# P and NP

Hengfeng Wei

hengxin0912@gmail.com

June 16, 2016

# P and NP

# Computability theory first

### Theorem

*Halting problem is undecidable.*

### Proof.

□

# Complexity theory to follow

Is a given Sodoku configuration solvable?

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

Is $3 \times 3 \times 3$ Rubik's Cube solvable in 20 moves?

# Decision problems

**Definition (Decision problems.)**

Decision problems are problems whose solution is "Yes/No".

**Remarks.**

- decision problem vs. optimization problem
- decision problem is "hard" $\Rightarrow$ its optimization problem is "hard"

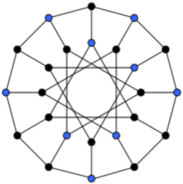
# Decision problems vs. optimization problem

INDEPENDENT SET

Optimization problem.

Instance: Undirected graph $G = (V, E)$.

Question: Find the maximal independent set in $G$.



Decision problem.

Instance: Undirected graph $G = (V, E)$ and an integer $k$.

Question: Does $G$ has an independent set of size (at least) $k$?

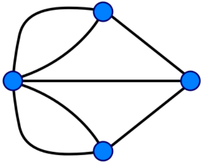
# The class P

## Definition (The class P)

P is the class of decision problems that are solvable in Polynomial time.

## Examples for the class P

Euler path.



Maze problem.

Primality testing.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | **2** | **3** | 4 | **5** | 6 | **7** | 8 | 9 | 10 |
| **11** | 12 | **13** | 14 | 15 | 16 | **17** | 18 | **19** | 20 |
| 21 | 22 | **23** | 24 | 25 | 26 | 27 | 28 | **29** | 30 |
| **31** | 32 | 33 | 34 | 35 | 36 | **37** | 38 | 39 | 40 |
| **41** | 42 | **43** | 44 | 45 | 46 | **47** | 48 | 49 | 50 |
| 51 | 52 | **53** | 54 | 55 | 56 | 57 | 58 | **59** | 60 |
| **61** | 62 | 63 | 64 | 65 | 66 | **67** | 68 | 69 | 70 |
| **71** | 72 | **73** | 74 | 75 | 76 | 77 | 78 | **79** | 80 |
| 81 | 82 | **83** | 84 | 85 | 86 | 87 | 88 | **89** | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | **97** | 98 | 99 | 100 |

# The class NP

**Definition (The class NP)**

NP is the class of decision problems that are solvable in Polynomial time by Non-deterministic algorithm.

$$\text{NP} \neq \text{Non-Polynomial}$$

$$\text{NP} \neq \text{No Problem}$$

# The class NP

**Definition (The class NP)**

NP is the class of decision problems that are solvable in Polynomial time by Non-deterministic algorithm.

$$\text{NP} \neq \text{Non-Polynomial}$$

$$\text{NP} \neq \text{No Problem}$$

**Definition (Non-deterministic polynomial algorithm.)**

Given an instance $\mathcal{I}$ of a decision problem:

Guessing: generate a certificate $c$ for $\mathcal{I}$

Verifying: $V(\mathcal{I}, c)$

$$O(\text{Guessing}) + O(\text{Verifying}) = O(n^c)$$

# Proof of being in NP

**Theorem**

*INDEPENDENT SET* $\in$ *NP.*

**Proof.**

Given $G = (V, E)$ and $k$:

Guessing: Nondeterministically select a subset $c$ of $k$ vertices of $G$.

Verifying: Test whether $G$ contains no edges for all vertices pairs in $c$.

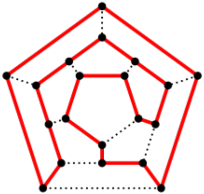
Output: If the test passes, ouput “yes”; otherwise, output “no”.

The complexity is $O(n^2)$. $\square$

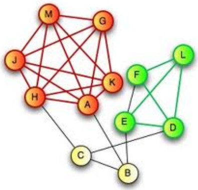
# The class NP

## Examples for the class NP

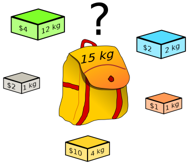
Hamiltonian path.



Clique problem.




Knapsack problem.

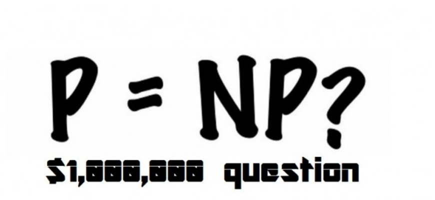


# P vs. NP

## P vs. NP

P: polynomially solvable

NP: polynomially verifiable



# P vs. NP

**Theorem**

$$P \subseteq NP.$$

**Proof.**

To design a non-deterministic polynomial algorithm given a deterministic polynomial algorithm (PA).

- Guessing: generate a certificate $c$ for instance $\mathcal{I}$.
- Verifying: ignore $c$; output $\text{PA}(\mathcal{I})$.

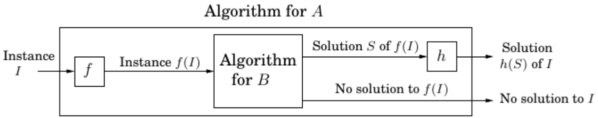
□

# Reduction

**Why reduction?**

To relate complexities of individual problems.

**Definition (Reduction.)**



# Polynomial reduction

**Definition (Polynomial reduction.)**

$f(I)$ in polynomial time.

**Theorem**

*Transitivity:* $A \leq_P B, B \leq_P C \Rightarrow A \leq_P C$.

**Proof.**

- $A \leq_P B \Rightarrow \exists \text{poly. } f : a \in A \iff f(a) \in B$
- $B \leq_P C \Rightarrow \exists \text{poly. } g : b \in B \iff g(b) \in C$
- $a \in A \iff \text{poly. } g(f(a)) \in C \Rightarrow A \leq_P C$

□

# Polynomial reduction

**Theorem**

$A \leq_P B, B \in P \Rightarrow A \in P.$

**Proof.**

- $A \leq_P B \Rightarrow \exists$poly. $f : a \in A \iff f(a) \in B$
- $B \in$ P $\Rightarrow \exists$poly. $g : g$ solves $B$
- poly. $g(f(\cdot))$ solves $A$

$\square$

# P and NP

# NP-hard

**Definition (NP-hard)**

$B$ is NP-hard if

$$\forall A \in \mathsf{NP} : A \leq_P B.$$

**Remark.**

NP-hard problems are at least as hard as any problem in NP.

# NP-complete

**Definition (NP-complete)**

$B$ is NP-complete if:

1. $B \in \mathrm{NP}$
2. $B$ is NP-hard.

**Remark.**

NP-complete problems are the hardest problems in NP.

# NP-complete

**Theorem**

*If $B$ is NP-complete and $B \in$ P, then P $=$ NP.*

**Proof.**

1. P $\subseteq$ NP
   - already proved
2. NP $\subseteq$ P:
   - $\forall A \in \text{NP}, A \leq_P B \wedge B \in \text{P} \Rightarrow A \in \text{P}$

□

# NP-complete

**Theorem**

*If $B$ is NP-complete and $B \leq_P C \in NP$, then $C$ is NP-complete.*

**Proof.**

1. $C \in$ NP
2. $C$ is NP-hard:
   - $\forall A \in \text{NP}, A \leq_P B \leq_P C \Rightarrow A \leq_P C$.

$\square$

# Proof of being in NP-complete

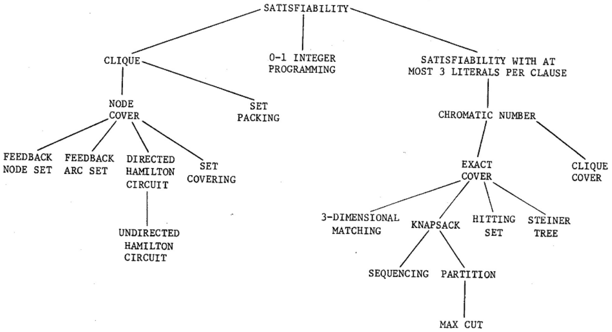
To prove $C$ is NP-complete.

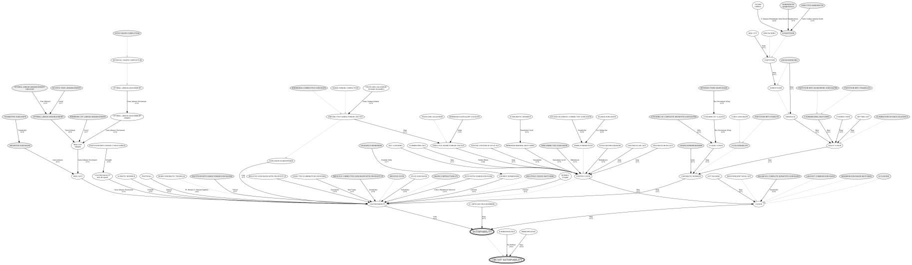
1. $C \in$ NP:
   - non-deterministic polynomial algorithm
2. $C$ is NP-hard:
   - choose a known NP-complete problem $B$
   - prove $B \leq_P C$

# NP-complete problem

The first known NP-complete problem.

SAT (circuit satisfiablity) is NP-complete.





http://adriann.github.io/npc/npc.html


https://github.com/hengxin/algorithm-ta-tutorial.git