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 computation process may **branch:** at some point of execution, there could be more than one (but a finite number of) possibilities to perform the next step.

The NP Class

- There are several equivalent definitions of the NP class.
- Def. 1: non-deterministic computations.
    - The computation process may **branch:** at some point of execution, there could be more than one (but a finite number of) possibilities to perform the next step.
    - **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 computation process may **branch:** at some point of execution, there could be more than one (but a finite number of) possibilities to perform the next step.
    - **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 $R \in \mathrm{P}$.

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 $R \in \mathsf{P}$.
  - 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

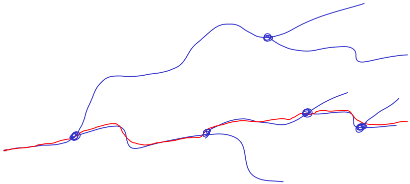
- Equivalence of definitions:
  - $2 \Rightarrow 1$: the hint can be guessed non-deterministically.

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.



$\uparrow 1$

$\downarrow 0$

$y = 0100$

NP-Completeness

- Trivially, $P \subseteq NP$.

NP-Completeness

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

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.

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.

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

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 $P = NP$.
    - Contraposition: if $P \neq NP$ (which is 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 $\boxed{A(x) = 1 \iff B(f(x)) = 1.}$

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: $A(x) = B(f(x))$.

NP-Completeness

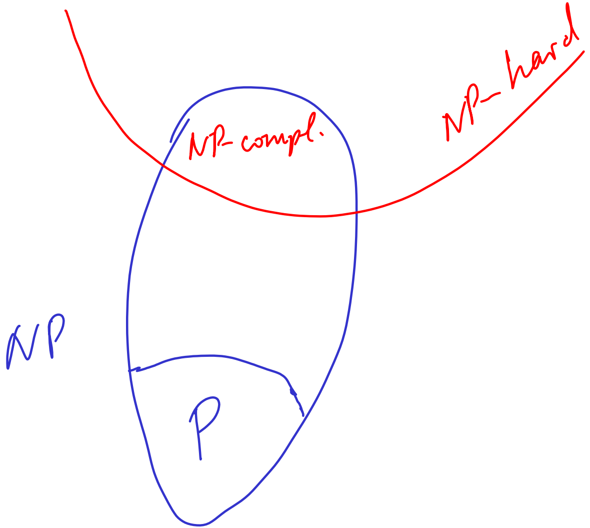
- **m-reduction** (Carp reduction): A is reducible to B ($A \leq_m^P B$), if there exists a polytime computable function $f\colon \Sigma^* \to \Sigma^*$, such that $\boxed{A(x) = 1 \iff B(f(x)) = 1.}$
- The idea of reduction: if we can solve B, we can also solve A: $A(x) = B(f(x))$.
- A problem B is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

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 (satisfiability of arbitrary Boolean formulae) NP-complete.