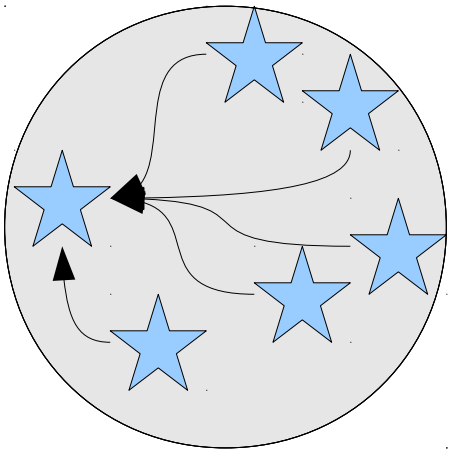
Complexity Theory

Part Three

The Tantalizing Truth

Theorem: If *any* **NP**-complete language is in **P**, then **P** = **NP**.

Proof: Suppose that L is **NP**-complete and $L \in \mathbf{P}$. Now consider any arbitrary **NP** problem A. Since L is **NP**-complete, we know that $A \leq_p L$. Since $L \in \mathbf{P}$ and $A \leq_p L$, we see that $A \in \mathbf{P}$. Since our choice of A was arbitrary, this means that $\mathbf{NP} \subseteq \mathbf{P}$, so **P** = **NP**. ■

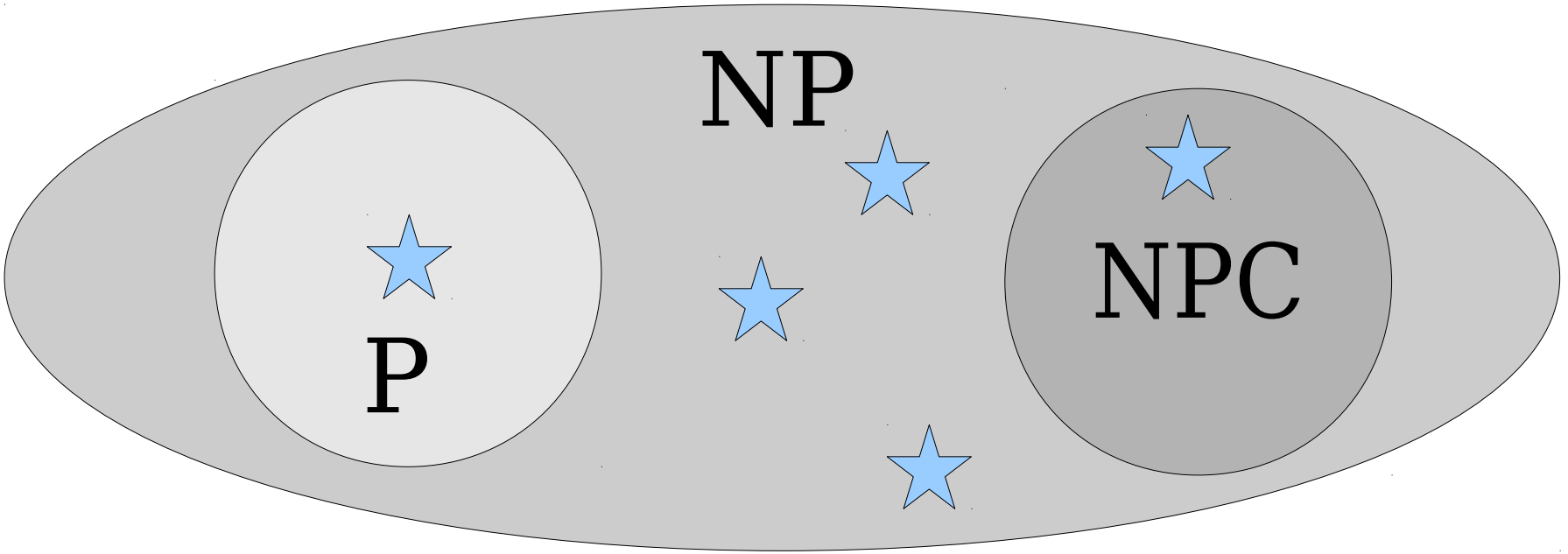


P = NP

The Tantalizing Truth

Theorem: If *any* **NP**-complete language is not in **P**, then **P** ≠ **NP**.

Proof: Suppose that L is an **NP**-complete language not in **P**. Since L is **NP**-complete, we know that $L \in$ **NP**. Therefore, we know that $L \in$ **NP** and $L \notin$ **P**, so **P** ≠ **NP**. ■



How do we even know NP-complete problems exist in the first place?

Satisfiability

- A propositional logic formula φ is called ***satisfiable*** if there is some assignment to its variables that makes it evaluate to true.
  - $p \wedge q$ is satisfiable.
  - $p \wedge \neg p$ is unsatisfiable.
  - $p \rightarrow (q \wedge \neg q)$ is satisfiable.
- An assignment of true and false to the variables of φ that makes it evaluate to true is called a ***satisfying assignment***.

SAT

- The ***boolean satisfiability problem*** (***SAT***) is the following:

  Given a propositional logic formula φ, is φ satisfiable?

- Formally:

  SAT **= { ⟨φ⟩ | φ is a satisfiable PL formula }**

SAT = { ⟨φ⟩ | φ is a satisfiable PL formula }

The language SAT happens to be in **NP**. Think about how a polynomial-time verifier for SAT might work. Which of the following would work as certificates for such a verifier, given that the input is a propositional formula φ?

A. The truth table of φ.
B. One possible variable assignment to φ.
C. A list of all possible variable assignments for φ.
D. None of the above, or two or more of the above.

Answer at **PollEv.com/cs103** or
text **CS103** to **22333** once to join, then ***A***, ***B***, ***C***, or ***D***.

Theorem (Cook-Levin): SAT is **NP**-complete.

Proof Idea: To see that **SAT** ∈ **NP**, show how to make a polynomial-time verifier for it. Key idea: have the certificate be a satisfying assignment.

To show that **SAT** is **NP**-hard, given a polymomial-time verifier V for an arbitrary **NP** language L, for any string w you can construct a polynomially-sized formula $\varphi(w)$ that says "there is a certificate c where V accepts $\langle w, c \rangle$." This formula is satisfiable if and only if $w \in L$, so deciding whether the formula is satisfiable decides whether w is in L.

Proof: Take CS154!

Why All This Matters

- Resolving **P** $\stackrel{?}{=}$ **NP** is equivalent to just figuring out how hard SAT is.
  - If SAT $\in$ **P**, then **P** = **NP**.

    If SAT $\notin$ **P**, then **P** $\neq$ **NP**.

Sample **NP**-Hard Problems

- ***Computational biology:*** Given a set of genomes, what is the most probable evolutionary tree that would give rise to those genomes? *(Maximum parsimony problem)*
- ***Game theory:*** Given an arbitrary perfect-information, finite, twoplayer game, who wins? *(Generalized geography problem)*
- ***Operations research:*** Given a set of jobs and workers who can perform those tasks in parallel, can you complete all the jobs within some time bound? *(Job scheduling problem)*
- ***Machine learning:*** Given a set of data, find the simplest way of modeling the statistical patterns in that data *(Bayesian network inference problem)*
- ***Medicine:*** Given a group of people who need kidneys and a group of kidney donors, find the maximum number of people who can end up with kidneys *(Cycle cover problem)*
- ***Systems:*** Given a set of processes and a number of processors, find the optimal way to assign those tasks so that they complete as soon as possible *(Processor scheduling problem)*

Please evaluate this course on Axess.
Your feedback really makes a difference.

Your Questions