BU CS 332 – Theory of Computation

Lecture 21:

- NP completeness
- SAT is NP-c
- Clicque is NP-c

Reading:

Sipser Ch 7.3-7.4

Ran Canetti

November 24, 2020

Complexity class NP

Definition: NP is the class of languages decidable in polynomial time on a nondeterministic TM

$$\mathrm{NP} = \bigcup_{k=1}^{\infty} \mathrm{NTIME}(n^k)$$

An alternative characterization of NP

Definition: A TM V is a verifier for language L if:

- For any $x \in L, \quad \exists w \quad s.t. \; V(x,w) = 1$
- If $x \notin L, \quad then \; \forall w, \; V(x,w) = 0$

We say that V is polynomial-time if its runtime is polynomial in the length of its first input(i.e., length of x).

Theorem: A language $L \in \mathrm{NP}$ iff there is a polynomial-time verifier for L.

Is P = NP?

Is P = NP?

- We don’t have any reason to believe it is…

- There are many natural, important problems in NP that we don’t know how to solve in polynomial time.
  (E.g. SAT, HamiltonPath, Clique, SubsetSum, …)

How can we prove that $\mathrm{P} \neq \mathrm{NP}$?

Natural route: Show a language $\mathrm{L} \in \mathrm{NP}$ that cannot be decided in polynomial time.

But:

- Which language is best to choose?
- How will that help us with all the problems that we cannot solve in P?

How can we prove that $\mathrm{P} \neq \mathrm{NP}$?

Natural route: Show a language $\mathrm{L} \in \mathrm{NP}$ that cannot be decided in polynomial time.

But:

- Which language is best to choose?
- How will that help us with all the problems that we cannot solve in P?

Idea: Identify the "hardest" problems in NP

Find $L \in \mathrm{NP}$ such that $L \in \mathrm{P}$ iff $\mathrm{P} = \mathrm{NP}$

Recall: Mapping reducibility

Definition:

A function $f: \Sigma^* \rightarrow \Sigma^*$ is computable if there is a TM M which, given as input any $w \in \Sigma^*$, halts with only $f(w)$ on its tape.

Definition:

Language A is mapping reducible to language B, written

$$A \leq_{\mathrm{m}} B$$

if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that for all strings $w \in \Sigma^*$, we have $w \in A \Leftrightarrow f(w) \in B$

Polynomial-time reducibility

Definition:

A function $f: \Sigma^* \rightarrow \Sigma^*$ is polynomial-time computable if there is a polynomial-time TM M which, given as input any $w \in \Sigma^*$, halts with only $f(w)$ on its tape.

Definition:

Language A is polynomial-time mapping reducible to language B, written

$$A \leq_{\mathrm{p}} B$$

if there is a polynomial-time computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that for all strings $w \in \Sigma^*$, we have $w \in A \Leftrightarrow f(w) \in B$

Implications of poly-time reducibility

Theorem: If $A \leq_{\mathrm{p}} B$ and $B \in P$, then $A \in P$.

Theorem: If A $\leq_P$ B and B$\leq_P$ C, then A $\leq_P$ C.

NP-complete languages: The hardest in NP

A language B is **NP-complete** if

1. $B \in NP$
2. B is **NP-hard**, i.e., $\forall A \in NP, \; A \leq_p B$

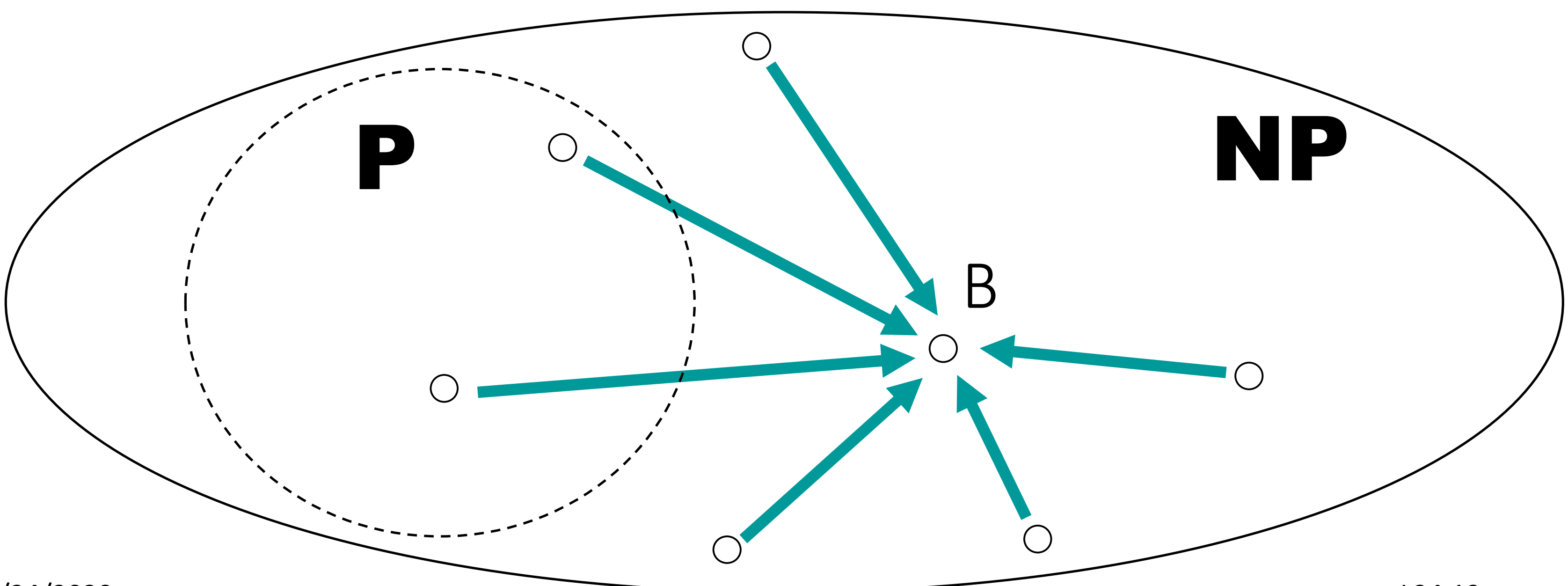
(every language in NP is poly-time reducible to B.)

NP-complete languages: The hardest in NP

A language B is **NP-complete** if

1. $B \in NP$
2. B is **NP-hard**, i.e., $\forall\, A \in NP,\ \ A \leq_p B$

(every language in NP is poly-time reducible to B.)

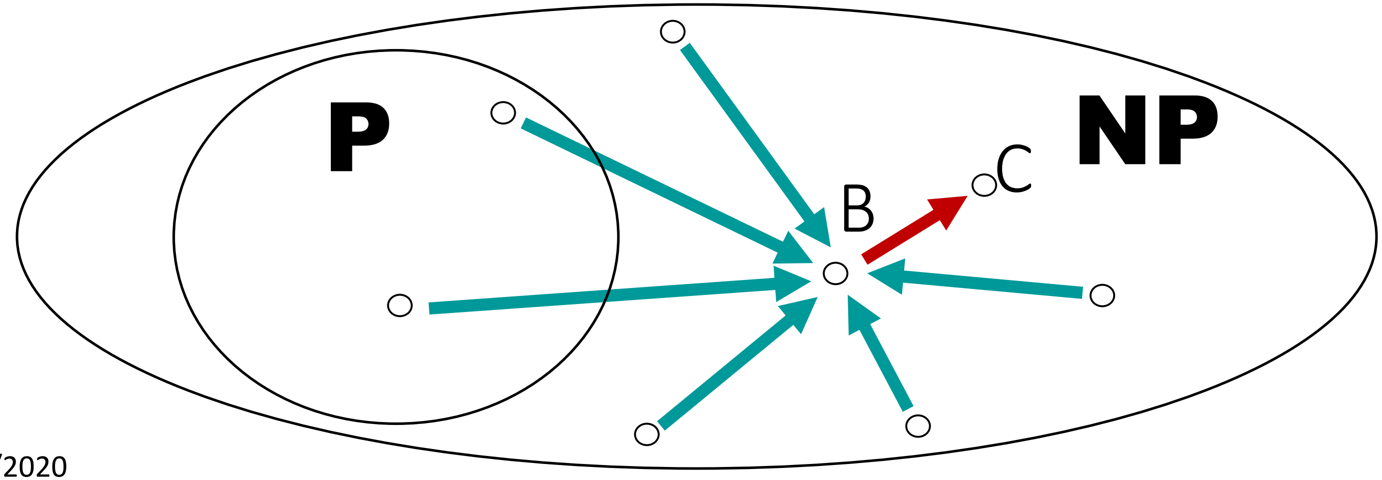


Implication of poly-time reductions

Theorem. If

- B is **NP**-complete,
- C$\in$ $\mathbf{NP}$ and
- B$\leq_p$C

then C is **NP**-complete.

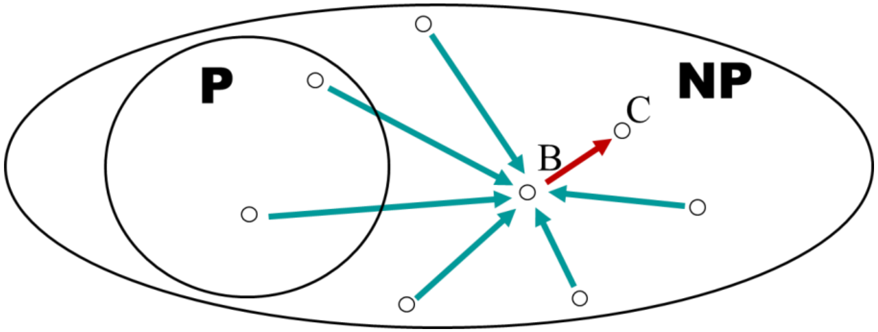


Implication of poly-time reductions

Theorem. If

- B is **NP**-complete,
- C$\in$ $\mathbf{NP}$ and
- B$\leq_p$C

then C is **NP**-complete.



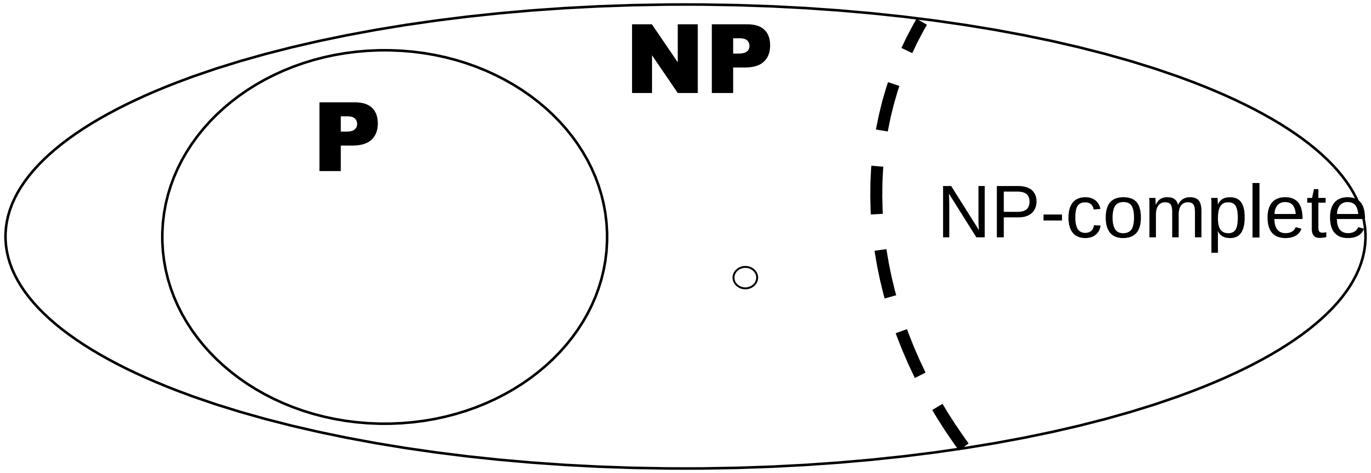
Theorem. If B is **NP**-complete and B$\in$ $\mathbf{P}$ then

$$\mathbf{P} = \mathbf{NP}.$$

(So, if B is **NP**-complete and $\mathbf{P} \neq \mathbf{NP}$

then there is no poly-time algorithm for B.)

NP-C problems: The hardest in NP



Different notions of reduction

Let $L \in NP$. Is the statement "If $\mathrm{L} \in \mathrm{P}$ then $\mathrm{P} = \mathrm{NP}$" equivalent to “L is NP-Complete”?

Different notions of reduction

Let $L \in NP$. Is the statement "If $\mathrm{L \in P}$ then $\mathrm{P=NP}$" equivalent to “L is NP-Complete”?

No!

-NP-C mandates a special form of reduction with nice properties (“many to one reductions”, or “Karp reductions”).

-More general (“Turing” or “Cook” reductions):

An NP-Complete problem

$$T_{NTM} = \{(N, x, 1^t):\ \ NTM\ N\ accepts\ x\ within\ \ t\ steps\}$$

T_{NTM} *Is NP-complete:*

- $T_{NTM} \in NP$
- For all $L \in NP$, $\ L \leq_p T_{NTM}$ *:*

A more natural language : SAT

"Is there an assignment to the variables in a logical formula that make it evaluate to true?"

- Boolean variable: Variable that can take on the value true/false (encoded as 0/1)
- Boolean operations: $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT)
- Boolean formula: Expression made of Boolean variables and operations. Ex: $(x_1 \vee \overline{x_2}) \wedge x_3$
- An assignment of 0s and 1s to the variables satisfies a formula φ if it makes the formula evaluate to 1
- A formula φ is satisfiable if there exists an assignment that satisfies it

Examples of NP languages: SAT

Ex: $(x_1 \vee \overline{x_2}) \wedge x_3$ Satisfiable?

Ex: $(x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge \overline{x_2}$ Satisfiable?

$$SAT = \{\langle \varphi \rangle | \varphi \text{ is a satisfiable formula}\}$$

Claim: $SAT \in \mathrm{NP}$

Cook-Levin Theorem

Theorem: SAT (Boolean satisfiability) is NP-complete

Proof: Already know $SAT \in \mathrm{P}$. Need to show every problem in NP reduces to SAT

Stephen A. Cook (1971)

Leonid Levin (1973)

Proof of Cook-Levin Theorem

- Proof idea
  - For each language A in NP, with a given input x for A, produce a Boolean formula ϕ that simulates the verification machine V for A on input x,w.

➔ ϕ is satisfiable if and only if there exists w such that V(x,w)=1.

Proof of Cook-Levin Theorem

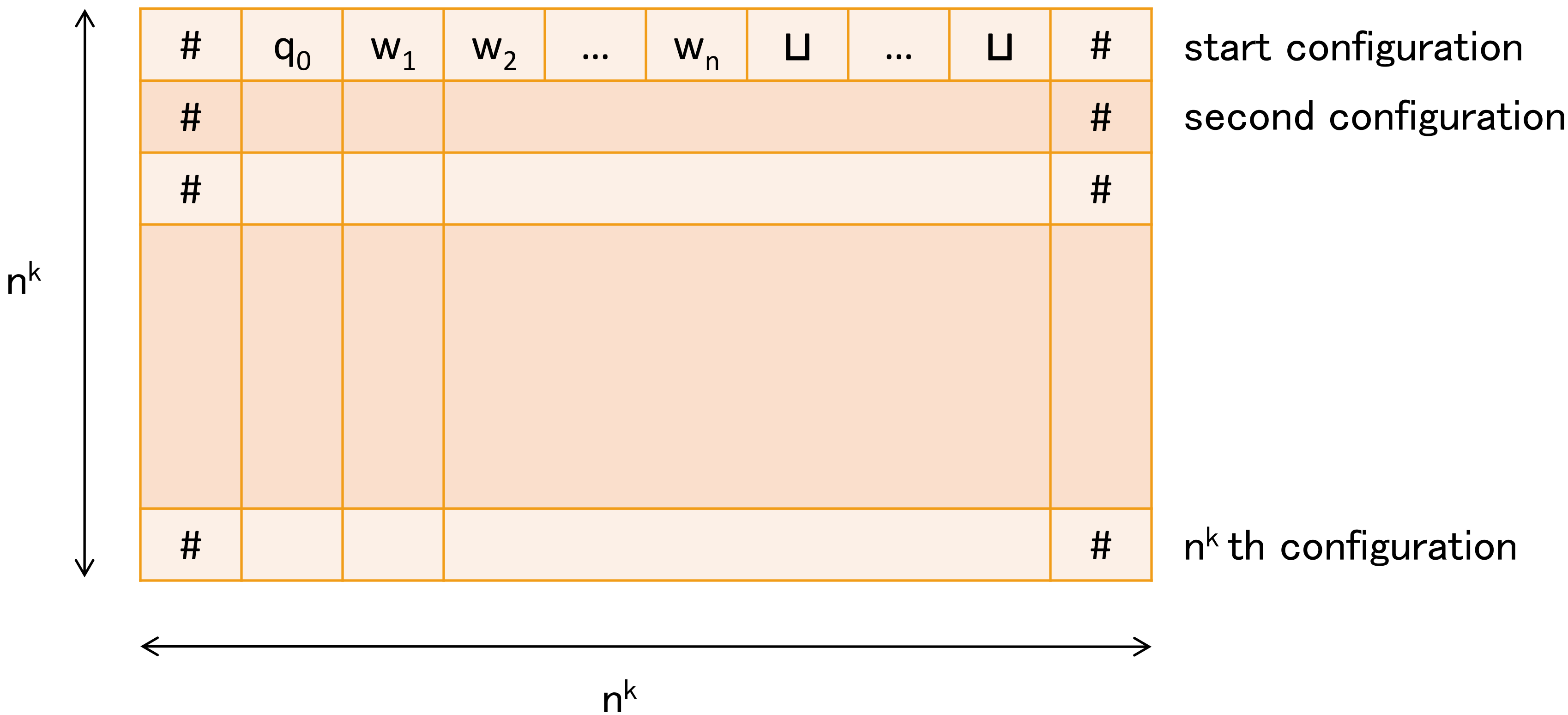
- Proof idea (cont.)
  - If there exist w s.t. V(x,w)=1, then there exists a series of configurations that results in the accept state, given x,w as the input of V.
  - We would construct a Boolean formula which is satisfiable if there exists such w.

Proof of Cook-Levin Theorem

- Proof
  - w: input
  - A: language
  - N: NP Turing machine that decides A
    - Assume that N decides whether $w \in A$ in n^k steps, for some constant k.

Proof of Cook-Levin Theorem

- Proof (cont.)
  - "$n^k \times n^k$-cell" ***tableau*** for N on input x,w



Proof of Cook-Levin Theorem

- Proof (cont.)
  - A variable could be represented as $x_{i,j,s}$.
  - $x_{i,j,s}$: true if cell[i,j] is s; otherwise, false.
  - cell[i,j]: the cell located on the ith row and the jth column.

#	q_0q_1	w_1	w_2	…	w_n	⊔	…	⊔	#
#									#
#									#
#									#

ntain many invalid

arting with the

ations not

corresponding the transition rules, not resulting in the accept state, and etc.

Proof of Cook-Levin Theorem

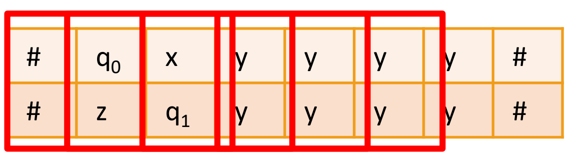
- Proof (cont.)
  - Produce a Boolean formula which forces the tableau to be valid and result in the accept state.

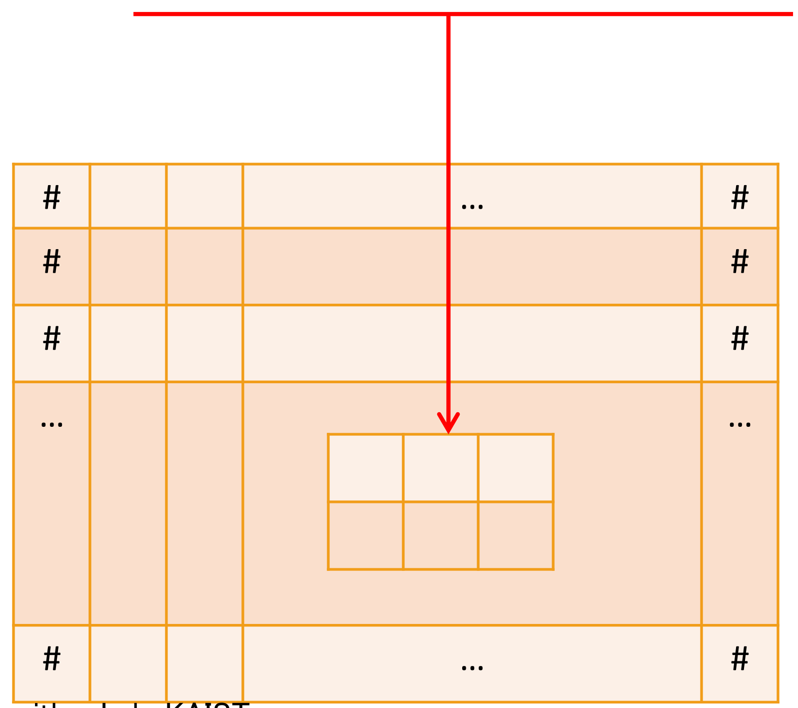
Proof of Cook-Levin Theorem

- Proof (cont.)
  - One cell can contain exactly one symbol among a state, a tape alphabet, and a #. (ϕ_{cell})
  - The first configuration should be corresponding to input w and the start state q_0. (ϕ_{start})
  - A configuration is derivable from the immediately previous configuration according to the transition rule of the Turing machine. (ϕ_{move})
  - There should exist a cell containing the accept state. (ϕ_{accept})
  - $\phi=\phi_{cell}\wedge\phi_{start}\wedge\phi_{move}\wedge\phi_{accept}$

Proof of Cook-Levin Theorem

- Proof (cont.)
  - $\phi=\phi_{cell}\wedge\phi_{start}\wedge\phi_{move}\wedge\phi_{accept}$
  - ϕ_{move} checks whether every 2×3 window is legal according to the transition rule of the Turing machine.





Proof of Cook-Levin Theorem

- Proof (cont.)
  - $\phi = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$
  - For example,
    - $\delta(q_1,a)=\{(q_1,b,R)\}$, $\delta(q_1,b)=\{(q_2,c,L), (q_2,a,R)\}$

a	q_1	b
q_2	a	c

a	a	q_1
a	a	b

#	b	a
#	b	a

some examples of **legal** 2 × 3 windows

a	b	a
a	a	a

a	q_1	b
q_1	a	a

b	q_1	b
q_2	b	q_2

some examples of **illegal** 2 × 3 windows

New NP-complete problems from old

Lemma: If $A \leq_{\mathrm{p}} B$ and $B \leq_{\mathrm{p}} C$, then $A \leq_{\mathrm{p}} C$

(poly-time reducibility is transitive)

Theorem: If $C \in \mathrm{NP}$ and $B \leq_{\mathrm{p}} C$ for some NP-complete language B, then C is also NP-complete

New NP-complete problems from old

All problems below are NP-complete and hence poly-time reduce to one another!