For each clause, make a vertex for each literal
* Add an edge between two literals in *different clauses* only if they're "compatible"



* **Claim:** We can build graph G_φ efficiently (poly-time in size of φ)
  * Suppose φ has m clauses. $|\langle\varphi\rangle| = O(m)$
  * There are $3m$ literals in φ
  * The graph G_φ has $3m$ vertices and $O((3m)^2) = O(m^2)$ edges
    * Takes $O(m^2)$ time to build and $|\langle G_\varphi, k_\varphi\rangle| = O(m^2)$
* **Conclusion:** $3\mathrm{SAT} \leq_p \mathrm{CLIQUE}$,
* So, CLIQUE is NP-Complete

Wrap Up

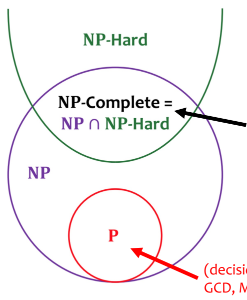
NP-Completeness via reductions

To show that a problem **B** is **NP**-Complete:

* Prove **B** is in **NP.**
  * Write a verifier V for B, show that it is correct and efficient.
* Prove **B** is NP-hard
  * Pick some *known* **NP**-hard problem **A.**
  * Show **A** $\leq_p$**B:**
    1. Show a mapping f from instances of **A** to instances of **B**
    2. x is a yes-instance for **A** $\Leftrightarrow f(x)$ is a yes-instance of **B** (both directions!)
    3. $f(x)$ runs in poly($|x|$) time

Classification of Problems: Efficient vs Inefficient

Assuming $P \neq NP$, we have classified problems into two classes



SAT (Cook-Levin)
3SAT
Vertex Cover, Clique
Next class: Set Cover, Independent Set, and more…

(decision version of)
GCD, Multiplication, LIS, LCS, shortest path…