VLSI Layout Design
Overview (2) Theoretical Aspect

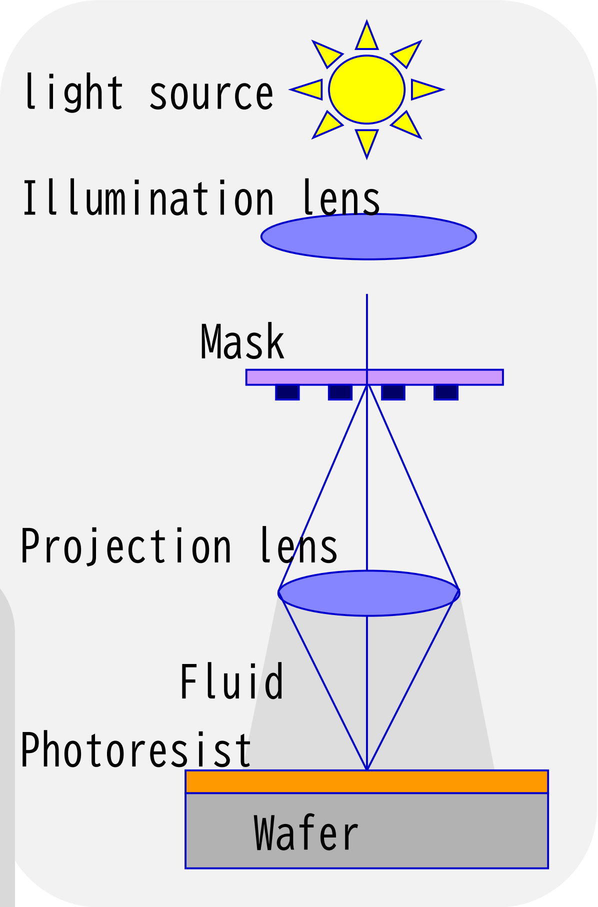
Atsushi Takahashi
Department of Information and Communications Engineering
School of Engineering
Tokyo Institute of Technology
atsushi@ict.e.titech.ac.jp

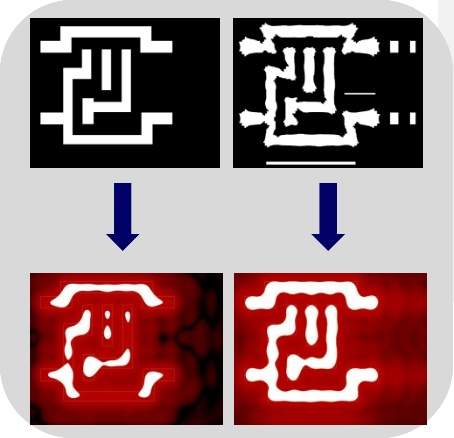
ICT.I419 VLSI Layout Design

VLSI Design / Manufacturing

Integration of Various Technologies

- Device Manufacture
  - Make transistors small
  - Mask Design, Exposure, Polishing, Dicing
- Circuit Design, Layout Design
  - High Speed, Low Power, Reliability
- Packaging, Printed Circuit Board
  - Wire Bonding
- System Design
- Software Design
- Marketing





VLSI Design (Synthesis)

- Design Automation
  - Essential in design productivity improvement
- Problem Definition
  - Inputs, outputs, and objectives
  - Design flow and Hierarchical synthesis
    - ✓ many sub-problems
  - ✓ Optimum solution for sub-problem may not be good for whole problem
  - Need to update Design methodology and Design flow

- Problem : Find an optimum solution
  - Is there an exact algorithm for the problem?
    - ✓ Yes (in most cases for combinatorial problem)
    - Enumerating all the cases and pick a best one
      - Impractical for large instances
  - Is there a practical exact algorithm for the problem?
    - ✓ NO (except limited cases)
    - Need sophisticated intelligent approach
      - Heuristic in most cases

How many seconds can you spend?

- 1 minute = 60 s = 6.0×10^{1} s
- 1 hour = 3,600 s = 3.6×10^{3} s
- 1 day = 86,400 s = 8.64×10^{4} s
- 1 month(30days) = 2,592,000 s = 2.592×10^{6} s
- 1 year(365days) = 31,536,000 s = 3.1536×10^{7} s
- 10 years = 315,360,000 s = 3.1536×10^{8} s
- 10 billion years = 3.1536×10^{17} s
- Age of the universe ≒ 13.8 billion years ≒ 4.35×10^{17} s

- $2^{60} \approx 1.15 \times 10^{18}$
- $20! \approx 2.43 \times 10^{18}$

P and NP

- Background of Algorithm Design
  - **P** and **NP**
  - **NP**-complete
  - **NP**-hard
  - Polynomial Time Reduction
  - Nondeterministic Polynomial Time Algorithm
  - (Deterministic) Polynomial Time Algorithm

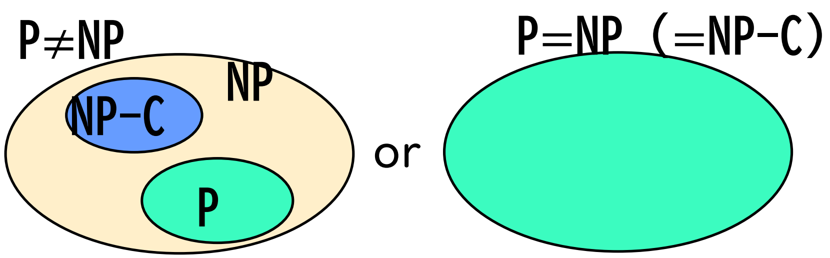
[Garey and Johnson, "Computers and Intractability, A Guide to the Theory of NP-Completeness", Freeman and Co., 1979]

P and NP

- Decision Problem (Yes/No Problem)
  - **NP** : Set of decision problems that have
    - Nondeterministic Polynomial time algorithm
  - **P** : Set of decision problems that have
    - (Deterministic) Polynomial time algorithm
  - **NP-C**: hardest problems in **NP**
    - If a problem in **NP-C** can be solved in polynomial time,
      then any problem in **NP** can be solved in polynomial time
    - A decision problem is said to be **NP**-complete if it is in **NP-C**
    - SAT, 3-SAT, COL, HG, IS, TS, …

- Theorem : P ⊆ NP

➢ Conjecture: P ≠ NP



Nondeterministic Polynomial Time Algorithm

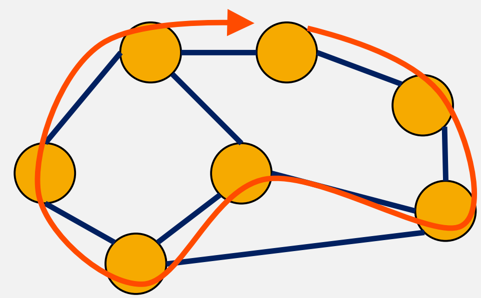
- Typical Structure
  - Step 1 (nondeterministic)
    - Generate an evidence in polynomial time
      (Pick up one arbitrary among exponential candidates)
  - Step 2 (deterministic)
    - Check the evidence in polynomial time
      - If the evidence is correct, then output YES
      - If the evidence is incorrect, then output NO

➢ Behavior of Correct Nondeterministic Algorithm

Correct Answer	Algorithm Output
YES	YES
No	No

Problem: Is Graph a Hamiltonian?

Evidence : sequence of vertices

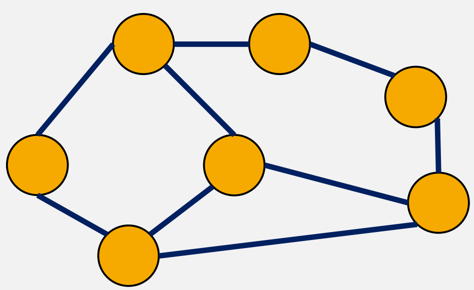


Evidence (Proof) for YES

- Problem: Is Graph Hamiltonian?
  - ✓ **NP**
    - An evidence that shows the graph is Hamiltonian which can be checked in polynomial time exists
- Problem: Is NOT Graph Hamiltonian?
  - ✓ ?? **NP** ?
    - An evidence that the graph is not Hamiltonian is not trivial
    - What is an evidence that shows the graph is not Hamiltonian?

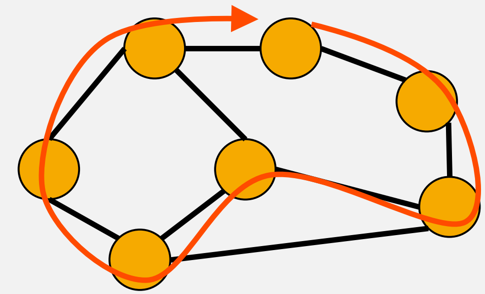
Problem: Is NOT Graph a Hamiltonian?

Evidence : ???



Problem: Is Graph a Hamiltonian?

Evidence : sequence of vertices

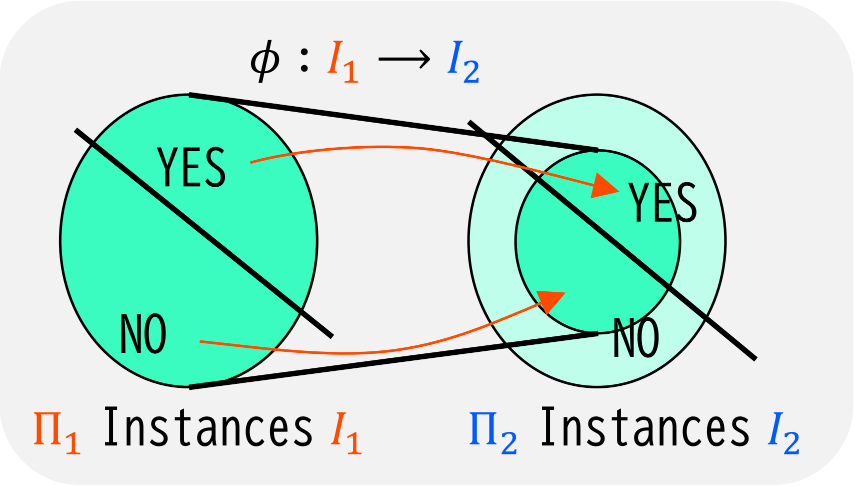


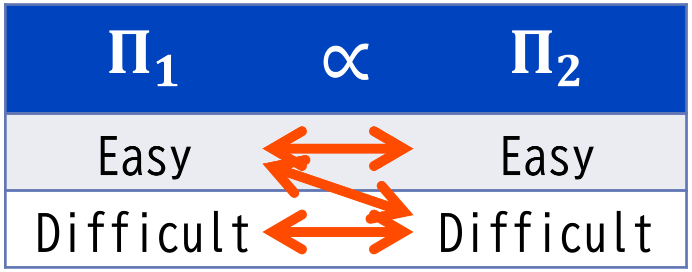
Polynomial Time Reduction ($\propto$)

- Provides difficulty relation between decision problems
  - Which is not difficult?
- Polynomial Time Recution of Problem ($\Pi_1 \propto \Pi_2$)
  - Instance of Π_1 can be converted to instance of Π_2 in polynomial time while maintaining Yes/No property
  - Problem Π_1 can be solved in polynomial time by utilizing (hypothetical) polynomial time algorithm for problem Π_2

✓ If Π_2 is solved in polynomial time, then Π_1 can be solved in polynomial time

  - Π_2 is not easier than Π_1 (same or more difficult)



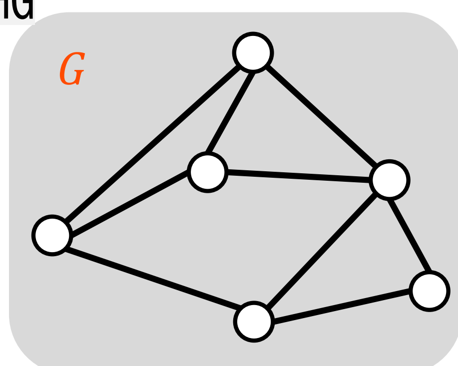


Example (HG $\propto$ TS)

- **Hamilton Graph Decision Problem (HG)**
  - INSTANCE : Graph G
  - QUESTION : Is G Hamiltonian?
- **Traveling Salesman Decision Problem (TS)**
  - INSTANCE : K_n, $w : E(K_n) \to \mathcal{R}^+$, r
  - QUESTION : Does Hamilton cycle C exist such that

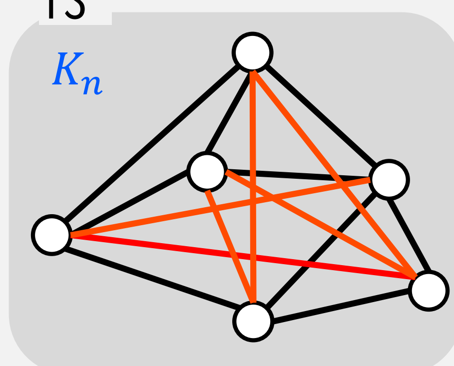
$$w(C) \le r, \quad C\ (\subseteq K_n)\ ?$$

HG



$n = |V(G)|$

TS



w

- 1 $(e \in E(G))$
- 2 $(e \notin E(G))$

$r := n$

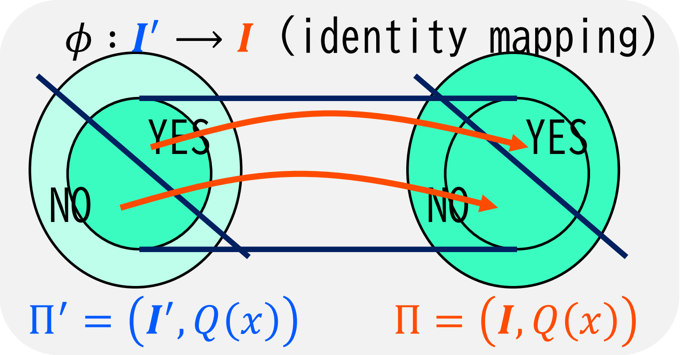
Hamilton cycle weight $\begin{cases} n & G \text{ edges only} \\ \ge n+1 & \text{contains other} \end{cases}$

G is Hamiltonian iff the minimum Hamiltonian cycle weight is n

Property of $\propto$

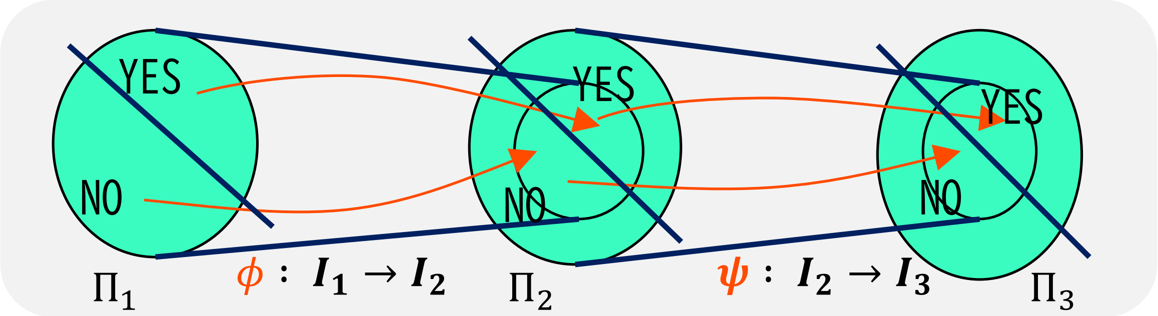
- **Theorem (subproblem)**:
  - Decision Problem $\Pi = (\boldsymbol{I}, Q(x))$
  - Subproblem $\Pi' = (\boldsymbol{I}', Q(x))$, $\boldsymbol{I}' \subseteq \boldsymbol{I}$

  ✓ $\Pi' \propto \Pi$



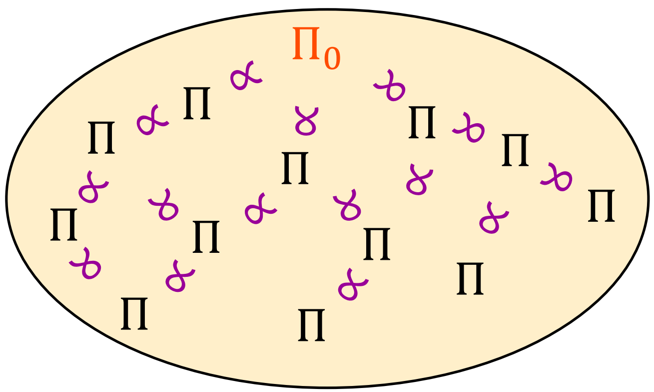
- **Theorem (transitivity)**: $\propto$ satisfies transitivity

  ✓ $\Pi_1 \propto \Pi_2, \Pi_2 \propto \Pi_3 \Rightarrow \Pi_1 \propto \Pi_3$
  - $\psi \circ \phi : \boldsymbol{I}_1 \rightarrow \boldsymbol{I}_3$



NP-complete Problem

- **NP-complete** problem Π_0: $^{\forall}\Pi \in \mathbf{NP}$, $\Pi \propto \Pi_0$
  - Not easier than any problem in **NP**

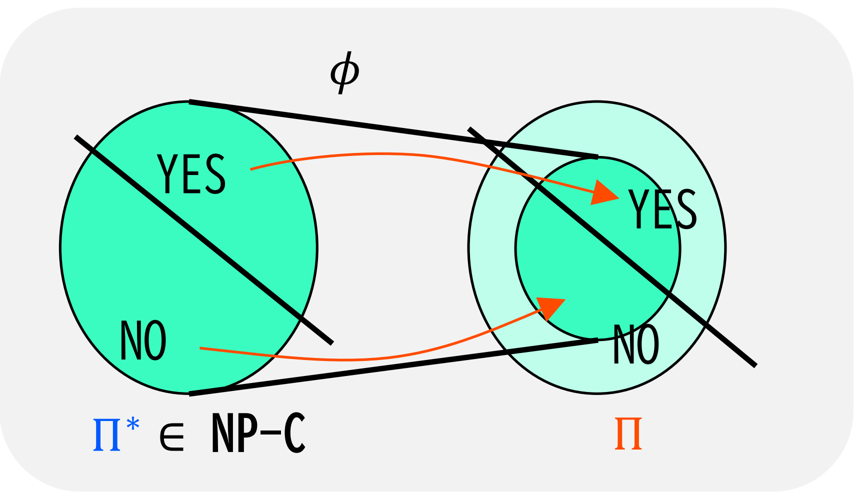


Π	$\propto$	Π_0
Easy		Easy
Difficult		Difficult

  - No polynomial time algorithm if **P** $\neq$ **NP**
    - We will give up to design efficient algorithm
      - Approximation algorithm
      - Heuristic algorithm

Typical Proof of NP-completeness

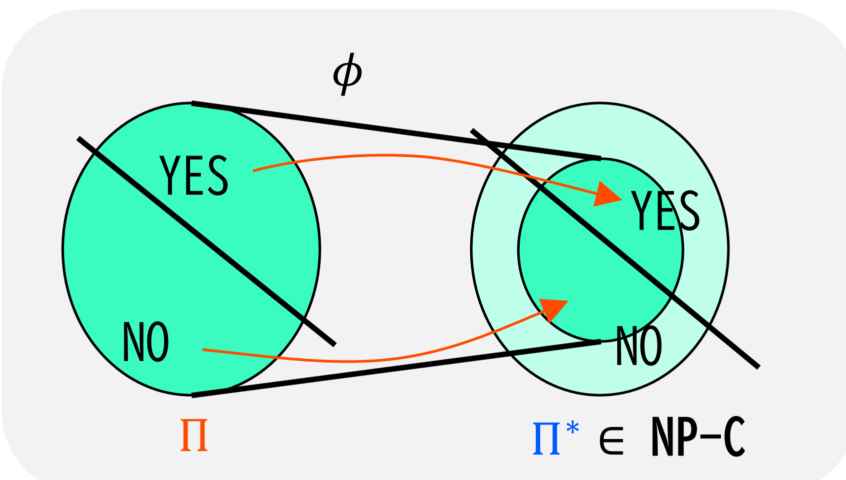
- **Theorem** : Π is **NP**-complete if
  1. $\Pi \in \mathbf{NP}$
  2. $\Pi^* \propto \Pi$ for some **NP**-complete problem Π^*
- Proof
  - $\forall \Pi' \in \mathbf{NP}, \Pi' \propto \Pi^*$ and $\Pi^* \propto \Pi \Rightarrow \forall \Pi' \in \mathbf{NP}, \Pi' \propto \Pi$



Π^*	$\propto$	Π
Easy		Easy
Difficult		Difficult

Incorrect Proof of NP-completeness

- Incorrect proof of **NP**-completeness of Π
  1. $\Pi \in$ **NP**
  2. Pick up **NP**-complete problem Π^*
- ✓ Show $\Pi \propto \Pi^*$
  - It is trivial by definition
  - It does not mean that Π is **NP**-complete



Π	$\propto$	Π^*
Easy		Easy
Difficult		Difficult

Boolean Logic

- Boolean variable
  - $a, b \in \mathbb{B} = \{0, 1\} = \{$ False, True $\}$
- Unary operator
  - $\neg$: NOT
- Binary operator
  - $\wedge$: AND, $\vee$: OR
- Truth Table

a	$\neg a$
0	1
1	0

a	b	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

a	b	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

a	b	c	$a \vee b$	$(a \vee b) \wedge c$
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

SATISFIABILITY (SAT)

- Satisfiability (SAT)
  - INSTANCE : Boolean formula F in CNF
    - CNF
      - Conjunctive Normal Form, Product of sums, NOT-OR-AND
  - QUESTION : Is F satisfiable?

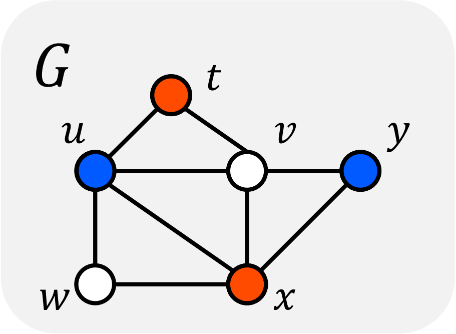
- Example
  - $F = (a \vee b) \wedge (\neg a \vee \neg b \vee c) \wedge (\neg a \vee \neg c)$
    - F is satisfiable
    - $a = 1, b = 0, c = 0 \implies F = 1$
  - $F = (a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b)$
    - F is unsatisfiable

SAT is **NP**-complete

- Theorem : SAT is **NP**-complete
  - SAT is in **NP**
  - Turing Machine behavior is modeled by polynomial size Boolean formula
  - ✓ SAT is a hardest decision problem

COLORING (COL)

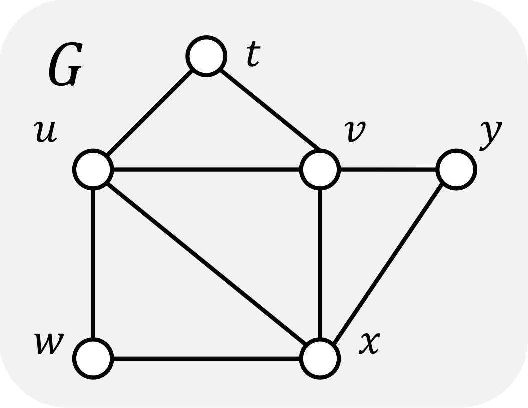
- 3-COLORING (3-COL)
  - INSTANCE
    - Graph G
  - QUESTION
    - Can G be colored with 3 colors?



- Coloring of a graph
  - a coloring of the vertices of the graph such that no two adjacent vertices have the same color

Example (3-COL $\propto$ SAT)

- **Example**: 3-COL $\propto$ SAT (cont.)
  - Certificate for 3-coloring
    - coloring of a graph $G = (V, E)$ with three colors
      - vertices are colored with three colors
      - a vertex is colored by one color
      - any two adjacent vertices are colored by different colors
  - $V(G) = \{v_1, v_2, \ldots, v_n\}$, $E(G) = \{e_1, e_2, \ldots, e_m\}$
  - three Boolean variables x_{i1}, x_{i2}, x_{i3} for each vertex v_i
    - $x_{ij} = \begin{cases} 1 \text{ if } v_i \text{ is color } j \\ 0 \text{ otheriwise} \end{cases}$



Example (3-COL$\propto$ SAT)

- Example: 3-COL $\propto$ SAT (cont.)
  - vertex v_i is colored by a color in colors 1, 2, 3
    $P_i = (x_{i1} \vee x_{i2} \vee x_{i3}) \wedge (\overline{x_{i1}} \vee \overline{x_{i2}}) \wedge (\overline{x_{i1}} \vee \overline{x_{i3}}) \wedge (\overline{x_{i2}} \vee \overline{x_{i3}})$
  - every vertex is colored by one color

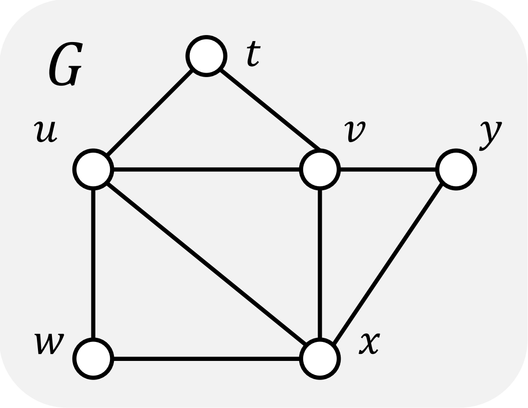
$$\bigwedge_{i \in V(G)} P_i$$

  - adjacent vertices v_i and v_j are colored by different colors

$$Q_{(i,j)} = (\overline{x_{i1}} \vee \overline{x_{j1}}) \wedge (\overline{x_{i2}} \vee \overline{x_{j2}}) \wedge (\overline{x_{i3}} \vee \overline{x_{j3}})$$

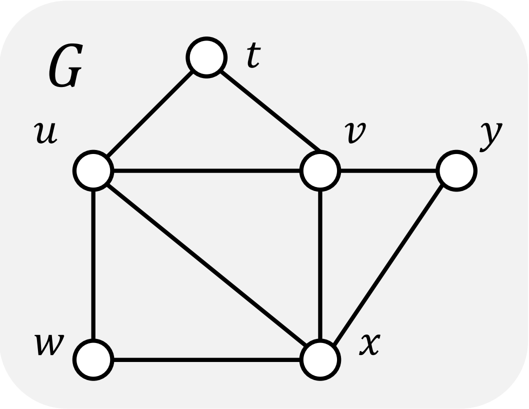
  - every two adjacent vertices are colored by different colors

$$\bigwedge_{(i,j) \in E(G)} Q_{(i,j)}$$



Example (3-COL ∝ SAT)

- **Example**: 3-COL $\propto$ SAT (cont.)
  - G can be colored by three colors
  - $f(G) = \left(\bigwedge_{i \in V(G)} P_i\right) \wedge \left(\bigwedge_{(i,j) \in E(G)} Q_{(i,j)}\right)$ is satisfiable
  - $\phi : G \mapsto f(G)$
    - polynomial time reduction from 3-COLORING to SAT



NP-Completeness (3-SAT)

- 3-SAT
  - INSTANCE : Boolean formula F in CNF with three literals per clause
  - QUESTION : Is F satisfiable?

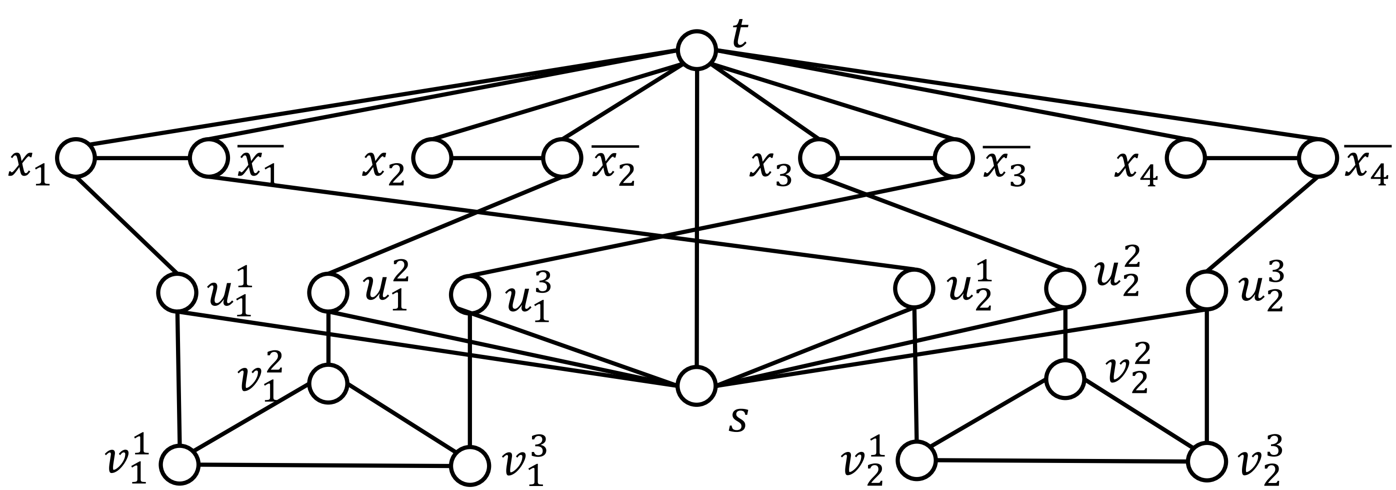
- Example
  - $F = (a \vee b \vee c) \wedge (\neg a \vee \neg b \vee c) \wedge (a \vee \neg c \vee \neg d)$

NP-Completeness (3-SAT)

- Theorem : 3-SAT is **NP**-complete
- Proof
  - By showing that SAT $\propto$ 3-SAT
  - Polynomial time reduction from 3-SAT to SAT
    - Prepare y literals not in SAT formula F
  - One literal clause of F
    - $(x) \Longrightarrow (x \vee y_1 \vee y_2) \wedge (x \vee y_1 \vee \overline{y_2}) \wedge (x \vee \overline{y_1} \vee y_2) \wedge (x \vee \overline{y_1} \vee \overline{y_2})$
  - Two literals clause of F
    - $(x_1 \vee x_2) \Longrightarrow (x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \bar{y})$
  - k literals clause of F $(k \geq 4)$
    - $(x_1 \vee x_2 \vee \cdots \vee x_k) \Longrightarrow (x_1 \vee x_2 \vee y_1) \wedge (\overline{y_1} \vee x_3 \vee y_2) \wedge$
      $(\overline{y_2} \vee x_4 \vee y_3) \wedge \cdots \wedge (\overline{y_{k-3}} \vee x_{k-1} \vee x_k)$
  - The size of obtained 3-SAT formula is polynomial of $|F|$

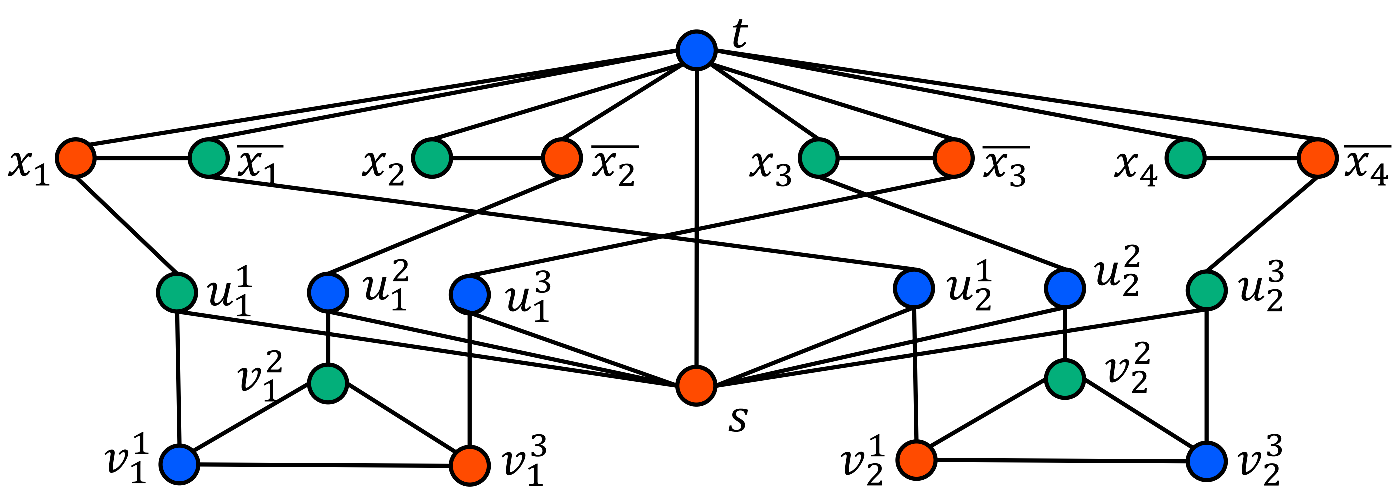
NP-Completeness (3-COL)

- Theorem : 3-COLORING is **NP**-complete
- Proof
  - By showing that 3-SAT $\propto$ 3-COLORING
    - Graph $G(f)$ corresponding to $f = (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4})$



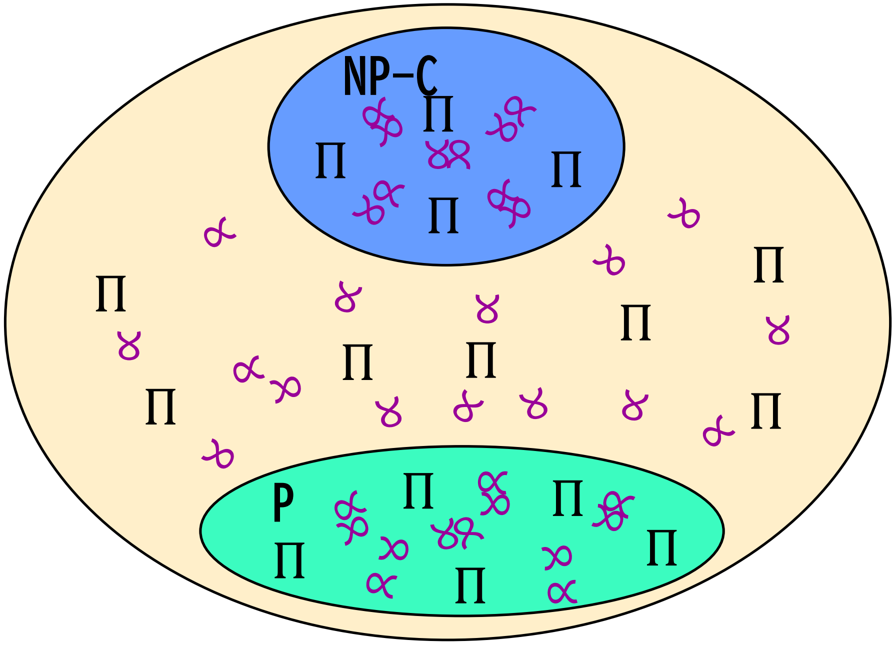
NP-Completeness (3-COL)

- Theorem : 3-COLORING is NP-complete
- Proof (cont.)
  - Coloring of $G(f)$ corresponding to the assignment
    - $x_1 = 1, x_2 = x_3 = x_4 = 0$
  - $\phi : f \mapsto G(f)$
    - polynomial time reduction from 3-SAT to 3-COLORING



Property of **NP**-completeness

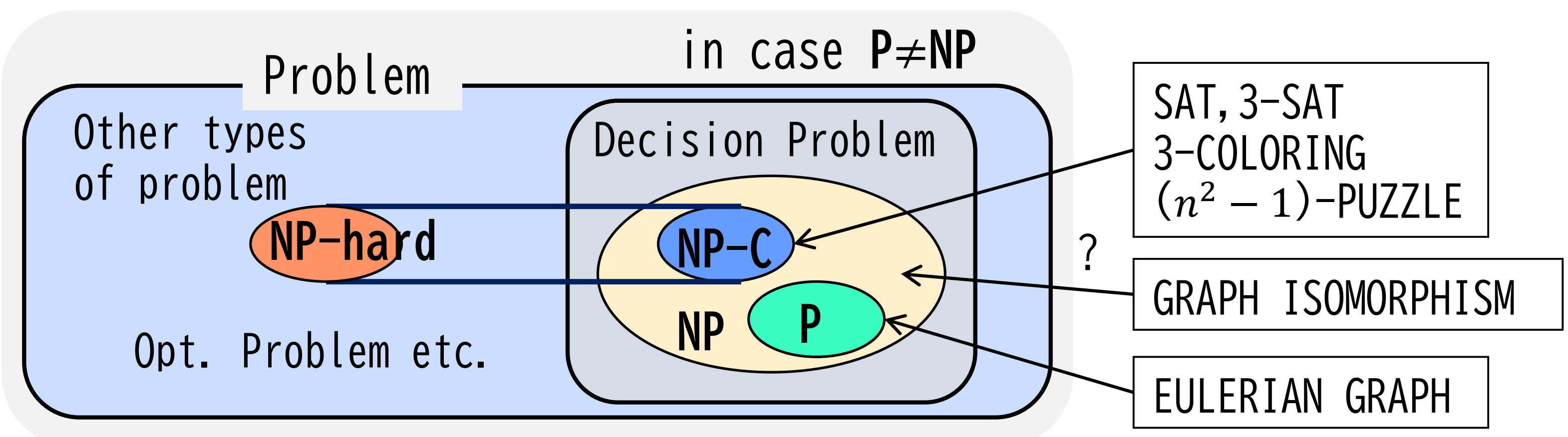
- Theorem 9.9 :
  - HG is **NP**-complete
  - TS is **NP**-complete
    - T-TS is **NP**-complete
    - MAX-TS is **NP**-complete
  - 3-SAT is **NP**-complete
  - 3-COL is **NP**-complete
  - …



in case **P**≠**NP**

NP-hardness

- A problem is NP-hard if the decision problem associated with the problem is NP-complete
- Optimization problem is
  - neither in **NP** nor in **NP-C**
  - not said to be NP-complete
  - said to be NP-hard if a related decision problem is NP-complete
- If **P≠NP**, then
  - No polynomial time algorithm for NP-hard problem
- If a problem is NP-hard, then
  - Approximation algorithm or Heuristic algorithm are pursued



NP-hardness

- Dealing with **NP**-hard problems
  - Subproblem
  - Approximation algorithm
  - Randomized algorithm
  - Heuristic algorithm

- Open Problem
  - Clay Mathematics Institute
  - Millennium Problems
  - **P** vs. **NP** (**P**=**NP**?)
  - http://www.claymath.org/millennium-problems/p-vs-np-problem

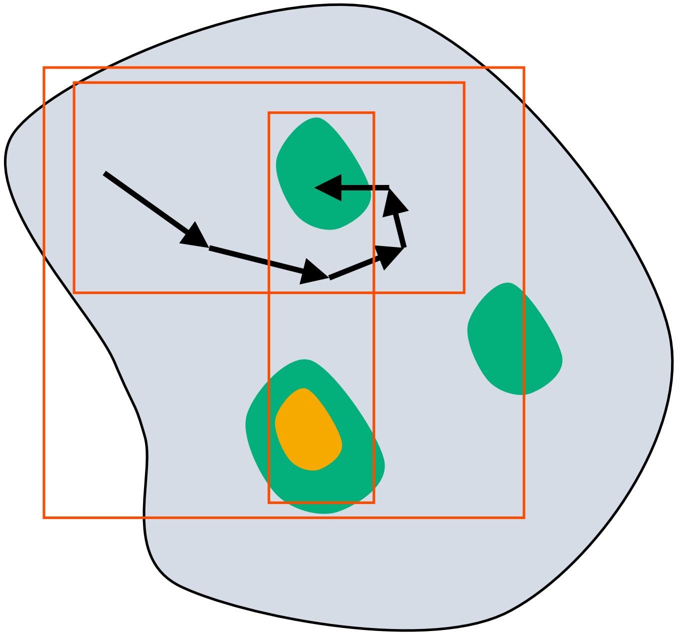
First Step of Algorithm Design

- Check whether problem is easy or not?
  - ✓ Assuming **P** $\neq$ **NP**
  - Difficult = **NP**-hard, **NP**-complete
    - Design heuristic
  - Easy = **P** (or decision version is in **P**)
    - Design exact polynomial time algorithm
    - Reduce time and space complexity
- Most of practical problems are difficult
  - **NP**-hardness seems trivial ....
    but proof of **NP**-hardness is not easy
  - So, proof is often skipped, recently

- In the following
  - **P** = problem solvable in polynomial time

Exploration of Solution Space

- Exploration of Huge Design Space
- Increase of computation power enable us to use computation power rich algorithms
  - Iterative improvement
  - Stochastic search
  - Analytical method
- Solution space design
  - Abandon useless area
  - Focus on promising area
  - Efficiency



Automated vs. Manual

- Good tools have been developed so far
- For large chips
  - Huge number of nets and enough resources
  - Looser constraint
  - Too many nets to design manually
  - Lower quality is affordable
  - Tools are essential in recent design
- For small chips, IoT devices, and etc.
  - Medium number of nets and limited resources
  - Tighter constraint
  - Time consuming, but designer can handle
  - Higher quality is essential
  - Automated Tools are still not popular in high-end designs