Lecture 23: NP, NP-Completeness

Today

- NP
- NP-Completeness
- NP-Completeness of Clique

Review of last lecture

1. $P = \text{polynomial time solvable decision problems}$

2. Many problems:
   - FACTORING (Gauss)
   - Divisor (Kilbey)
   - INTEGER PROGRAMMING (von Neumann)
   - CLIQUE (Karp)
   - CNF-SAT (Cook)
3. \( \leq \) : \( \mathcal{P}_1 \leq \mathcal{P}_2 \) if

\[ \exists \text{ polynomial time } f \text{ s.t.} \]

\[ \mathcal{P}_1(x) = 1 \iff \mathcal{P}_2(f(x)) = 1. \]

4. \text{Factoring} \leq \text{Ckt-\text{SAT}}

5. if \( \mathcal{P}_1 \leq \mathcal{P}_2 \) and \( \mathcal{P}_2 \leq \mathcal{P}_1 \), then we can say \( \mathcal{P}_1 = \mathcal{P}_2 \);

we claimed without proof that

\[ \text{1.} \ \mathcal{P} = \text{Clique} = \text{Ckt-\text{SAT}}; \]

Today we’ll see some of this.)
Common feature to IP, factoring, divisor, Clique

if \( \Pi_1(x) = 1 \) then easy to "prove" this

Example:
" \( G \) has clique of size \( \geq k \)"

Proof: Clique \( C \)

" \( X \) has divisor between \( A \) & \( B \)"

Proof: Divisor \( D \)

- Despite these "easy" proofs: no simple algorithm to solve: only Brute-Force

- What makes "proof": Polynomial time alg. to verify.
NP: Class of problems \( \Pi \) given by a "verification procedure" \( V \).

- \( V(x, y) \in \{0, 1\} \)
- \( V \) runs in polynomial time
- \( \Pi(x) = 1 \iff \exists y \text{ s.t. } V(x, y) = 1 \).

\( |y| = |x|^{O(1)} \).

Example

Factoring \( \in \text{NP} \): \( n = x, b \)

\( y = d \)

\[ V(x, b, d) = \begin{cases} 0 & \text{if } d | x \text{ and } 2 \leq d \leq b \text{?} \\ 1 & \text{otherwise} \end{cases} \]
Ckt-SAT $\in$ NP : $x = \text{Ckt} \perp C$

$y = \emptyset$, assignment to $x$

$V_{\text{Ckt-SAT}}(x, y) : \begin{cases} 1 \text{ if } C(x) = 1 \end{cases}$

Clique $\in$ NP : $x = (G, k)$

$y = \text{Clique} \subseteq C$

$V_{\text{Clique}}(x, y) : \begin{cases} 1 \text{ if } |C| \geq k \\ 2 \text{ if } u, v \in C, (u,v) \notin E \end{cases}$
Is $P = NP$?

Ask the question: Is every problem in NP also in P?

Does the existence of $V^\Pi$ imply existence of $D^\Pi$?

- $D^\Pi$ runs in polynomial time
- $D^\Pi(x) = \Pi(x)$

Surely not?

Karp $\neq$ Surely Not

 Levin $=$ Surely Yes

Cook $=$ Maybe Independent
if \( P = NP \)

suffices to show one problem in \( NP \) that is not in \( P \).

(e.g. factoring,

or Clique,

or \( \text{CIRCUIT-SAT} \ldots \))

What if \( P = NP \)

need infinitely many \( D_T \) \( \forall \)

for each \( T \) ?

Turn out not

"\( NP \)-completeness"
Definition: \( \Pi \) is \( \text{NP-complete} \) if for every problem \( \Pi' \in \text{NP} \),

\[
\Pi' \leq \Pi
\]

(i.e. \( \Pi \) is a largest problem in \( \text{NP} \))

To show \( \text{P} = \text{NP} \), suffices show \( \Pi \in \text{P} \) for any one \( \Pi \) that is \( \text{NP-complete} \).

Why? if \( \Pi' \in \text{NP} \) & \( \Pi \in \text{P} \)

then \( \Pi' \leq \Pi \in \text{P} \)

\[
\Rightarrow \Pi' \in \text{P}
\]

\[
\Rightarrow \text{NP} \leq \text{P}
\]
But is the definition useful?
Are there any NP-complete problems?
Turns out... many, including 3 we've met:
INTEGER PROGRAMMING; CLIQUE; CHF-SAT...

---

How does one prove NP-completeness?

\[
\begin{align*}
\text{Need infinitely many reductions?} \quad & \quad \text{One for each } \Pi \in \text{NP?}
\end{align*}
\]

Fortunately not; can come up with uniform reduction based on \( V_{\Pi} \) (verifier for \( \Pi \)).
NP-completeness of Ckt-SAT

More or less by definition...

Need to define "\( V_\Pi (n, y) \) is polynomial"...

we know what a polynomial time algorithm is, but can we be mathematical about it?

... we don't do it here, but it means that given \( V_\Pi \) & \( n \), we can construct a circuit \( \hat{C} = \hat{C}_{V_\Pi, n} \)

\[ \forall x, \quad |\tilde{C}| = n \]

\[ \hat{C}(x, y) = 1 \iff V_\Pi (x, y) = 1 \]

then the map \( x \to C_x = \hat{C}(x, \cdot) \)

reduces \( \Pi \to \text{Ckt-SAT}. \)
NP-Completeness of CLIQUE

Suffices to reduce \( \text{Conj-SAT} \rightarrow \text{CLIQUE} \)

Need a polynomial time algorithm

\[ C \rightarrow f \rightarrow (G, k) \]

\( x_1, \ldots, x_n \) \hspace{1cm} Vertices = ?

\( g_1, \ldots, g_m \) \hspace{1cm} Edges = ?

Idea: Vertices of \( G \) represent possible assignments to \( x_1, \ldots, x_n \)

\& \( g_1, \ldots, g_m \) (plus some additional vertices)

Edges represent local consistency

(Actually non-edges represent local inconsistency)
$\text{Vertex } \ V = V_1 \cup V_2$

$V_1 = \{ x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n \}$

$g_0, g_1, g_2, \ldots, g_m, \overline{g_m}$

$\uparrow$

$\text{Gadgets (Enforce gate rules)}$

$\text{NOT gate :}$

\[ A \xleftarrow{\text{gate/variable}} \neg \rightarrow \xrightarrow{\text{gate output}} B \]

\[ \begin{array}{c}
A \\
\hline
\hline
A
\end{array} \quad \begin{array}{c}
B \\
\hline
\hline
\bar{B}
\end{array} \]

\[ \text{Green edges included} \]

\[ \text{Red edges not included} \]
- A New vertices $\overline{AB}$, $\overline{A\overline{B}}$, $\overline{A\overline{B}}$, $A\overline{B}$.

- Red edges from $\overline{B}$ to $\overline{C}$.

- Red edges from $A$ to $\overline{AB}$, $\overline{A\overline{B}}$ etc.
Idea: if $\overline{A}, B$ are true
then $\overline{A}, B, \overline{AB} \lor \overline{C}$
form a clique of size 4.

These are the only such cliques in $\{A, \overline{A}, B, \overline{B}, AB, \overline{AB}, \overline{A}B, \overline{A}B, C, \overline{C}\}$

OR gate: Similar

Summary: if $C$ has $n$ variables

- $m_0$ NOT gates
- $m$, AND/OR gates
then \( G \) has

\[ 2n + 2m_0 + 6m_1 - 7 \]

vertices;

- All edges included except "the red pairs" in OR/NOT/AND gadgets

\[ R = n + m_0 + 2m_1 \]

Clique exists only if we select

- \( n \) input variables
- \( m_0 \) NOT gate vertices
- \( 2m_1 \) AND/OR gate vertices

All consistent \( \land Gm = \text{true} \).
Conclusions:

- Know P, NP
- Don't know: Does P = NP?
- Evidence so far: Been considered for 3 200 years & been unsuccessful.
- Also know NP-completeness...
  - CLIQ, SAT-COL, FACTORING
  - Harder
To prove new problem NP-complete...

Start with known NP-complete problem \((\text{Clique})\)

Show \(\text{Clique} \leq \text{BLAH}\)

Must also show \(\text{BLAH} \in \text{NP}\)

(i.e., \(\forall_{x} \text{ BLAH} \), s.t.

\[ \exists y \quad \forall_{x,y} (x, y) = 1 \iff \text{BLAH}(x) = 1 \]

In general, Reductions are powerful.