CS711008Z Algorithm Design and Analysis

Lecture 4. **NP** and intractability (Part II) [1]

Dongbo Bu

Institute of Computing Technology
Chinese Academy of Sciences, Beijing, China

[1] The slides were prepared based on Introduction to algorithms, Algorithm design, Computer and Intractability, and slides by Kevin Wayne with permission.

Outline

- Reduction: understanding the relationship between different problems. $A \leq_P B$ implies "B is harder than A".
- Problem classes: **P**, **NP**, **coNP**, **L**, **NL**, **PSPACE**, **EXP**, etc.
- CIRCUIT SATISFIABILITY is one of the hardest problems in **NP** class.
- **NP-Complete** problems

Complexity classes

- A complexity class of problems is specified by several parameters:
  1. Computation model: multi-string Turing machine;
  2. Computation mode: When do we think a machine accepts its input? deterministic or non-deterministic?
  3. Computation resource: time, space.
  4. Bound: a function f to express how many resource can we use.
- The complexity class is then defined as the set of all languages decided by a multi-string Turing machine M operating in the deterministic/non-deterministic mode, and such that, for input x, M uses at most $f(|x|)$ units of time or space.

(See ppt for description of Turing machine.)

Deterministic versus nondeterministic

- **DTM**: In a deterministic Turing machine, the set of rules prescribes at most one action to be performed for any given situation.
- **NTM**: A non-deterministic Turing machine (NTM), by contrast, may have a set of rules that prescribes more than one actions for a given situation.
- For example, a non-deterministic Turing machine may have both "If you are in state 2 and you see an 'A', change it to a 'B' and move left" and "If you are in state 2 and you see an 'A', change it to a 'C' and move right" in its rule set.

Example: NFA and DFA

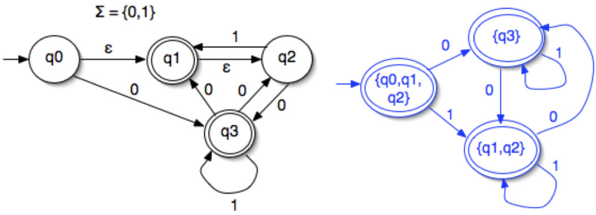
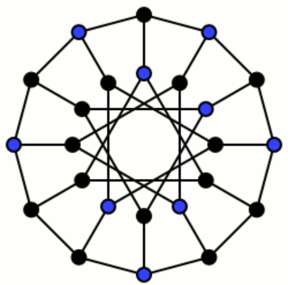


Figure: NFA and DFA

- Perhaps the easiest way to understand determinism and nondeterminism is by looking at NFA and DFA.
- In a DFA, every state has exactly one outgoing arrow for every letter in the alphabet.
- However, the NFA in state 1 has two possible transitions for the letter "b".

DTM vs. NTM: the difference between finding and verifying a solution



- Consider the INDEPENDENT SET problem: does the given graph have an independent set of 9 nodes?
- If your answer is "Yes", you just need to provide a **certificate** having 9 nodes.
- **Certifier:** it is easy to verify whether the certificate is correct, i.e., the given 9 nodes form an independent set for this graph of 24 vertices.
- **Solver:** However, it is not easy to find this independent set

Another example

- Consider the following problem: does the formula $f(x) = x^5 - 3x^4 + 5x^3 - 7x^2 + 11x - 13 = 0$ have a real-number solution?
- If your answer is "`Yes`", you just need to provide a **certificate**, say $x = 0.834...$.
- **Certifier:** it is easy to verify whether the certificate is correct, i.e., $f(x) = 0$.
- **Solver:** however, it is not easy to get a solution.

P class

- **P**: decision problems for which there is a polynomial-time algorithm to **solves** it.
- Here, we say that an algorithm A **solves** problem X if for all instance s of X, $A(s) = \texttt{YES}$ iff s is a $\texttt{YES}$instance.
- Time-complexity: A runs in polynomial-time if for **every** instance s, $A(s)$ ends in at most $polynomial(|s|)$ steps.
- Stable Matching problem: $O(n^2)$.

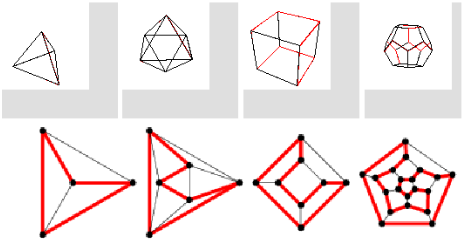
NP class

- **NP**: decision problems for which there exists a polynomial-time **certifier**. [2]
- Here we say that an algorithm $C(s,t)$ **certificates** problem X if for each "YES" instance s, there exists a **certificate** t such that $C(s,t) =$YES, and $|t| = polynomial(|s|)$.
- **Certificate**: an evidence to demonstrate this instance is YES;
- Note: a certifier approach the problem from a **managerial** point of view as follows:
  - It will not actually try to solve the problem directly;
  - Rather, it is willing to efficiently evaluate proposed "proof".

[2] **NP** denotes "non-deterministic polynomial-time". This is just simple but equivalent definition.

Certificate and certifier for Hamilton Cycle problem

- Problem: Is there a Hamiltonian cycle in the give graph?
- If your answer is YES, you just need to provide a certificate, i.e. a permutation of n nodes;
- Certifier: checking whether this path forms a cycle;
- Example:
- Certifier: it takes polynomial time to verify the certificate. Thus Hamilton Cycle is in **NP** class.



Certificate and certifier for SAT problem

- Problem: Is the given **CNF** satisfiable?
- If your answer is YES, you just need to provide a certificate, i.e. an assignment for all n boolean variables;
- Certifier: checking whether each clause is satisfied by this assignment;
- Example:
    - An instance: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$
    - Certificate: $x_1 = \mathtt{TRUE}, x_2 = \mathtt{TRUE}, x_3 = \mathtt{FALSE}$;
    - Certifier: it takes polynomial time to verify the certificate. Thus SAT is in **NP** class.

The "certificate" idea is not entirely trivial.

1. For UNSAT problem, it is difficult to provide a short "certificate":
   - Suppose we want to prove a SAT instance is **unsatisfiable**, what evidence could convince you, in polynomial time, that the instance is unsatisfiable?
2. In addition, we can also transform a **certifier** into an **algorithm**.
   - A certifier can be used as the core of a "brute-force" algorithm to solve the problem: enumerate all possible certificate t in $O(2^{|t|})$ time, and run $C(s,t)$. It will take exponential-time.

Problem classes: **P**, **NP**, and **EXP**

Three classes of problems:

- **P:** decision problems for which there is a polynomial-time algorithm;
- **NP:** decision problems for which there is a polynomial-time certifier;
- **EXP:** decision problems for which there is an exponential-time algorithm;

$\mathbf{P} \subseteq \mathbf{NP}$

Theorem

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

Proof.

- Consider any problem X in **P**;
- There is an algorithm A to solve it;
- We design a certifier C as follows: when presented with input (s, t), simply return $A(s)$.

□

$\mathbf{NP} \subseteq \mathbf{EXP}$

Theorem

$\mathbf{NP} \subseteq \mathbf{EXP}$.

Proof.

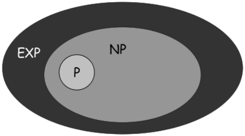
- Consider any problem X in **NP** ;
- There is a polynomial-time certifier C to certificate it;
- For an instance s, run $C(s,t)$ on **all** possible certificates t, $|t| = polynomial(|s|)$;
- Return `Yes` if $C(s,t)$ returns `Yes` for any certificate t.

□

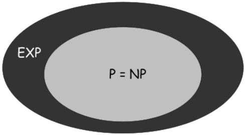
Question 1: $\mathbf{P} = \mathbf{NP}$?

P vs. NP

- The main question: $\mathbf{P} = \mathbf{NP}$? [S. Cook]
- In other words, is solving a problem as easy as certificating an evidence?
  - If **P = NP**, then for a "Yes" instance, an efficient "verifying" a certificate means an efficient "finding" a solution, and there will be efficient algorithms for SAT, TSP, HAMILTON CYCLE, etc.
  - If **P** $\neq$ **NP**: there is no efficient algorithms for these problems;
- Clay \$7 million prize. (See http://www.claymath.org/millennium/P_vs_NP/)



If P ≠ NP



If P = NP

A first NP-Complete problem

NP – complete class: the hardest problem in **NP** class

- Due to the absence of progress of **P=NP?** question, a more approachable question was posed:
  What is the hardest problems in **NP**?
- This is approachable since by using polynomial-time reduction, one can find connection between problems, and gain insight of the structure of **NP** class.
- The hardest problems in the NP class:
  - **NP-hard**: a problem Y is **NP-hard** if for **any** **NP** problem X, $X \leq_p Y$;
  - **NP-complete**: a problem Y is in **NP**, and is **NP-hard**.

Theorem

Suppose Y is a **NP-complete** *problem. Y is solvable in polynomial-time iff* **P=NP**

Proof.

- Let X be any problem in **NP** ;
- Since $X \leq_P Y$, X can be solved in polynomial-time through the "reduction algorithm".

□

- Consequence: if there is any problem in **NP** that cannot be solved in polynomial-time, then no NP-Complete can be solved in polynomial-time.

The first NP-Complete problem [Cook, Levin 1971]

- It is not at all obvious that **NP-complete** problems should even exist.
- Two possible cases:
  1. two incomparable problem X' and X'', and there is no problem X such that $X' \leq_P X$, and $X'' \leq_P X$?
  2. an infinite sequence of problems $X_1 \leq_P X_2 \leq_P$ ...;
- The difficulty is to prove that **any** **NP** problem X can be reduced to a **NP-complete** problem.


Figure: Stephen Cook and Leonid Levin

In 1982, Cook received the Turing award. His citation reads: *For his advancement of our understanding of the complexity of computation in a significant and profound way. His seminal paper, The Complexity of Theorem Proving Procedures,..., laid the foundations for the theory of NP-Completeness. The ensuing exploration of the boundaries and nature of NP-complete class of problems has been one of the most active and important research activities in computer science for the last decade.*

- CIRCUIT: a labeled, directed acyclic graph to simulate computation process on physical circuit.

CIRCUIT SATISFIABILITY problem

INPUT: a circuit;
OUTPUT: is there an assignment of input making output to be 1?

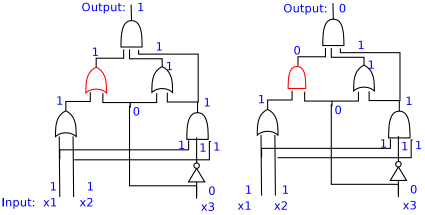
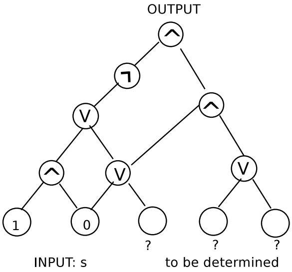


Figure: Left: satisfiable. Right: unsatisfiable.

Circuit Satisfiability problem

INPUT: a circuit;
OUTPUT: is there assignment of input that cause the output to take the value 1?

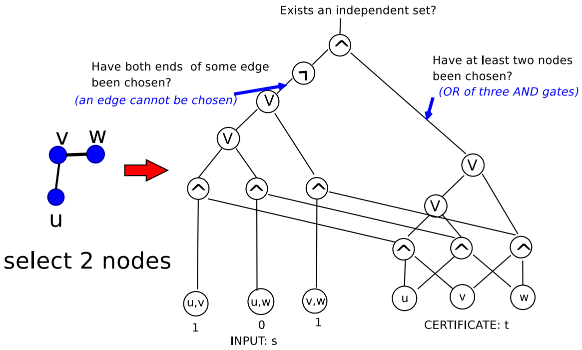


CIRCUIT SATISFIABILITY is the most natural problem.

- For example, INDEPENDENT SET problem can be reducible to CIRCUIT SATISFIABILITY.
- In other words, a circuit can be designed to simulate certifier of INDEPENDENT SET problem, i.e., the circuit can be satisfied iff the INDEPENDENT SET instance is a "Yes" instance.

CIRCUIT SATISFIABILITY problem

CIRCUIT SATISFIABILITY problem can be used to represent a large family of problems, say INDEPENDENT SET $\leq_P$ CIRCUIT SATISFIABILITY.



- Existing an independent set $\Rightarrow C$ is satisfiable.
- No independent set $\Rightarrow C$ is unsatisfiable.

CIRCUIT SATISFIABILITY is the most natural problem.

- In fact, besides INDEPENDENT SET problem, ALL **NP** problems can be reducible to CIRCUIT SATISFIABILITY.
- In other words, specific circuits can be designed to simulate the certifiers of ALL **NP** problems.
- CIRCUIT SATISFIABILITY is NP-Complete.

Theorem

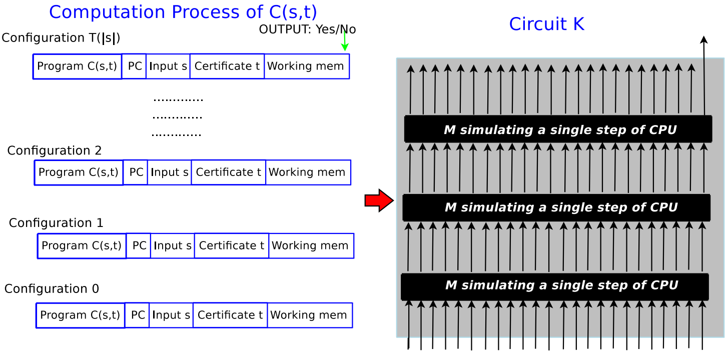
CIRCUIT SATISFIABILITY *is NP-Complete.*

Proof.

- We will show for any problem $X \in NP$, $X \leq_P$ CIRCUIT-SAT.
- Remember that $X \in NP$ implies a certifier $C(s,t)$ running in $T(|s|) = poly(|s|)$ time.
- And s is a "Yes" instance of $X \Leftrightarrow$ there is a certificate t of length $p(|s|)$ such that $C(s,t) = Yes$.
- Our objective is to design a circuit that generates same output to the certifier $C(s,t)$.
- Key idea: Represent the computation process of certifier $C(s,t)$ as a sequence of configurations. Here, configuration refers to any particular state of computer, including program $C(s,t)$, program counter PC, memory, etc. (You can image configuration as the tape of a universal Turing machine.)
- The i-th configuration is transformed to the $(i+1)$-st configuration by a combinatorial circuit M simulating CPU (in a single clock cycle).
- Simply paste $T(n)$ copies of M to generate a single circuit K.
- When inputed with initial configuration, K will generate ALL configurations.
- The output (a specific bit in working memory) appears on a pin.

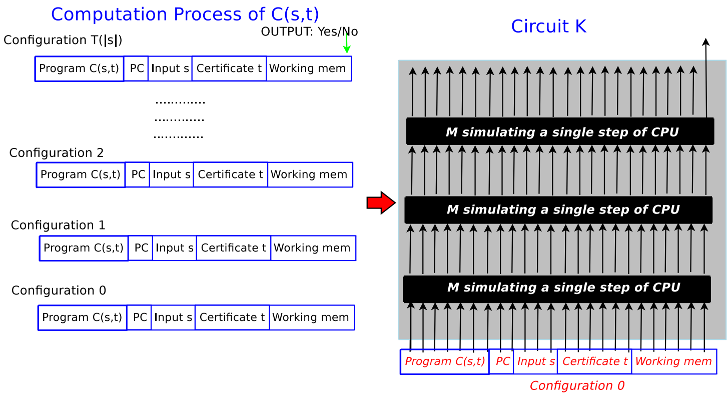
□

Certifier ⇒ circuit: an example



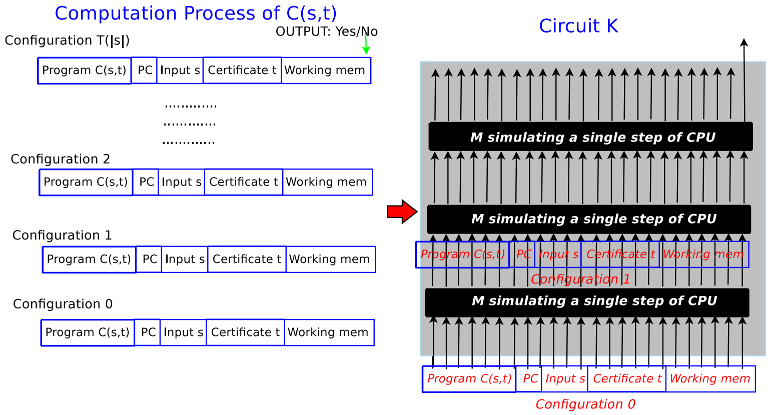
- Configuration: any particular state of computer, including program $C(s,t)$, program counter PC, working memory, etc.
- Transformation: simply connecting $T(n)$ copies of physical circuit M to generate a single circuit.
- Note that both #configuration and #working_memory are polynomial.

Certifier ⇒ circuit: an example



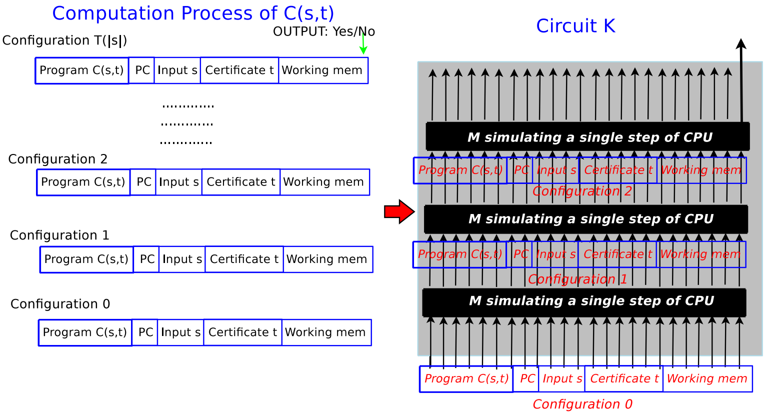
- Equivalence: When inputed with the initial configuration, ALL configurations will appear step-by-step (as how CPU does in a single clock cycle). Finally a specific pin outputs Yes/No.

Certifier ⇒ circuit: Step 1



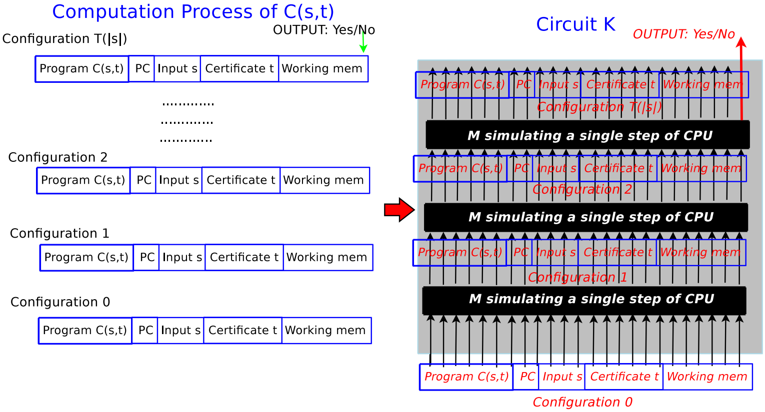
- Equivalence: configuration 1 will appear in the second layers of pins when inputed with initial configuration.

Certifier ⇒ circuit: Step 2



- Equivalence: configuration 2 will appear in the third layers of pins when inputed with initial configuration.

Certifier ⇒ circuit: Step $T(|s|)$



- Equivalence: configuration $T(|s|)$ will appear in the topest layers of pins. A specific pin reports Yes/No. Thus, circuit K outputs "Yes" ⇔ certifier $C(s,t)$ reports "Yes".

Proving further NP-Complete problems

Proving further NP-Complete problems

- Once we have a first **NP-complete**, we can discover many more via the following property:

Theorem

If Y is an **NP-complete***, and X is in* **NP** *with the property $Y \leq_P X$, then X is also NP-Complete.*

- General strategy for proving new problem X NP-Complete:
  1. Prove that X is in NP;
  2. Choose an NP-Complete problem Y;
  3. Consider an arbitrary instance y of Y, and show how to construct, in polynomial-time, an instance x of X, such that y is a "Yes" instance $\Leftrightarrow$ x is a "Yes" instance.

Theorem

SAT *is* **NP-complete**.

(Part 1: SAT belongs to NP.)

Proof.

- Certificate: assignment of variables.
- Certifier: simply evaluate each clause and Φ.

□

e.g., $\Phi = (x_1 \vee \neg x_2 \vee x_3)$ Certificate: $x_1 = T$ $x_2 = T$ $x_3 = T$.

Theorem

SAT *is NP-Complete.*

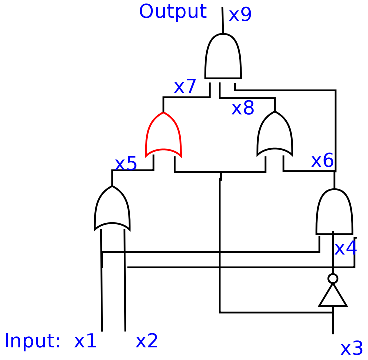
(Part 2: SAT is **NP-hard**. In particular, CIRCUIT SATISFIABILITY $\leq_P$ SAT)

Proof.

- each wire in $C \Rightarrow$ a variable;
- a gate in $C \Rightarrow$ a formula involving variables of incident wires;
- Φ is the AND of output variable with the conjunction of clauses of all gates.
- The CIRCUIT SATISFIABILITY instance is satisfied iff the constructed SAT instance is satisfied.

□

Circuit Satisfiability $\leq_P$ SAT



phi = x9 ∧
(x9 <-> x7 ∧ x8 ∧ x6) ∧
(x7 <-> x3 V x5) ∧
(x8 <-> x3 V x6) ∧
(x6 <-> x1 ∧ x2 ∧ x4) ∧
(x5 <-> x1 V x2) ∧
(x4 <-> NOT x3)

Theorem

$\mathrm{3SAT}$ *is NP-Complete.*

[3] ($\mathrm{3SAT}$: each clause has exactly 3 literals.)

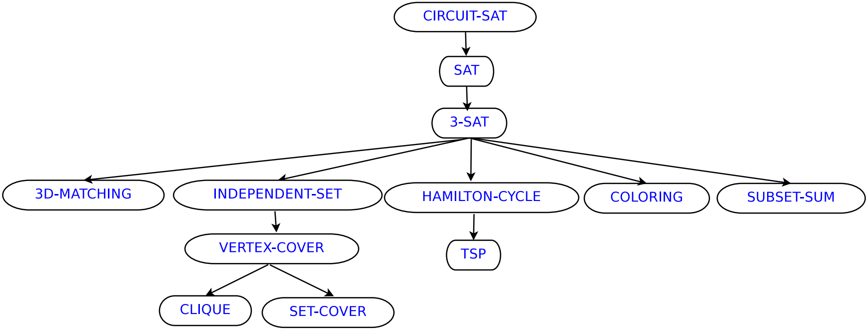
Proof.

- 1 literal: $(x_1) \iff$
  $(x_1 \vee p \vee q) \wedge (x_1 \vee p \vee \neg q) \wedge (x_1 \vee \neg p \vee q) \wedge (x_1 \vee \neg p \vee \neg q)$
- 2 literals: $(x_1 \vee x_2) \iff (x_1 \vee x_2 \vee p) \wedge (x_1 \vee x_2 \vee \neg p)$
- 3 literals: simply copy it.
- 4 literals:
  $(x_1 \vee x_2 \vee x_3 \vee x_4)$
  $\iff (x_1 \vee x_2 \vee p) \wedge (p \leftrightarrow x_3 \vee x_4)$
  $\iff (x_1 \vee x_2 \vee p) \wedge (\neg p \vee x_3 \vee x_4) \wedge (p \vee \neg x_3) \wedge (p \vee \neg x_4) \ldots$
- and so on....

□

[3] $\mathrm{2SAT}$ belongs to **P**. See slides by D. Moshko.

Thus the following problems are NP-Complete.



A partial taxonomy of hard problems

Given a collection of objects,

1. PACKING problems: to choose at least k of them. Restrictions: conflicts among objects, e.g. INDEPENDENT SET
2. COVERING problems: to choose at most k of them to meet a certain goal, e.g., SET COVER, VERTEX COVER.
3. PARTITIONING problems: to divide them into subsets so that each object appears in exactly one of the subsets, e.g., 3-COLORING.
4. SEQUENCING problems: to search over all possible permutations of objects under restrictions that some objects cannot follow certain others, e.g., HAMILTON CYCLE, TSP;
5. NUMERICAL problems: objects are weighted, to select objects to meet the constraint on the total weights, e.g., SUBSET SUM
6. CONSTRAINT SATISFACTION problems. e.g., 3SAT, CIRCUIT SATISFIABILITY.

The asymmetry of **NP** and **coNP**

The asymmetry of **NP**

NP is fundamentally asymmetry since:

- For a “Yes” instance, we can provide a short “certificate” to support it is “Yes”;
- But for a “No” instance, no correspondingly short “Disqualification” is guaranteed;

Example: SAT vs. $UNSAT$.

- Certificate of a “Yes” instance: an assignment;
- Disqualification of a “No” instance: ?

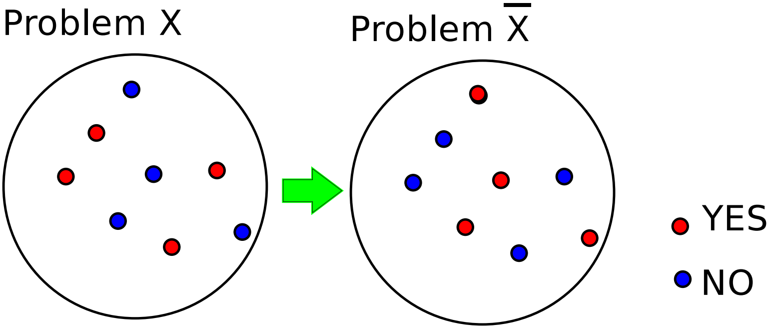
Example: Hamilton Cycle vs. No Hamilton Cycle

- Certificate of a “Yes” instance: a permutation of nodes;
- Disqualification of a “No” instance: ?

Problem X and its complement $\bar{X}$

- $\bar{X}$ has different property: s is a "Yes" instance of $\bar{X}$ iff for ALL t of length at most $p(|s|)$, we have $C(s,t) = No$.
- co-NP: the collection of X if $\bar{X}$ is in **NP**.

Example: $UNSAT$, No Hamilton Cycle.



Question 2: NP = coNP?

If yes, then the existence of short certificates for "Yes" instances means that we can find short disqualifications for all "No" instances.

NP = coNP?

- Widespread belief: No.
- Just because we have a short certificate for all "Yes" instances, it is not clear why we should believe that the "No" instances also have a short certificate.
- Proving **NP=coNP** is a bigger step than **P=NP**.

Theorem

P=NP $\Rightarrow$ **NP=coNP**.

Proof.

- Key idea: **P** is closed under complementation, i.e., $X \in P \Leftrightarrow \bar{X} \in P$.
- $X \in \mathbf{NP} \Rightarrow X \in P \Rightarrow \bar{X} \in P \Rightarrow \bar{X} \in \mathbf{NP} \Rightarrow X \in \mathbf{coNP}$, and
- $X \in co - NP \Rightarrow \bar{X} \in \mathbf{NP} \Rightarrow \bar{X} \in P \Rightarrow X \in P \Rightarrow X \in NP$.

□

Good characterizations: the class $\mathbf{NP} \cap \mathbf{coNP}$

If X is in both **NP** and **coNP**, it has a nice property:

1. if an instance is "Yes" instance, we have a short proof;
2. if the input instance is a "No" instance, we have a short disqualification, too.

Example: MAXIMUM FLOW

- Certificate for "Yes" instance: list a flow of value $\geq v$ directly;
- Certificate for "No" instance: list a cut whose capacity $\leq v$;

Duality immediately implies that both problems are in **NP** and $coNP$.

Question 3: $\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$?

If yes, a problem with good characterization always has an efficient algorithm.

Good characterizations: the class $\mathbf{NP} \cap \mathbf{coNP}$

Mixed opinions:

- finding good characterization is usually easier than designing an efficient algorithm;
- good characterization $\Rightarrow$ conceptual leverage in reasoning about problems;
- good characterization $\Rightarrow$ efficient algorithm: There are many cases in which a problem was found to have a nontrivial good characterization; and then (sometimes many years later) it was discovered to have a polynomial-time algorithm.

Examples:
— linear programming [Khachiyan 1979]
— primality testing [Agrawal-Kayal-Saxena, 2002]
[4]

[4] These slides are excerpted from the presentation by Kevin Wayne.

Four possibilities for the relationships among **P**, **NP**, and **coNP**.