P & NP

Stepan Kuznetsov

Discrete Math Bridging Course, HSE University

The P Class

- Now we discuss only **decision problems:** that is, algorithmic questions with a “yes/no” answer.

The P Class

- Now we discuss only **decision problems:** that is, algorithmic questions with a “yes/no” answer.
- For convenience, let the input data be a word over an alphabet: $x \in \Sigma^*$.

The P Class

- Now we discuss only **decision problems:** that is, algorithmic questions with a “yes/no” answer.
- For convenience, let the input data be a word over an alphabet: $x \in \Sigma^*$.
- The size of input, $|x|$ is the length of x in symbols.

The P Class

- Now we discuss only **decision problems:** that is, algorithmic questions with a "yes/no" answer.
- For convenience, let the input data be a word over an alphabet: $x \in \Sigma^*$.
- The size of input, $|x|$ is the length of x in symbols.
- A decision problem is in the P class, if there exists an algorithm for solving it, whose **worst case** running time is bounded by $p(|x|)$.

The NP Class

- There are several equivalent definitions of the NP class.

The NP Class

- There are several equivalent definitions of the NP class.
- Def. 1: non-deterministic computations.

The NP Class

- There are several equivalent definitions of the NP class.
- Def. 1: non-deterministic computations.
  - The Turing machine may branch:

$$(p, a) \to (q_1, b_1, D_1)$$
$$(p, a) \to (q_2, b_2, D_2)$$

The NP Class

- There are several equivalent definitions of the NP class.
- Def. 1: non-deterministic computations.
  - The Turing machine may branch:

$$(p, a) \to (q_1, b_1, D_1)$$
$$(p, a) \to (q_2, b_2, D_2)$$

  - **Angelic choice:** if at least one execution trajectory yields “yes,” then the answer is “yes.”

The NP Class

- There are several equivalent definitions of the NP class.
- Def. 1: non-deterministic computations.
  - The Turing machine may branch:

$$
\begin{aligned}
(p, a) &\to (q_1, b_1, D_1) \\
(p, a) &\to (q_2, b_2, D_2)
\end{aligned}
$$

  - **Angelic choice:** if at least one execution trajectory yields "yes," then the answer is "yes."
  - One can implement **non-deterministic guess** (say, guess the satisfying assignment for a 3-CNF or guess a Hamiltonian cycle in a graph).

The NP Class

- Def. 2: hints.

The NP Class

- Def. 2: hints.
  - Denote the decision problem by $A(x)$.

The NP Class

- Def. 2: hints.
  - Denote the decision problem by $A(x)$.
  - $A(x) = 1 \iff \exists y\,(|y| < q(|x|)\,\&\,R(x, y) = 1)$, where $R \in P$.

The NP Class

- Def. 2: hints.
  - Denote the decision problem by $A(x)$.
  - $A(x) = 1 \iff \exists y \, (|y| < q(|x|) \,\&\, R(x, y) = 1)$, where $R \in P$.
  - y is a *hint*, given by someone to help us solve the problem.

The NP Class

- Def. 2: hints.
  - Denote the decision problem by $A(x)$.
  - $A(x) = 1 \iff \exists y \, (|y| < q(|x|) \,\&\, R(x, y) = 1)$, where $R \in P$.
  - y is a *hint*, given by someone to help us solve the problem.
  - Examples of y: the satisfying assignment; the Hamiltonian cycle; ...

The NP Class

- Equivalence of definitions:

The NP Class

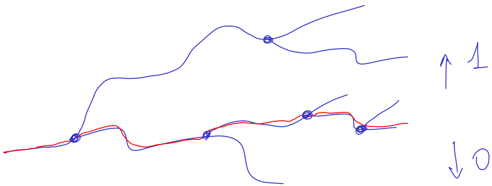
- Equivalence of definitions:
  - $2 \Rightarrow 1$: the hint can be guessed non-deterministically.

The NP Class

- Equivalence of definitions:
  - $2 \Rightarrow 1$: the hint can be guessed non-deterministically.
  - $1 \Rightarrow 2$: one can suppose that branching is binary. Then the hint is just the sequence of choices to be made.

The NP Class

- Equivalence of definitions:
    - $2 \Rightarrow 1$: the hint can be guessed non-deterministically.
    - $1 \Rightarrow 2$: one can suppose that branching is binary. Then the hint is just the sequence of choices to be made.



$y = 0100$

NP-Completeness

- Trivially, $\mathrm{P} \subseteq \mathrm{NP}$.

NP-Completeness

- Trivially, $\mathrm{P} \subseteq \mathrm{NP}$.
- Nobody knows, whether this inclusion is strict: say, whether 3-SAT $\in \mathrm{P}$.

NP-Completeness

- Trivially, $\mathrm{P} \subseteq \mathrm{NP}$.
- Nobody knows, whether this inclusion is strict: say, whether 3-SAT $\in \mathrm{P}$.
- As an *ersatz*, the theory of NP-completeness was invented.

NP-Completeness

- Trivially, $\mathrm{P} \subseteq \mathrm{NP}$.
- Nobody knows, whether this inclusion is strict: say, whether $3\text{-SAT} \in \mathrm{P}$.
- As an *ersatz*, the theory of NP-completeness was invented.
- Informally, NP-complete problems are the **hardest possible** problems in NP.

NP-Completeness

- Trivially, $P \subseteq NP$.
- Nobody knows, whether this inclusion is strict: say, whether 3-SAT $\in$ P.
- As an *ersatz*, the theory of NP-completeness was invented.
- Informally, NP-complete problems are the **hardest possible** problems in NP.
    - In particular, if an NP-complete problem is solvable in poly time, then $P = NP$.

NP-Completeness

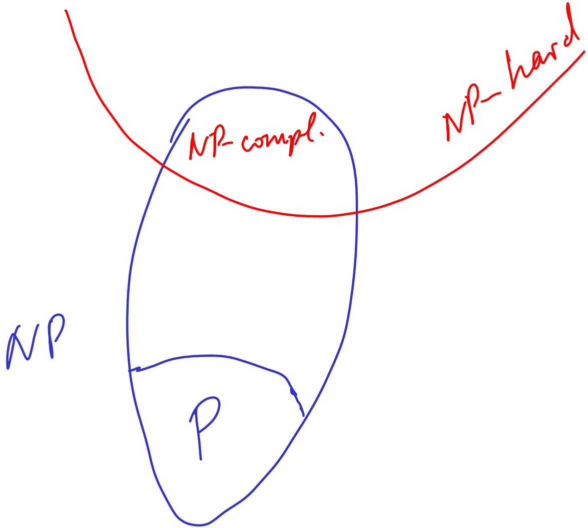
- Trivially, $\mathrm{P} \subseteq \mathrm{NP}$.
- Nobody knows, whether this inclusion is strict: say, whether 3-SAT $\in \mathrm{P}$.
- As an *ersatz*, the theory of NP-completeness was invented.
- Informally, NP-complete problems are the **hardest possible** problems in NP.
    - In particular, if an NP-complete problem is solvable in poly time, then $\mathrm{P} = \mathrm{NP}$.
    - Contraposition: if $\mathrm{P} \neq \mathrm{NP}$ (which highly likely), then any NP-complete problem is not in P.

NP-Completeness

- **m-reduction** (Carp reduction): A is reducible to B ($A \leq_m^P B$), if there exists a polytime computable function $f\colon \Sigma^* \to \Sigma^*$, such that $\boxed{A(x) = 1 \iff B(f(x)) = 1.}$
- The idea of reduction: if we can solve B, we can also solve A: $A(x) = B(f(x))$.
- A problem B is **NP-hard** if $A \leq_m^P B$ for any $A \in$ NP.
- B is **NP-complete** if $B \in$ NP and B is NP-hard.

Complexity Picture

(if $P \neq NP$)



Backwards Reduction

- Proving that a problem is NP-complete gives an evidence that it is hard (probably not polytime solvable).

Backwards Reduction

- Proving that a problem is NP-complete gives an evidence that it is hard (probably not polytime solvable).
- The common method of proving NP-hardness is **backwards reduction.**

Backwards Reduction

- Proving that a problem is NP-complete gives an evidence that it is hard (probably not polytime solvable).
- The common method of proving NP-hardness is **backwards reduction.**
    - Suppose we know A to be already NP-hard.

Backwards Reduction

- Proving that a problem is NP-complete gives an evidence that it is hard (probably not polytime solvable).
- The common method of proving NP-hardness is **backwards reduction.**
    - Suppose we know A to be already NP-hard.
    - In order to prove NP-hardness of a problem B, we reduce the **old** problem A to B.

Backwards Reduction

- Proving that a problem is NP-complete gives an evidence that it is hard (probably not polytime solvable).
- The common method of proving NP-hardness is **backwards reduction.**
    - Suppose we know A to be already NP-hard.
    - In order to prove NP-hardness of a problem B, we reduce the **old** problem A to B.
- But how to bootstrap and obtain the first example of an NP-complete problem?

Cook – Levin Theorem

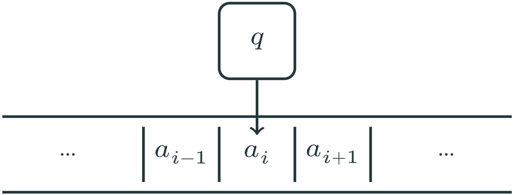
Theorem

SAT is NP-complete.

Cook – Levin Theorem

Proof sketch.

- Suppose $A \in$ NP, let us show $A \leq_m^P$ SAT.
- We encode each configuration of the Turing machine for A as a binary word:



$$0^m \quad a_1 \quad \dots \quad 0^m \quad a_{i-1} \quad q \quad a_i \quad 0^m \quad a_{i+1} \quad \dots$$

Cook – Levin Theorem

- The sequence of configurations (protocol) of A on input x is encoded by a binary matrix (b_{ij}) of size $(m \cdot p(|x|)) \times p(|x|)$.

Cook – Levin Theorem

- The sequence of configurations (protocol) of A on input x is encoded by a binary matrix (b_{ij}) of size $(m \cdot p(|x|)) \times p(|x|)$.
- Next, we construct a formula φ_x with variables $b_{00}, b_{01}, \ldots$ which expresses the fact that this matrix represents a correct protocol of a successful execution.

Cook – Levin Theorem

φ_x is a conjunction of the following claims:

1. the first row represents the configuration with x on the tape, the machine observing its first letter;
2. each next row is obtained from the previous one by one of the rules of the machine;
3. the last row includes state q_F and the answer “yes” (1).

Cook – Levin Theorem

φ_x is a conjunction of the following claims:

1. the first row represents the configuration with x on the tape, the machine observing its first letter;
2. each next row is obtained from the previous one by one of the rules of the machine;
3. the last row includes state q_F and the answer “yes” (1).

This is all expressible as Boolean formulae.

Cook – Levin Theorem

- The reducing function is $f\colon x \mapsto \varphi_x$.
- $A(x) = 1 \iff \varphi_x$ is satisfiable.
- Thus, $A \leq_m^P$ SAT.
- Since A was taken arbitrarily, we get NP-hardness of SAT.
- On the other hand, SAT is in NP, so it is NP-complete.

NP-completeness of 3-SAT

- 3-SAT is a special version of SAT, where only 3-CNFs are allowed.
- Trivially, 3-SAT $\leq_m^P$ SAT... but we need the opposite reduction!
- Let us show that SAT $\leq_m^P$ 3-SAT.

Tseitin's Transformations

Theorem

For any Boolean formula φ, there exists an equisatisfiable 3-CNF ψ of polynomial size.

- Equisatisfiability means that ψ is satisfiable iff so is φ.

Tseitin's Transformations

Theorem

For any Boolean formula φ, there exists an equisatisfiable 3-CNF ψ of polynomial size.

- Equisatisfiability means that ψ is satisfiable iff so is φ.
- Constructing an *equivalent* 3-CNF of polynomial size is not always possible: even translation to CNF can lead to exponential blowup.

Tseitin's Transformations

- Tseitin's transformations look like translation into 3-address (Assembler-like) code:
  $(a + b) * (c + d)$ is translated to
  "add a b t_1; add c d t_2; mul t_1 t_2 r"

Tseitin's Transformations

- Tseitin's transformations look like translation into 3-address (Assembler-like) code:
  $(a + b) * (c + d)$ is translated to
  "add a b t_1; add c d t_2; mul t_1 t_2 r"
- For each subformula we introduce a new variable and write the corresponding equivalences.

Tseitin's Transformations

Example: $(p \to q) \vee (q \to (p \to r))$

Tseitin's Transformations

Example: $(p \to q) \vee (q \to (p \to r))$

$$
\begin{aligned}
&(t_1 \leftrightarrow (p \to q)) \wedge \\
&(t_2 \leftrightarrow (p \to r)) \wedge \\
&(t_3 \leftrightarrow (q \to t_2)) \wedge \\
&(t_4 \leftrightarrow (t_1 \vee t_3)) \wedge \\
&t_4
\end{aligned}
$$

Tseitin's Transformations

Transform into 3-CNF by the following table:

$$
\begin{array}{l|l}
t_k \leftrightarrow (t_i \wedge t_j) & (\neg t_i \vee \neg t_j \vee t_k) \wedge (t_i \vee \neg t_k) \wedge (t_j \vee \neg t_k) \\
t_k \leftrightarrow (t_i \vee t_j) & (t_i \vee t_j \vee \neg t_k) \wedge (\neg t_i \vee t_k) \wedge (\neg t_j \vee t_k) \\
t_k \leftrightarrow (t_i \rightarrow t_j) & (\neg t_i \vee t_j \vee \neg t_k) \wedge (t_i \vee t_k) \wedge (\neg t_j \vee t_k) \\
t_k \leftrightarrow \neg t_i & (t_i \vee t_k) \wedge (\neg t_i \vee \neg t_k)
\end{array}
$$

Tseitin's Transformations

Transform into 3-CNF by the following table:

$t_k \leftrightarrow (t_i \wedge t_j)$	$(\neg t_i \vee \neg t_j \vee t_k) \wedge (t_i \vee \neg t_k) \wedge (t_j \vee \neg t_k)$
$t_k \leftrightarrow (t_i \vee t_j)$	$(t_i \vee t_j \vee \neg t_k) \wedge (\neg t_i \vee t_k) \wedge (\neg t_j \vee t_k)$
$t_k \leftrightarrow (t_i \rightarrow t_j)$	$(\neg t_i \vee t_j \vee \neg t_k) \wedge (t_i \vee t_k) \wedge (\neg t_j \vee t_k)$
$t_k \leftrightarrow \neg t_i$	$(t_i \vee t_k) \wedge (\neg t_i \vee \neg t_k)$

For our example, we get:

$$(\neg p \vee q \vee \neg t_1) \wedge (p \vee t_1) \wedge (\neg q \vee t_1) \wedge$$
$$(\neg p \vee r \vee \neg t_2) \wedge (p \vee t_2) \wedge (\neg r \vee t_2) \wedge$$
$$(\neg q \vee t_2 \vee \neg t_3) \wedge (q \vee t_3) \wedge (\neg t_2 \vee t_3) \wedge$$
$$(t_1 \vee t_3 \vee \neg t_4) \wedge (\neg t_1 \vee t_4) \wedge (\neg t_3 \vee t_4) \wedge t_4$$

Beyond Decision Problems

- An NP decision problem is the question **whether there exists** a *witness* y such that $R(x, y) = 1$.

Beyond Decision Problems

- An NP decision problem is the question **whether there exists** a *witness* y such that $R(x, y) = 1$.
    - E.g., a satisfying assignment for φ.

Beyond Decision Problems

- An NP decision problem is the question **whether there exists** a *witness* y such that $R(x, y) = 1$.
    - E.g., a satisfying assignment for φ.
- **Search problem:** yield a witness or say "no."

Beyond Decision Problems

- An NP decision problem is the question **whether there exists** a *witness* y such that $R(x, y) = 1$.
    - E.g., a satisfying assignment for φ.
- **Search problem:** yield a witness or say "no."
- **Counting problem** (the #P class): yield the number of witnesses.

Beyond Decision Problems

- An NP decision problem is the question **whether there exists** a *witness* y such that $R(x, y) = 1$.
    - E.g., a satisfying assignment for φ.
- **Search problem:** yield a witness or say "no."
- **Counting problem** (the #P class): yield the number of witnesses.
- Finally, we could ask for **all** witnesses.

Beyond Decision Problems

- The problem of yielding all witnesses could be unconditionally non-polynomial, since the answer could be exponential.

Beyond Decision Problems

- The problem of yielding all witnesses could be unconditionally non-polynomial, since the answer could be exponential.
- If $P = NP$, then any search problem is solvable in poly time.

Beyond Decision Problems

- The problem of yielding all witnesses could be unconditionally non-polynomial, since the answer could be exponential.
- If P $=$ NP, then any search problem is solvable in poly time.
  - Dichotomy: take $\varphi[p_1 := 0]$ and $\varphi[p_1 := 1]$, find out which is satisfiable, then do the same for $p_2, p_3, \ldots$.

Beyond Decision Problems

- The problem of yielding all witnesses could be unconditionally non-polynomial, since the answer could be exponential.
- If P $=$ NP, then any search problem is solvable in poly time.
  - Dichotomy: take $\varphi[p_1 := 0]$ and $\varphi[p_1 := 1]$, find out which is satisfiable, then do the same for $p_2, p_3, \ldots$.
  - This gives a poly-time algorithm for the search problem for 2-CNF.

Beyond Decision Problems

- The problem of yielding all witnesses could be unconditionally non-polynomial, since the answer could be exponential.
- If P $=$ NP, then any search problem is solvable in poly time.
    - Dichotomy: take $\varphi[p_1 := 0]$ and $\varphi[p_1 := 1]$, find out which is satisfiable, then do the same for $p_2, p_3, \ldots$.
    - This gives a poly-time algorithm for the search problem for 2-CNF.
- The counting problem could be harder than the decision one (example: DNF-SAT).