ANALYSIS OF ALGORITHM & RESEARCHING COMPUTING

By Prof. Punit Mangal

Polynomial-Time Algorithms

- Are some problems solvable in polynomial time?
  - Of course: every algorithm we've studied provides polynomial-time solution to some problem
  - We define **P** to be the class of problems solvable in polynomial time
- Are all problems solvable in polynomial time?
  - No: Turing's "Halting Problem" is not solvable by any computer, no matter how much time is given
  - Such problems are clearly intractable, not in **P**

NP-Completeness

- So far we've seen a lot of good news!
  - Such-and-such a problem can be solved quickly (i.e., in close to linear time, or at least a time that is some small polynomial function of the input size)
- NP-completeness is a form of bad news!
  - Evidence that many important problems can not be solved quickly.
- NP-complete problems really come up all the time!

Optimization & Decision Problems

- **Decision problems**
  - Given an input and a question regarding a problem, determine if the answer is yes or no
- **Optimization problems**
  - Find a solution with the "best" value
- Optimization problems can be cast as decision problems that are easier to study
  - *E.g.:* Shortest path: G = unweighted directed graph
    - Find a path between u and v that uses the fewest edges
    - *Does a path exist from u to v consisting of at most k edges?*

Class of "P" Problems

- **Class P** consists of (decision) problems that are solvable in polynomial time
- Polynomial-time algorithms
  - Worst-case running time is $\mathbf{O(n^k)}$, for some constant k
- Examples of polynomial time:
  - $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
- Examples of non-polynomial time:
  - $O(2^n)$, $O(n^n)$, $O(n!)$

Tractable/Intractable Problems

- Problems in P are also called **tractable OR easy**
- Problems **not** in P are **intractable or unsolvable or hard**
  - Can be solved in reasonable time only for small inputs
  - Or, can not be solved at all

Example of Unsolvable Problem

- Turing discovered in the 1930's that there are problems **unsolvable** by *any* algorithm.
- The most famous of them is the ***halting problem***
  - Given an arbitrary algorithm and its input, will that algorithm eventually halt, or will it continue forever in an "*infinite loop*?"

Nondeterministic and NP Algorithms

Nondeterministic algorithm = two stage procedure:

1) Nondeterministic ("guessing") stage:

    generate randomly an arbitrary string that can be thought of as a candidate solution ("certificate")

2) Deterministic ("verification") stage:

    take the certificate and the instance to the problem and returns YES if the certificate represents a solution

NP algorithms (Nondeterministic polynomial)

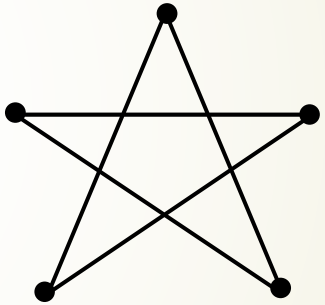
verification stage is polynomial

Class of "NP" Problems

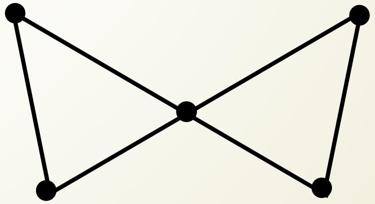
- **Class NP** consists of problems that could be solved by NP algorithms
  - i.e., verifiable in polynomial time
- If we were given a "certificate" of a solution, we could verify that the certificate is correct in time polynomial to the size of the input
- Warning: NP does **not** mean "non-polynomial"

E.g.: Hamiltonian Cycle

- **Given:** a directed graph G = (V, E), determine a simple cycle that contains each vertex in V
  - Each vertex can only be visited once
- **Certificate**:
  - Sequence: $\langle v_1, v_2, v_3, \ldots, v_{|V|} \rangle$

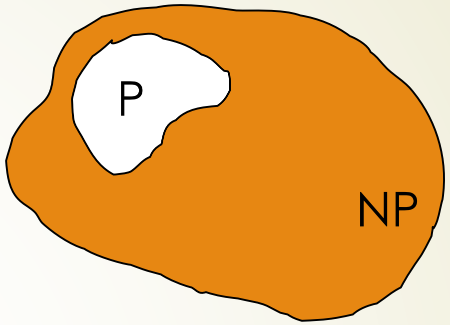


hamiltonian



not hamiltonian

Is P = NP?



- Any problem in P is also in NP:

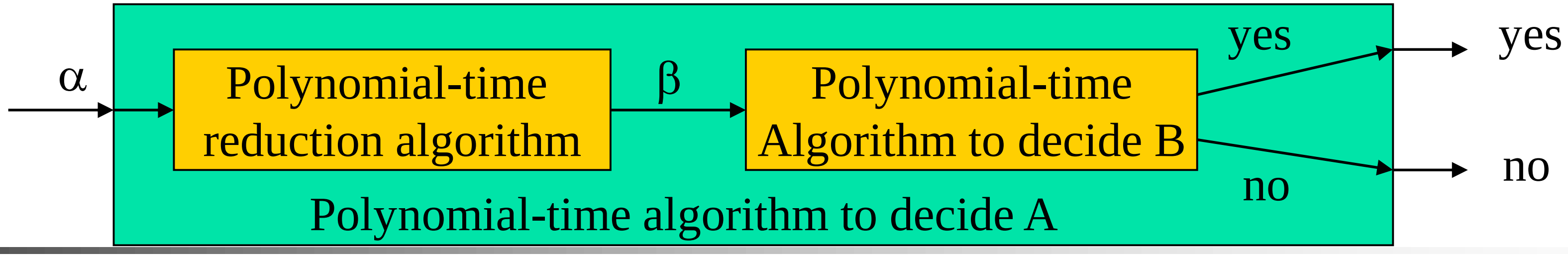
  $P \subseteq NP$

- The big (and **open question**) is whether $NP \subseteq P$ or $P = NP$
  - i.e., if it is always easy to check a solution, should it also be easy to find a solution?
- Most computer scientists believe that this is false but we do not have a proof …

Reductions

Suppose that there is a different decision problem, say B, that we already know how to solve in polynomial time. Finally, suppose that we have a procedure that transforms any instance α of A into some instance β of B with the following characteristics:

1. The transformation takes polynomial time.
2. The answer are the same. That is, the answer for α is "yes" if and only if the answer for β is also "yes."



We can call such a procedure a polynomial-time reduction algorithm and, it provides us a way to solve problem A in polynomial time:

1. Given an instance α of problem A, use a polynomial-time reduction algorithm to transform it to an instance β of problem B.
2. Run the polynomial-time decision algorithm for B on the instance β.
3. Use the answer for β as the answer for α.

34.1 Polynomial time

Polynomial time solvable problem are regarded as tractable.

- Even if the current best algorithm for a problem has a running time of $\Theta(n^{100})$, it is likely that an algorithm with a much better running time will soon be discovered.
- Problems for many reasonable models of computation, can be solved in one model can be solved in polynomial in another.
- Polynomial-time solvable problems has a nice closure property.

$$f, g \quad are \quad polynomial$$
$$\Rightarrow f(g) \quad is \quad also\ polynomial$$

We say that a function $f:\{0,1\}^* \to \{0,1\}^*$ is ***polynomial-time computable*** if there exists a polynomial-time algorithm A that given any $x \in \{0,1\}^*$, produces as output $f(x)$.

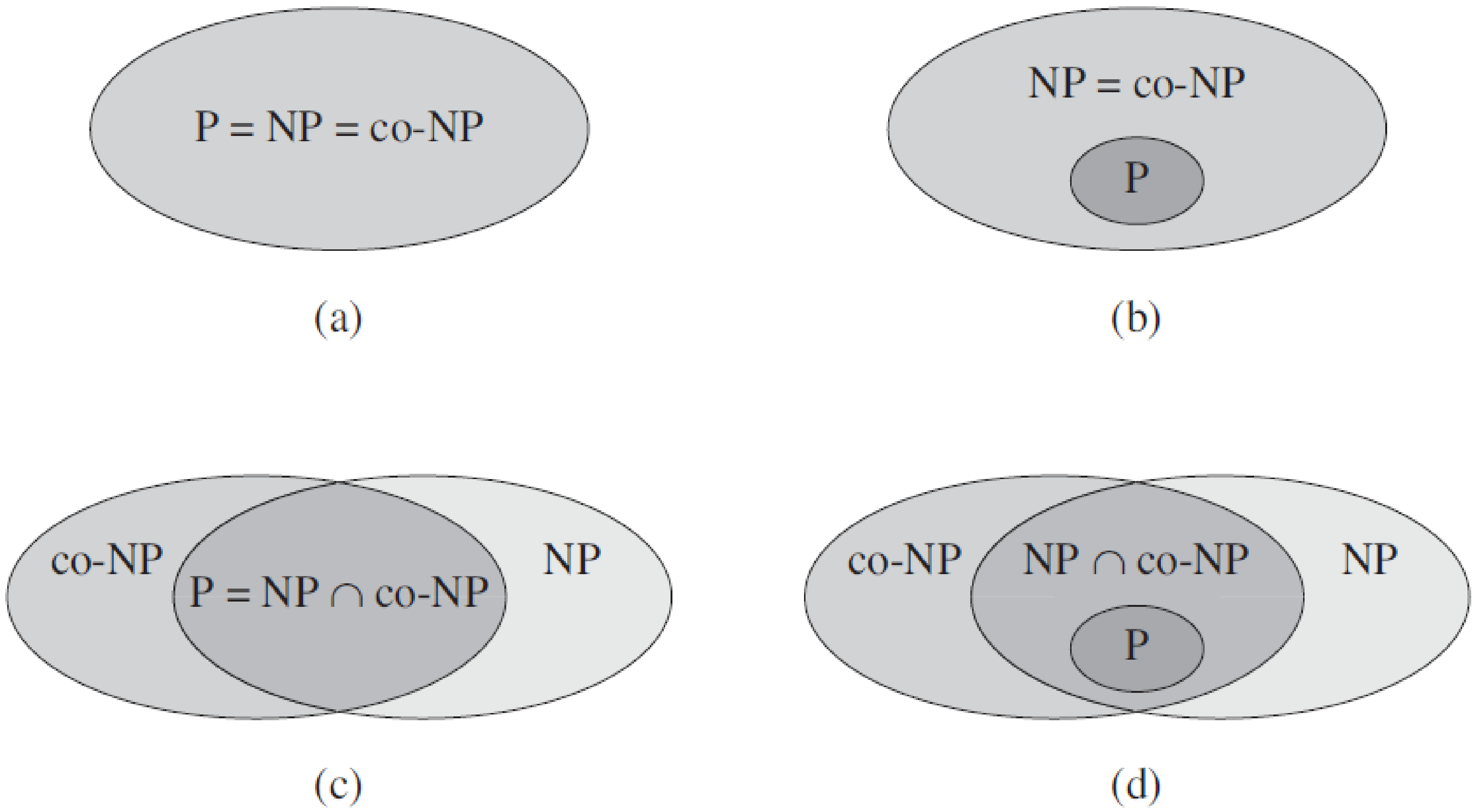


Figure 34.3 Four possibilities for relationships among complexity classes. In each diagram, one region enclosing another indicates a proper-subset relation. **(a)** $P = NP = \text{co-}NP$. Most researchers regard this possibility as the most unlikely. **(b)** If NP is closed under complement, then $NP = \text{co-}NP$, but it need not be the case that $P = NP$. **(c)** $P = NP \cap \text{co-}NP$, but NP is not closed under complement. **(d)** $NP \neq \text{co-}NP$ and $P \neq NP \cap \text{co-}NP$. Most researchers regard this possibility as the most likely.