INT02
Algorithmic Foundations And Problem Solving
The Limitations of Algorithm Power
-- Introduction to Computational Complexity Theory

Dr Jia WANG
Department of Computer Science

西交利物浦大学
Xi'an Jiaotong-Liverpool University

Acknowledgment: The slides are adapted from ones by Dr. Prudence Wong

Finding Critical Users in Social Communities: The Collapsed Core and Truss Problems

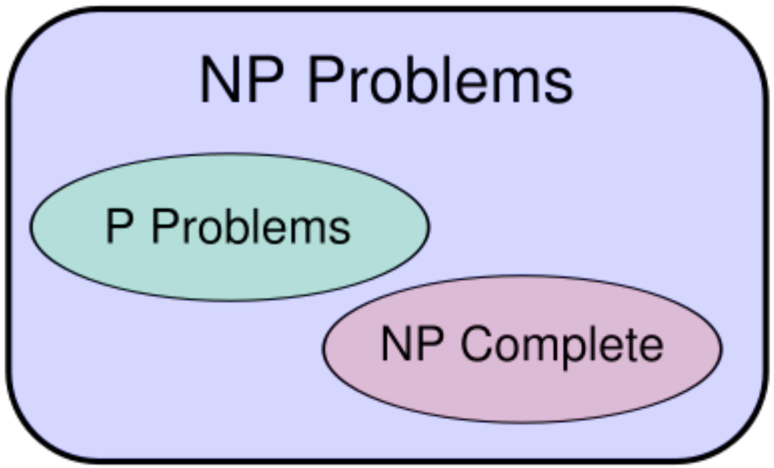
November 2018 · IEEE Transactions on Knowledge and Data Engineering PP(99):1-1 ·
Follow journal
DOI: 10.1109/TKDE.2018.2880976

Fan Zhang · Conggai Li · Ying Zhang · Show all 5 authors · Wenjie Zhang

Abstract

In social networks, the leave of critical users may significantly break network engagement, i.e., lead a large number of other users to drop out. A popular model to measure social network engagement is k-core, the maximal subgraph in which every vertex has at least k neighbors. To identify critical users, we propose the collapsed k-core problem: given a graph G, a positive integer k and a budget b, we aim to find b vertices in G such that the deletion of the b vertices leads to the smallest k-core. We prove the problem is NP-hard and inapproximate. An efficient algorithm is proposed, which significantly reduces the number of candidate vertices. We also study the user leave towards the model of k-truss which further considers tie strength by conducting additional computation w.r.t. k-core. We prove the corresponding collapsed k-truss problem is also NP-hard and inapproximate. An efficient algorithm is proposed to solve the problem. The advantages and disadvantages of the two proposed models are experimentally compared. Comprehensive experiments on 9 real-life social networks demonstrate the effectiveness and efficiency of our proposed methods.

Hard Computational Problems



CSAT: Circuit-SAT
HCP: Hamiltonian circuit problem
TSP: Traveling Salesman Problem

Complexity Classes P and NP

The complexity class P is the set of all decision problems that can be solved in worst-case polynomial time.

The complexity class NP is the set of all problems that can be verified in polynomial time.

P stands for polynomial, and
NP stands for non-deterministic polynomial.

The Class P

MST problem is in P

- Given a weighted graph G and a value k, does there exists an MST with weight at most k?
- run Kruskal's algorithm (polynomial time) and if the MST found has weight at most k, then the answer is "Yes"
- Kruskal's Algorithm in O(ElogE)

Single-source-shortest-paths problem is in P

- Given a weighted graph G, a source vertex s, and a value k, does there exist shortest paths from s to every other vertex whose path length is at most k?
- run Dijkstra's algorithm (polynomial time) and if the paths found have lengths at most k, then answer is "Yes"

The Class NP

Hamiltonian circuit problem is in NP

- we can check in polynomial time if a proposed circuit is a Hamiltonian circuit

0/1 Knapsack problem is in NP

- we can check in polynomial time if a proposed subset of items whose weight is at most W and whose value is at least k

Circuit-SAT is in NP

- we can check in polynomial time if proposed values lead to a final output value of 1

P = NP ?

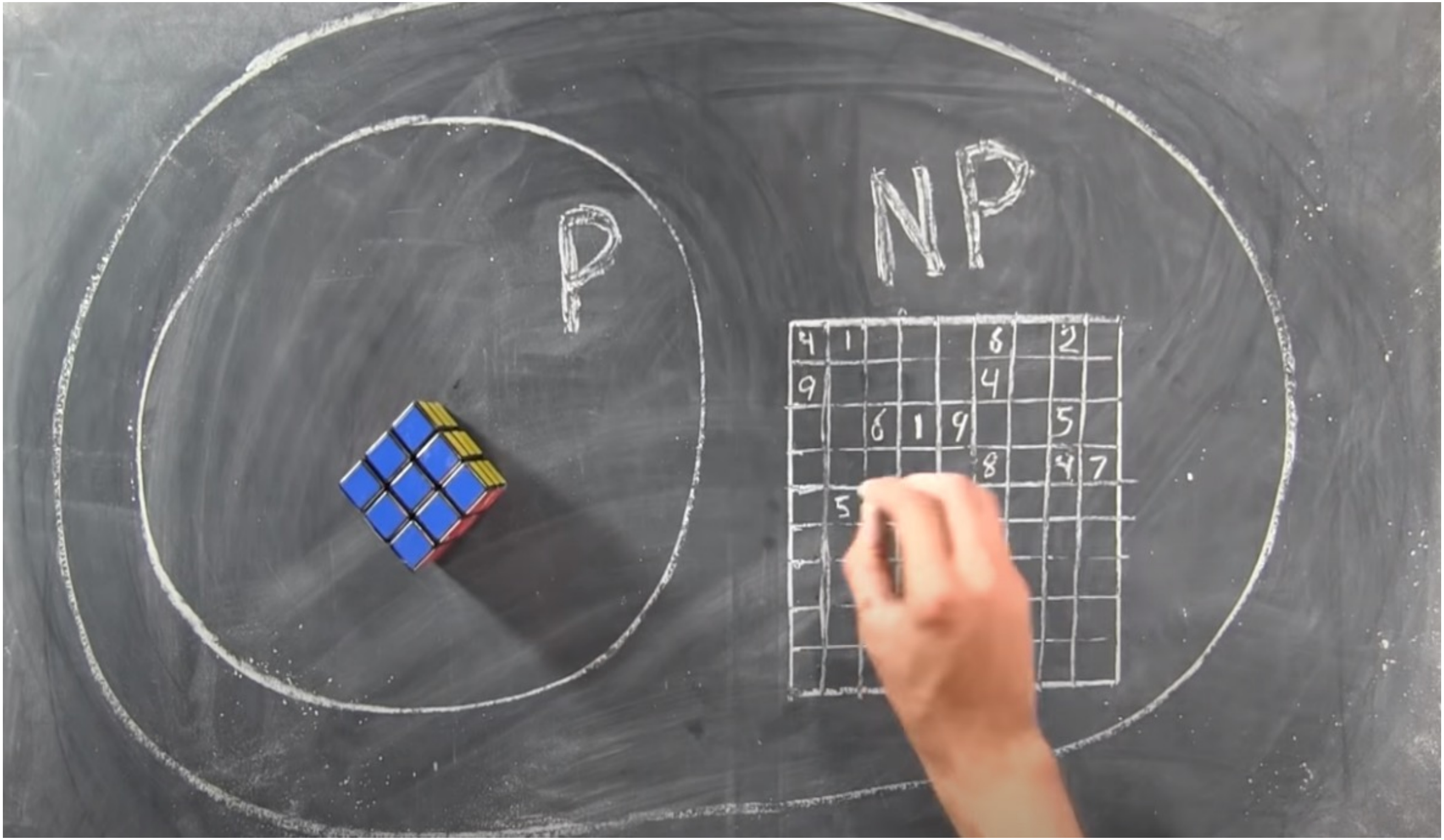
Note that $P \subseteq NP$

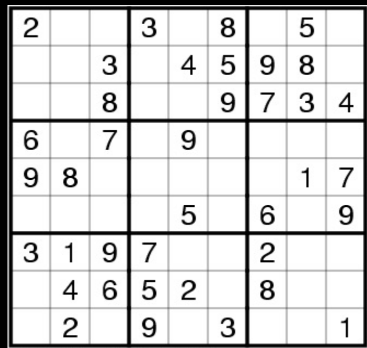
The (million dollar) question is that mathematicians and computer scientists do not know whether $P = NP$ or $P \neq NP$

However, there is a common belief that P is different from NP

- i.e., there is some problem in NP that is not in P

P vs. NP





n x n x n

Sudoku

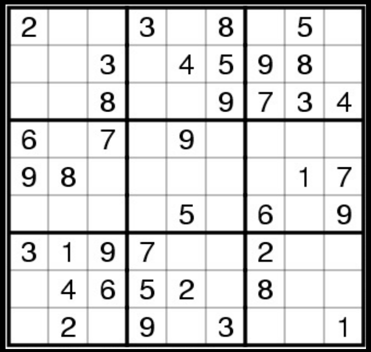
Suppose it takes you S(n) to solve n x n x n

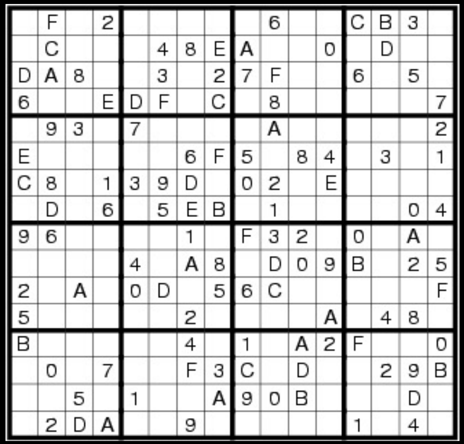
V(n) time to verify the solution

Fact: $V(n) = O(n^2 \times n^2)$

Question: is there some constant such that $S(n) \leq n^{\text{constant}}$?

Sudoku





⋮

n x n x n

P vs NP problem

=

Does there exist an algorithm for n x n x n Sudoku that runs in time p(n) for some polynomial p( ) ?

What is an efficient algorithm?

Is an $O(n)$ algorithm efficient?

How about $O(n \log n)$?

$O(n^2)$?

$O(n^{10})$?

$O(n^{\log n})$?

$O(2^n)$?

$O(n!)$?

polynomial time

$O(n^c)$ for some constant c

non-polynomial time

Polynomial-time reduction (prove hardness)

Given any two decision problems A and B, we say that

(1) A is <u>polynomial time reducible</u> to B, or

(2) there is a polynomial time reduction from A to B

if given any input α of A, we can **construct** in <u>polynomial time</u> an input β of B such that

α is yes **if and only if** β is yes.

We use the notation $A \leq_P B$

Intuitively, this means that problem <u>**B**</u> is at least as difficult as problem <u>**A**</u>

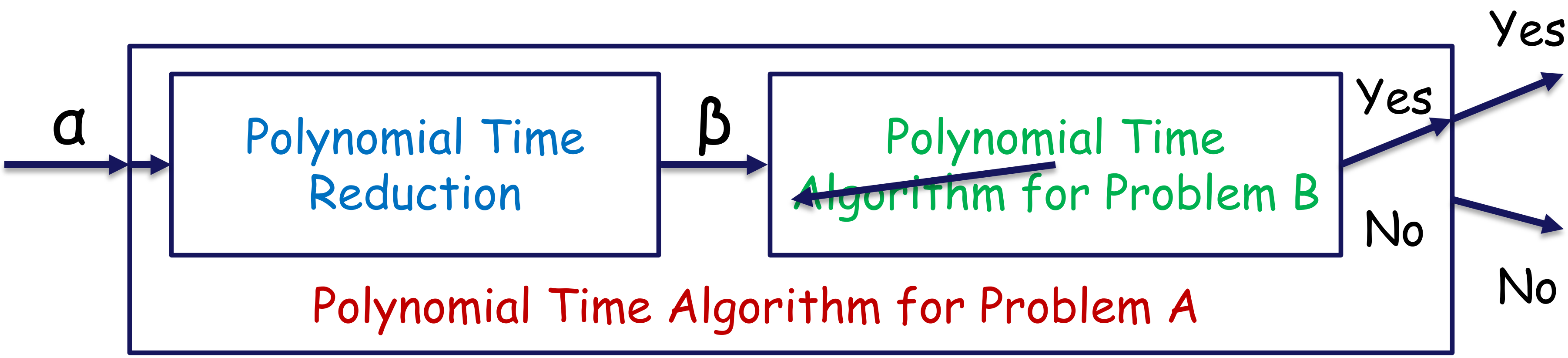
Hamiltonian Cycle Problem (A)

Problem Statement: Given a graph G=(V,E)G=(V,E), is there a cycle that visits every vertex exactly once and returns to the starting vertex?

Traveling Salesman Problem (B)

Problem Statement: Given a set of cities (vertices), distances between each pair of cities (edges with weights), and a limit k on the total distance, is there a route that visits each city exactly once and returns to the starting city with a total travel distance less than or equal to k?

Polynomial-time reduction



If problem A is **reducible** to problem B in polynomial time, then which problem is easier?

A ✓ Or B ?

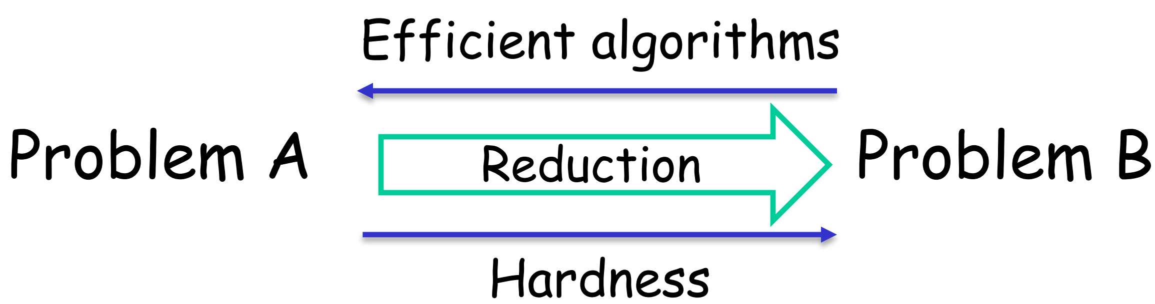
Problem B is at least **as hard as** Problem A!

Two Ways to Use Reductions

Suppose Problem A is reducible to Problem B

- Solve problem
  - ➢ If there exists **efficient algorithm** for Problem B, then we can solve Problem A efficiently

- Prove Hardness
  - ➢ If Problem A is hard, then Problem B is **also hard**



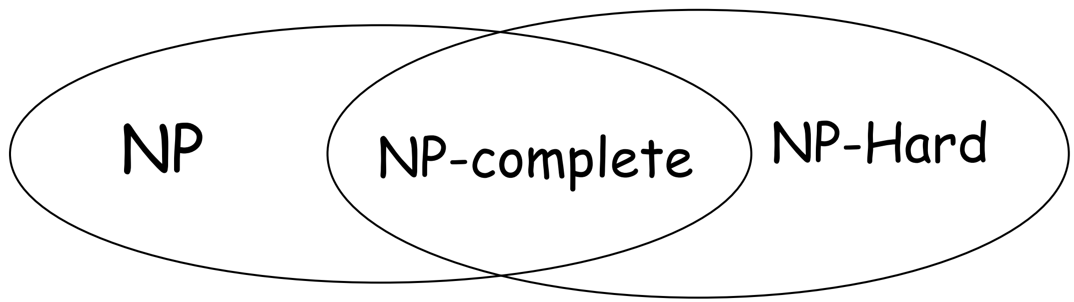
NP-hardness

A problem M is said to be **NP-hard** if every other problem in NP is polynomial time reducible to M

- intuitively, this means that M is at least as difficult as all problems in NP

M is further said to be NP-complete if

1. M is in NP, and
2. M is NP-hard



NP-complete problems are some of the hardest problems in NP

NP-Completeness

Problem A is NP-complete if

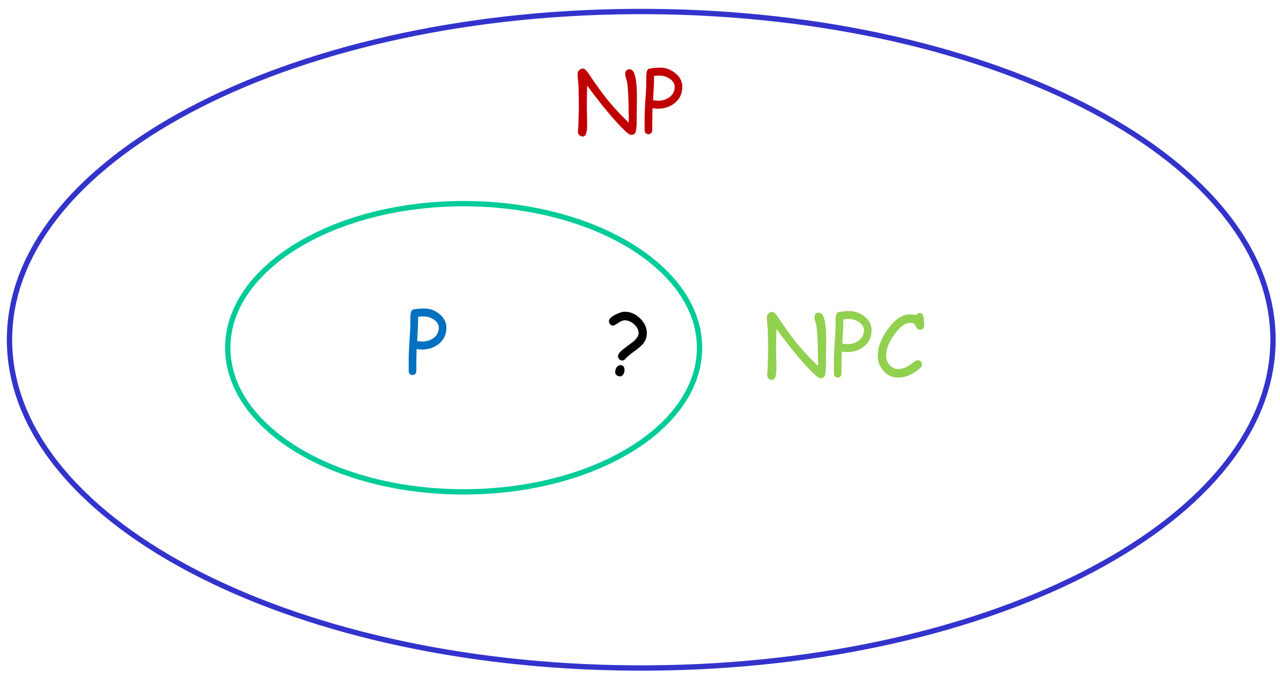
1. Problem A is **in NP**

2. For **any** Problem A' in NP, A' is **reducible** to A in polynomial time

- Class NPC: The class of all NP-complete problems, which is a subclass of NP
  - The **hardest** problems in NP
  - Solve **a** problem in NPC, you can solve **ALL** problems in NP

NP = P ?

If NPC $\cap$ P is not empty, then NP = P



NP-Complete Problem

The **Cook-Levin Theorem** states that Circuit-SAT is NP-complete (a "first" NP-complete problem)

Using polynomial time reducibility we can show existence of other NP-complete problems

A useful result to prove NP-completeness:

Lemma

If L1 $\leq_P$ L2 and L2 $\leq_P$ L3, then L1 $\leq_P$ L3

Proof of NP-Completeness

Given a Problem *A*, prove that *A* is NP-complete

Proof Scheme 1

- Show Problem *A* is in **NP** (easier part)
- For all Problems in **NP**, reduce them to *A* in **polynomial time**
  - ❖ This has been done for 3SAT, the first NP-complete problem

Proof Scheme 2

- Show Problem *A* is in **NP** (easier part)
- For arbitrary problem *A'* in NPC, reduce *A'* to *A* in **polynomial time**
  - ❖ This would be much easier

Other NP-Complete Problems

We have seen these NP-Complete Problems

- Hamiltonian Circuit Problem
- 0/1 Knapsack Problem
- Circuit-SAT

Others

- CNF-SAT and 3-SAT (conjunctive normal form satisfiability problem)
- Vertex Cover

Conjunctive normal form (CNF)

- a Boolean formula is in CNF if it is formed as a collection of clauses combined using the operator AND (·) and each clause is formed by literals (variables or their negations) combined using the operator OR (+)
- example: ***(x1+x2+x4+x5) · (x2+x1+x4)***

CNF-SAT and 3-SAT

CNF-SAT

- **Input:** a Boolean formula in CNF
- **Question:** Is there an assignment of Boolean values to its variables so that the formula evaluates to **true**? (i.e., the formula is satisfiable)

3-SAT

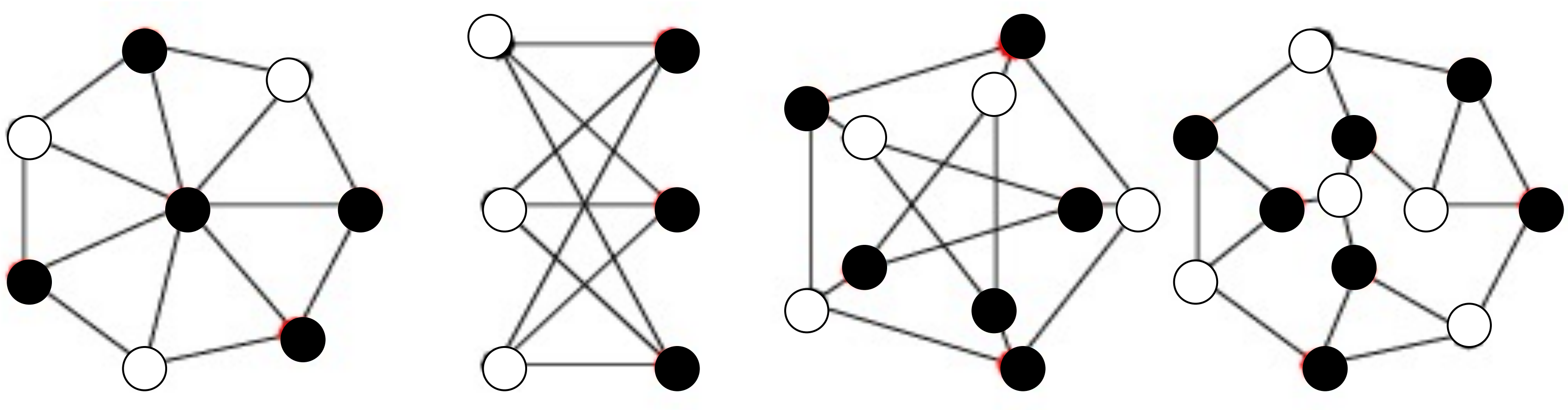
- **Input:** a Boolean formula in CNF in which each clause has exactly 3 literals

CNF-SAT and 3-SAT are NP-complete

Vertex Cover

Given a graph $G = (V,E)$

A **vertex cover** is a subset $C \subseteq V$ such that for every edge (v,w) in E, **$v \in C$ or $w \in C$**



some graphs and their vertex cover (shaded vertices)

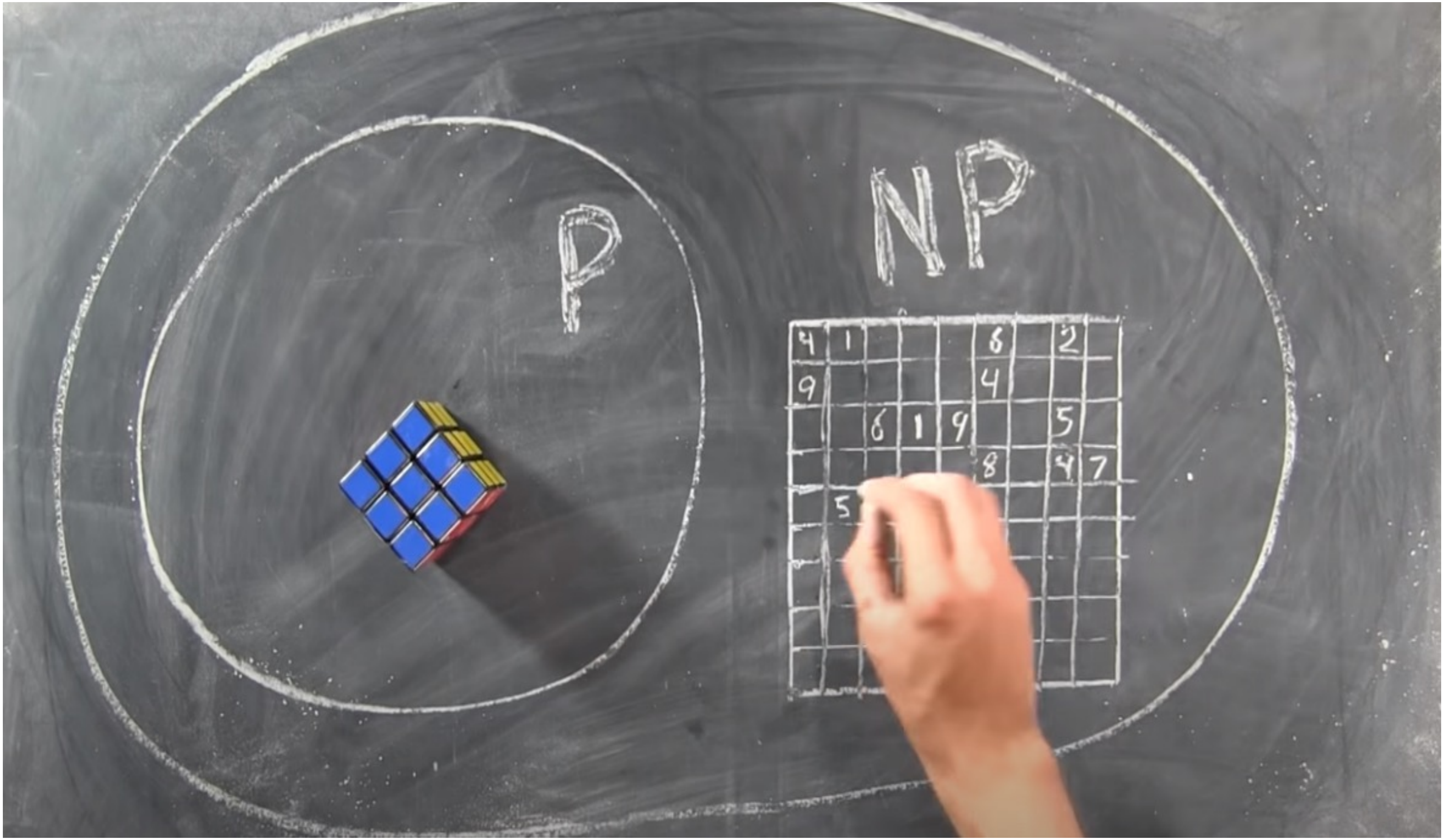
Vertex Cover

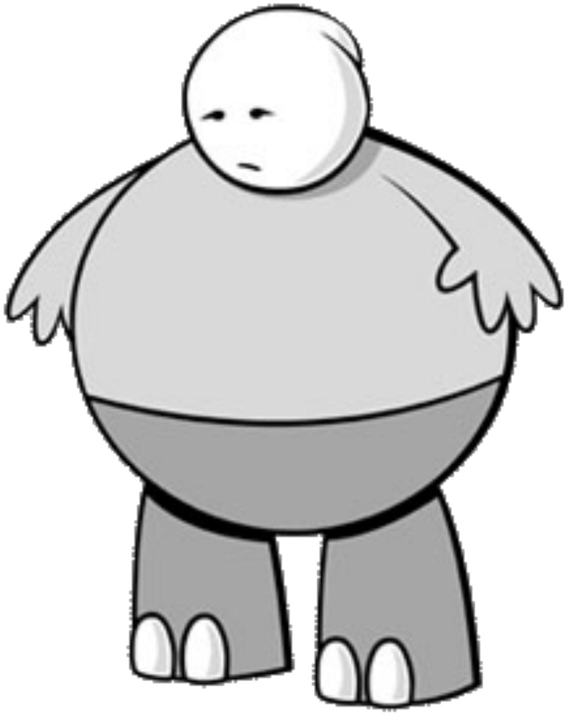
The optimisation problem is to find as **small** a vertex cover as possible

Vertex Cover is the **decision** problem that takes a graph **G** and an integer **k** and asks whether there is a vertex cover for G containing **at most** k vertices

Vertex Cover is NP-complete

Summary





Here's What You Need to Know…

Definition of P and NP

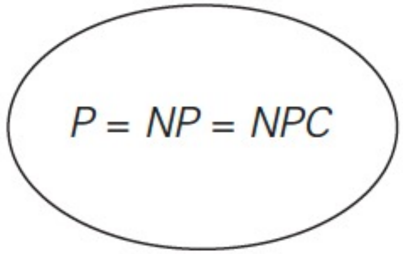
Definition of NPC and NP-hard

Examples of NP-Complete problems

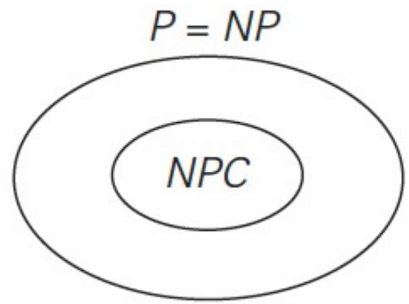
P, NP, NPC, NP-hard

Solve any one in poly-time
$\Rightarrow$ solve <u>all</u> of them in poly-time

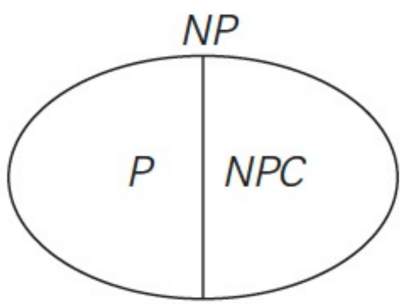
a.



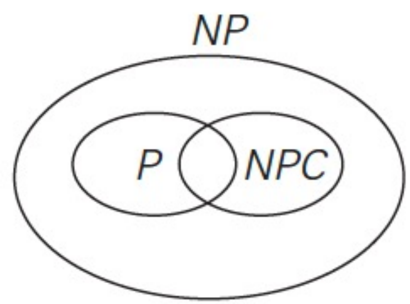
b.



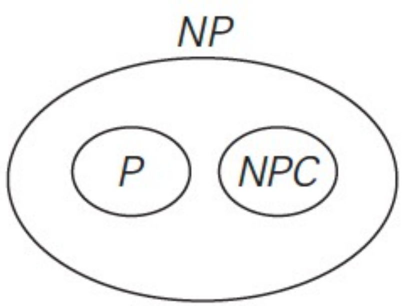
c.



d.



e.



current state of our knowledge

1.Diagram a: This diagram suggests that P, NP, and NPC are all equivalent, which implies every P problem is NP-complete, and every NP problem is in P. This is contrary to current understanding and highly speculative without concrete proof. **This diagram contradicts known complexity theory.**

1.Diagram b: Here, P equals NP, but NP-complete (NPC) problems form a proper subset of NP (and thus P as well). This implies all NP problems, including NP-complete problems, are solvable in polynomial time. While this diagram does reflect a scenario where P = NP, it suggests that not all NP problems are NP-complete, which is accurate if P = NP. **This diagram does not contradict known theory**, though it depicts an unresolved scenario in complexity theory.

2.Diagram c: This diagram presents P and NP as disjoint sets, which is incorrect. It is known that all problems in P are also in NP (P is a subset of NP). **This diagram contradicts known complexity theory.**

3.Diagram d: This diagram shows P as a subset of NP, and NPC as a subset of NP with an intersection between P and NPC. This implies some P problems are NP-complete, which contradicts the definition of NP-completeness (NP-complete problems are the hardest problems in NP, and if any of them were solvable in polynomial time, then P = NP). **This diagram contradicts known complexity theory.**

4.Diagram e: This diagram correctly shows P as a subset of NP, with NP-complete problems also being a subset of NP. P and NPC do not overlap, meaning no problem in P is NP-complete unless P = NP, which is consistent with current knowledge since we do not yet know if P = NP or not. **This diagram does not contradict known complexity theory.**

Prove: 3SAT is NP-hard

optional

Reduce CNF-SAT to 3SAT

- Let E be an arbitrary instance of SAT.
- We will replace each clause of E with several clauses, each has exactly three variables
- Target: to prove that the clause of E is satisfiable if and only if all the transformed clauses are satisfied

optional

How It Works

For example, consider the clause $(x \vee y \vee z \vee w)$. This clause has four literals, but we need to express it in a form where each clause contains exactly three literals. Here's how the new variables $u1$ and $u2$ are used:

- **First Clause**: $(x \vee y \vee u1)$
- **Second Clause**: $(\neg u1 \vee z \vee u2)$
- **Third Clause**: $(\neg u2 \vee w \vee v)$

optional

Imagine you have a group project with four tasks (x, y, z, w) and four team members need to choose tasks such that the project is completed. The rule is that each member can only sign up in groups of three for a discussion round, and you need to make sure everyone agrees on who does what without leaving any task out.

- **First Meeting (First Clause: $\mathbf{1x \vee y \vee u1}$)**:
  - In the first discussion round, three members meet and discuss tasks x, y, and $u1$. Here, $u1$ doesn't represent a real task. It's a placeholder or a "promise" that helps connect this group's decision with the next group. They decide if either x or y can be done, or they might pass the decision to the next group using $u1$ as a "maybe" flag.

optional

- **Second Meeting (Second Clause: ¬u1∨z∨u2)**:
  - A different group of three meets next. They know that if the previous group left a "maybe" (i.e., $u1$ is true), they now need to ensure task z gets done, or they can again pass a "maybe" to the next group using a new placeholder, $u2$. The negation $\neg u1$checks if the first group passed on the decision (didn't conclusively decide x or y), making it necessary for this group to handle it.

- **Third Meeting (Third Clause: ¬u2∨w∨v)**:
  - The final group sees if the "maybe" flag $u2$ was passed along. If it was (meaning task z wasn't definitively handled), they must now take care of ww and possibly handle another task v (which is another placeholder if needed).

Prove: 3SAT is NP-hard (2)

optional

Suppose $C = (x_1+x_2+x_3+\ldots+x_k)$ be a clause of E such that $k \geq 4$.

We introduce new variables $y_1, y_2, y_3, \ldots, y_{k-3}$ to form C'

$$C' = (x_1+x_2+y_1) \cdot (x_3+\overline{y_1}+y_2) \cdot (x_4+\overline{y_2}+y_3) \cdot \cdot \cdot (x_{k-2}+\overline{y_{k-4}}+y_{k-3}) \cdot (x_{k-1}+x_k+\overline{y_{k-3}})$$

Target: Prove C' is satisfiable if and only if C is satisfiable

Prove: 3SAT is NP-hard (3)

optional

If C is satisfiable

- one of the x_i must be set to 1
- if $x_3=1$, we set $y_1=1$, $y_2=0$, and the rest of $y_i=0$
  - $y_1=1$ implies $(x_1+x_2+y_1)$ is 1; $y_2=0$ & $x_3=1$ implies $(x_3+\cancel{y_1}+y_2)$ is 1; rest of $y_i=0$ implies all other clauses are 1
- if $x_i=1$, we set $y_1,y_2,\ldots,y_{i-2} = 1$, the rest $= 0$
  - $y_1,y_2,\ldots,y_{i-2} = 1$ implies the first (i-2) clauses are 1; $x_i=1$ implies $(x_i+\overline{y_{i-2}}+y_{i-1})$ is 1; the rest = 0 implies the remaining clauses are 1

$$C = (x_1+x_2+x_3+\ldots+x_k)$$

$$C' = (x_1+x_2+y_1) \cdot (x_3+\overline{y_1}+y_2) \cdot (x_4+\overline{y_2}+y_3)$$

$$\cdot \cdot \cdot (x_{k-2}+\overline{y_{k-4}}+y_{k-3}) \cdot (x_{k-1}+x_k+\overline{y_{k-3}})$$

Prove: 3SAT is NP-hard (4) optional

If C' is satisfiable

- Claim: at least one of x_i must be 1
- If all x_i are 0, C' becomes
  $(y_1) \cdot (y_1+\overline{y_2}) \cdot (y_2+\overline{y_3}) \cdot \cdot \cdot (y_{k-4}+\overline{y_{k-3}}) \cdot (y_{k-3})$ C' is NOT satisfiable, contradiction
- Therefore, C is satisfiable

$$C = (x_1+x_2+x_3+\ldots+x_k)$$

$$C' = (x_1+x_2+y_1) \cdot (x_3+\overline{y_1}+y_2) \cdot (x_4+\overline{y_2}+y_3) \cdot \cdot \cdot (x_{k-2}+\overline{y_{k-4}}+y_{k-3}) \cdot (x_{k-1}+x_k+\overline{y_{k-3}})$$

Prove: 3SAT is NP-hard (5)

optional

For clauses with one or two variables

If C has two variables, $C = (x_1+x_2)$, then we set $C' =$

$(x_1+x_2+z) \cdot (x_1+x_2+\overline{z})$

If C has one variable, $C = (x_1)$, then we set

$C' = (x_1+y+z) \cdot (x_1+\overline{y}+z) \cdot (x_1+y+\overline{z}) \cdot (x_1+\overline{y}+\overline{z})$

We have transformed every clause C in E to a sequence of clause C' with exactly three variables, such that C is satisfiable if and only if C' is.

This reduction can be done in polynomial time.