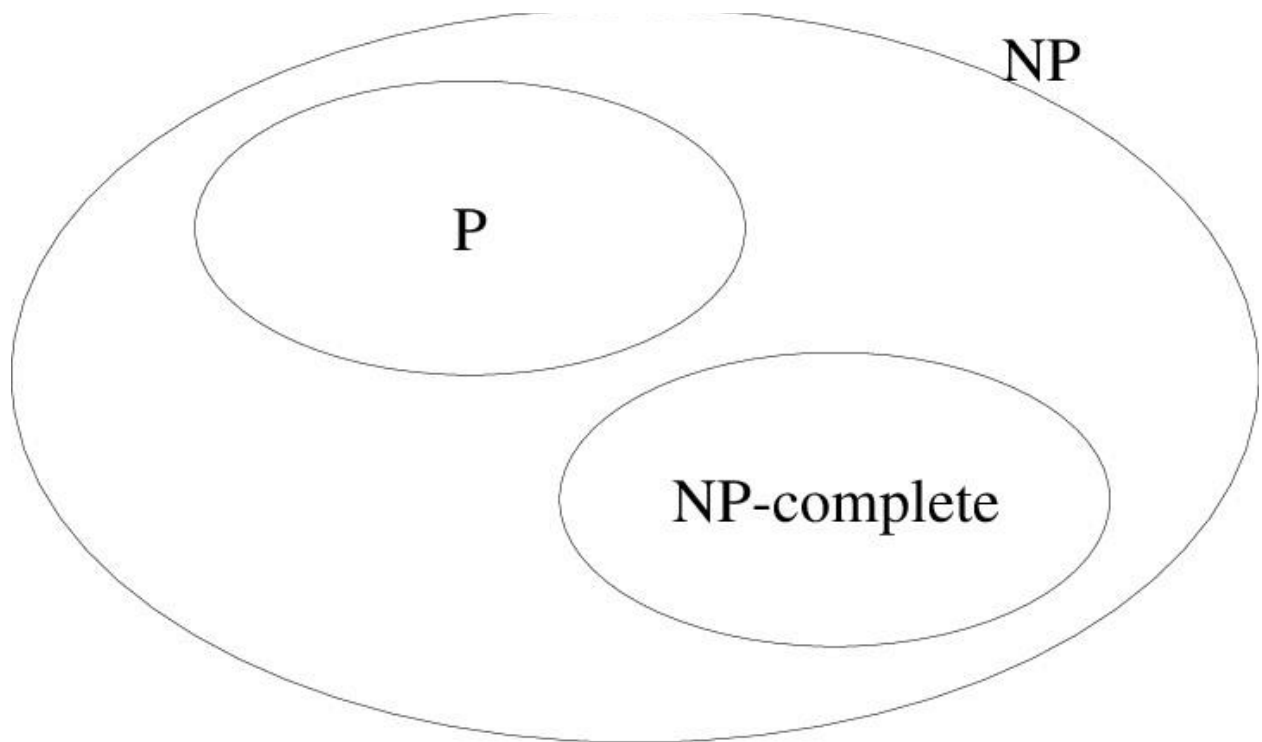


TOPIC: Intractability ||: P, NP, and NP-Complete

Welcome to the world of computational complexity! In this lesson, we will dive into the intriguing realm of P, NP, and NP-Complete problems. These concepts are fundamental in understanding the efficiency of algorithms and determining whether a problem is easy or hard to solve. Let's get started!



P Problems - Polynomial Time

P Problems: P stands for "polynomial time" and refers to the class of problems that can be solved by algorithms in a reasonable amount of time, where the time taken grows polynomial with the input size. In simple terms, these problems are efficiently solvable.

Example: Sum of Array

```
def sum_of_array(arr):  
    total = 0  
    for num in arr:  
        total += num  
    return total
```

NP Problems - Nondeterministic Polynomial Time

NP Problems:

NP stands for "nondeterministic polynomial time" and comprises problems for which, if a potential solution is given, it can be verified as correct in polynomial time. However, finding the solution itself efficiently is not guaranteed.

Example: Subset Sum

```
def subset_sum_exists(nums, target):  
    dp = [False] * (target + 1)  
    dp[0] = True  
    for num in nums:  
        for i in range(target, num - 1, -1):  
            dp[i] |= dp[i - num]  
    return dp[target]
```

NP-Complete Problems:

NP-Complete problems are a subset of NP problems that are believed to be the hardest problems in NP. If an efficient solution is found for any NP-Complete problem, it implies an efficient solution for all NP problems, making them inherently difficult.

Example: Traveling Salesman Problem (TSP)

```
import itertools
def tsp(graph):
    n = len(graph)
    min_distance = float('inf')
    for perm in itertools.permutations(range(n)):
        distance = 0
        for i in range(n - 1):
            distance += graph[perm[i]][perm[i + 1]]
        distance += graph[perm[-1]][perm[0]]
        min_distance = min(min_distance, distance)
    return min_distance
```

Implications and Conclusion

Implications: The P vs. NP question, which asks whether $P = NP$ or not, remains one of the seven Millennium Prize Problems, offering a \$1 million reward for its resolution. If $P = NP$, it would mean that problems with efficiently verifiable solutions also have efficiently findable solutions. However, if $P \neq NP$, then there exist problems for which verifying a solution is easy, but finding the solution is inherently hard.

Reducing Problems to Each Other

Reductions: To understand the relationship between different problems, we use reductions. A reduction is a way to show that one problem can be solved using a solution to another problem. In the context of NP-Completeness, we use polynomialtime reductions.

Example: Vertex Cover to Independent Set

```

def vertex_cover_to_independent_set(graph, k):
    complement_graph = {}
    for vertex in graph:
        complement_graph[vertex] = set()
        for neighbor in graph[vertex]:
            if neighbor != vertex and neighbor not in complement_graph[vertex]:
                complement_graph[vertex].add(neighbor)

    return independent_set(complement_graph, k)

```

Cook's Theorem and NP-Completeness

Cook's Theorem: In 1971, Stephen Cook introduced the concept of NP-Completeness and proved that the Boolean satisfiability problem (SAT) is NP-Complete. This was a groundbreaking result that led to the identification of numerous other problems as NP-Complete.

Example: Boolean Satisfiability Problem (SAT)

```

def is_satisfiable(clauses, assignment):
    for clause in clauses:
        clause_satisfied = False
        for literal in clause:
            if literal > 0 and assignment[literal] or literal < 0 and not assignment[-literal]:
                clause_satisfied = True
                break
        if not clause_satisfied:
            return False
    return True

```

P vs. NP Question and Beyond

P vs. NP Question: The question of whether $P = NP$ is one of the most important open problems in computer science. If $P = NP$, it implies that efficient algorithms exist for solving all problems for which solutions can be verified efficiently. However, if $P \neq NP$, there are problems for which efficient solutions cannot be found easily.

Beyond NP: Beyond NP-Complete problems, there are problems that are even harder. These are called "NP-Hard" problems. They may not be in NP themselves but are at least as hard as the hardest problems in NP.

P Decision problem.

Problem X is a set of strings.

Instance s is one string.

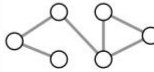
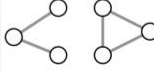
Algorithm A solves problem X : Def. Algorithm A runs in polynomial time if for every string s , $A(s)$ terminates in $\leq p(|s|)$ "steps," where $p(\cdot)$ is some polynomial function.

Def. P = set of decision problems for which there exists a poly-time algorithm.

4 length of s problem

PRIMES: { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, ... }

Some Problems in P

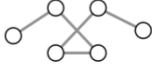
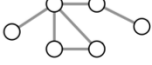
problem	description	poly-time algorithm	yes	no
MULTIPLE	Is x a multiple of y ?	grade-school division	51, 17	51, 16
REL-PRIME	Are x and y relatively prime?	Euclid's algorithm	34, 39	34, 51
PRIMES	Is x prime?	Agrawal-Kayal-Saxena	53	51
EDIT-DISTANCE	Is the edit distance between x and y less than 5?	Needleman-Wunsch	niether neither	acgggt ttttta
L-SOLVE	Is there a vector x that satisfies $Ax = b$?	Gauss-Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
U-CONN	Is an undirected graph G connected?	depth-first search		

NP:

Def. Algorithm $C(s, t)$ is a certifier for problem X if for every string $s : s \in X$ iff there exists a string t such that $C(s, t) = \text{yes}$.

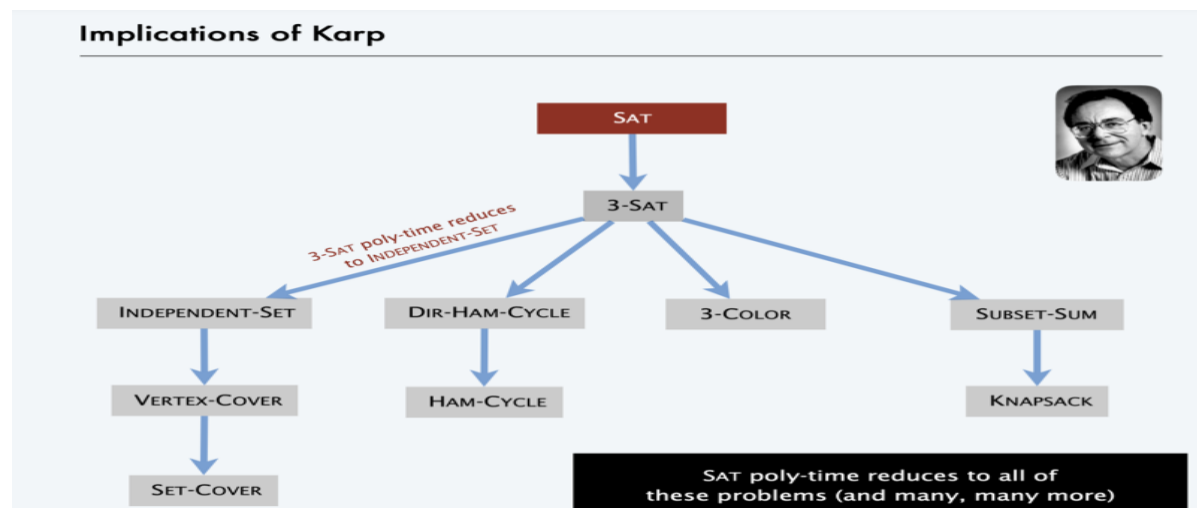
Def. NP = set of decision problems for which there exists a poly-time certifier. • $C(s, t)$ is a poly-time algorithm. • Certificate t is of polynomial size: $|t| \leq p(|s|)$ for some polynomial $p(\cdot)$.

Some Problems in NP:

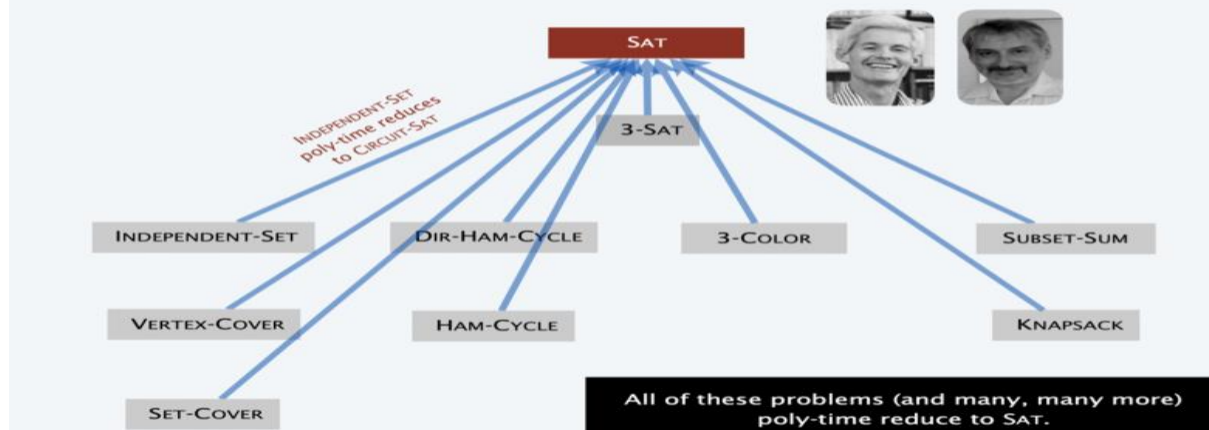
problem	description	poly-time algorithm	yes	no
L-SOLVE	Is there a vector x that satisfies $Ax = b$?	Gauss-Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
COMPOSITES	Is x composite?	Agrawal-Kayal-Saxena	51	53
FACTOR	Does x have a nontrivial factor less than y ?	???	(56159, 50)	(55687, 50)
SAT	Given a CNF formula, does it have a satisfying truth assignment?	???	$\neg x_1 \vee x_2 \vee \neg x_3$ $x_1 \vee \neg x_2 \vee x_3$ $\neg x_1 \vee \neg x_2 \vee x_3$	$\neg x_2$ $x_1 \vee x_2$ $\neg x_1 \vee x_2$
HAMILTON-PATH	Is there a simple path between u and v that visits every node?	???		

“NP captures vast domains of computational, scientific, and mathematical endeavors, and seems to roughly delimit what mathematicians and scientists have been aspiring to compute feasibly.” — Christos Papadimitriou

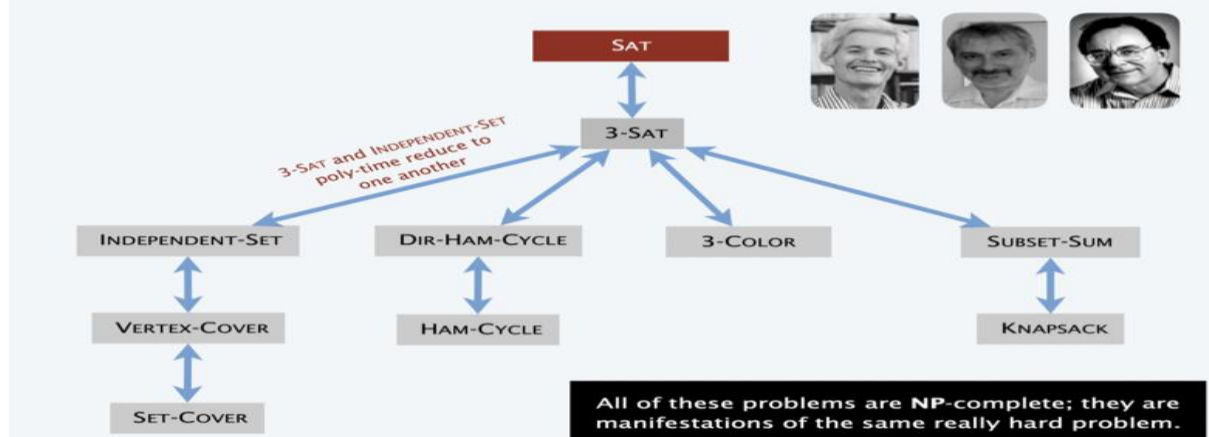
“In an ideal world it would be renamed P vs VP.” — Clyde Kruskal



Implications of Cook-Levin



Implications of Karp + Cook-Levin



Some NP-complete problems

Basic genres of NP-complete problems and paradigmatic examples.

Packing/covering problems: SET-COVER, VERTEX-COVER, INDEPENDENT-SET.

- Constraint satisfaction problems: CIRCUIT-SAT, SAT, 3-SAT.
- Sequencing problems: HAMILTON-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Practice. Most NP problems are known to be either in P or NP-complete. NP-intermediate?

FACTOR, DISCRETE-LOG, GRAPH-ISOMORPHISM,

Theorem. [Ladner 1975] Unless $P = NP$, there exist problems in NP that are neither in P nor NP-complete.

Practical Implications

Real-World Impact: The distinction between P, NP, and NP-Complete problems has significant real-world implications. Many real-world optimization problems are NP-Complete (e.g., scheduling, routing), making them challenging to solve optimally for large inputs. Despite their theoretical difficulty, approximate solutions and heuristics are often used in practice.

Cryptography: The hardness of some problems forms the basis of modern cryptography. If factoring large numbers (as in RSA encryption) were to become efficient, it would have serious implications for online security.

Conclusion and Future Directions

Conclusion: The study of P, NP, and NP-Complete problems provides us with essential insights into the boundaries of computation. It helps us classify problems based on their computational complexity and understand the relationships between different problems.

Future Directions: As technology advances, researchers continue to explore the depths of computational complexity. Solving the P vs. NP question remains a goal, and finding practical algorithms for NP-Complete problems continues to be a subject of research, potentially leading to breakthroughs in various fields.

Conclusion: Understanding the classes of problems within computational complexity—P, NP, and NP-Complete—sheds light on the limits of algorithmic efficiency. P problems are efficiently solvable, NP problems have easily verifiable solutions, and NP-Complete problems are some of the most challenging. This knowledge is vital for algorithm design, optimization, and evaluating.