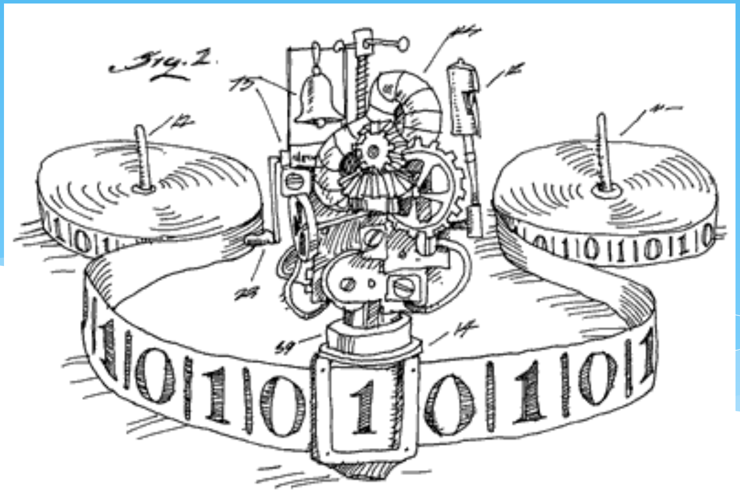
EECS 376: Foundations of Computer Science

Chris Peikert

8 March 2023

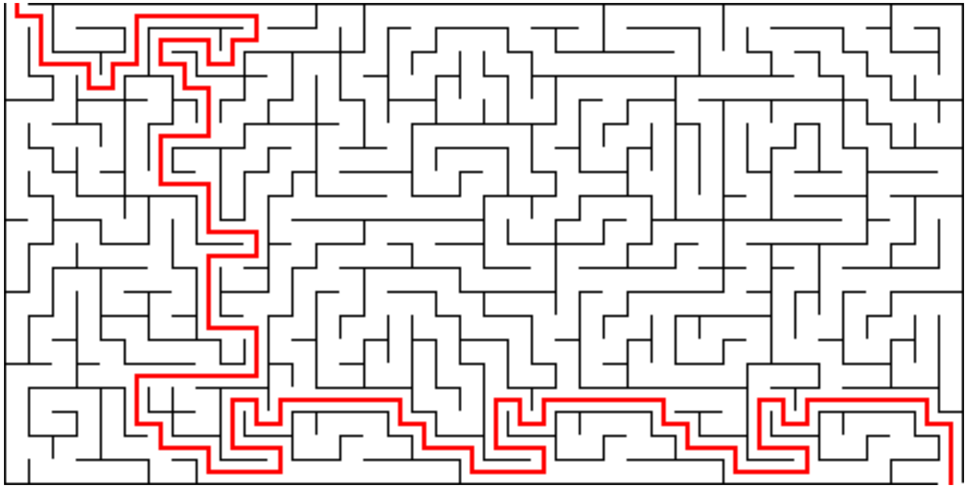


Today's Agenda

1) Recap: NP (efficiently verifiable languages)
2) Satisfiability Problem: SAT
3) Cook-Levin Theorem:
   - SAT is "as hard as" *any* problem in NP

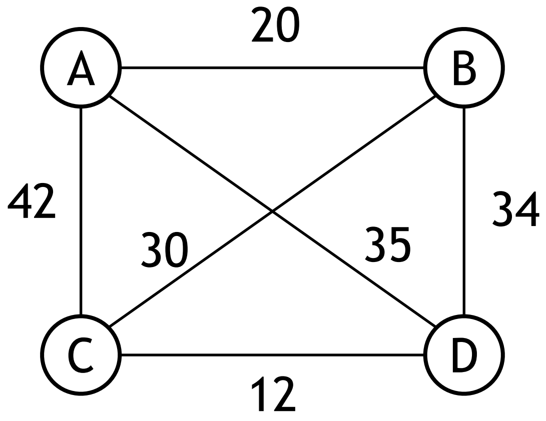
Verifiable Problems

* **Example 1:** Given a maze, is there a route through the maze from the start to finish?
* **Answer (from wizard):** Yes; see the route below.
* **Response:** We follow the route and are convinced.



Efficiently Verifiable Problems

- **Example 2: TSP (decision version):**
  Given 4 cities and pair-wise distances between them, is there a cycle of length at most 100 through all the cities?
- **Answer (from wizard):** Yes; $A \to B \to C \to D \to A$.
- **Response:** We check that this visits all the cities in a cycle, compute the cost 20+30+12+35 = 97 < 100, and are convinced.



The Class NP

* **Definition:** A decision problem L is ***efficiently verifiable*** if there exists an algorithm $V(x,c)$, called a ***verifier***, satisfying:

1. $V(x,c)$ is *efficient* with respect to x, i.e., polynomial time in $|x|$.
2. For every $x \in L$, there exists some c such that $V(x,c)$ accepts.
3. For every $x \notin L$, $V(x,c)$ rejects for *all* c.

Given 1, conditions 2+3 are equivalent to:

$$x \in L \iff \exists\, c \text{ s.t. } V(x,c) \text{ accepts.}$$

Definition: the class **NP** = the set of all efficiently verifiable languages.
I.e.: $L \in \mathbf{NP}$ if L is efficiently verifiable.

Practice with Verifiers

L_{Comp}= {n : n is composite (not prime)}

L_{HAM}= {G : G has a Hamiltonian cycle}

L_{Primes} = {n : n is prime } (complement of L_{Comp})

Not obvious. But there is a clever verifier and cert! [Pratt 1975]

$L_{non-HAM}$ = {G : G has no Hamiltonian cycle}
We **do not expect** the problem to have an efficient verifier!
(Would have very surprising consequences.)

P vs NP

* $L \in$ **P** if there is a polynomial-time (in $|x|$) algorithm M where:
  * $x \in L \implies M(x)$ accepts
  * $x \notin L \implies M(x)$ rejects

$x \in L \iff M(x)$ accepts

* $L \in$ **NP** if there is a polynomial-time (in $|x|$) algorithm V where:
  * $x \in L \implies V(x, c)$ accepts for <u>some</u> c
  * $x \notin L \implies V(x, c)$ rejects for <u>every</u> c

$x \in L \iff \exists\, c : V(x, c)$ accepts

* **Observe:** P $\subseteq$ NP ($V(x, c)$ can ignore c and just run $M(x)$.)
* **$1,000,000 question:** Is P = NP?
  Is every efficiently *<u>verifiable</u>* problem also efficiently *<u>solvable</u>*?
  Seems unlikely… but we don't know for sure!

EECS M ECE CSE

What P vs NP is "About"

Intuitively: Verifying a correct solution seems much "easier" than finding one (or even determining if there is one).

Examples: TSP, Ham-Cycle, Subset-Sum, **Sudoku**, **376 Homework**, …
All these problems are in NP — but we don't know if they are in P!

Major open question (P vs NP): Does P = NP ?
Is every efficiently verifiable problem also efficiently solvable?

P vs NP

* *… Let p(n) be the number of steps to find a proof of length n. The question is, how rapidly does p(n) grow for an optimal machine? It is possible to show that p(n) > Kn. If there really were* ***a machine with p(n) ~ Kn (or even just ~ Kn^2)*** *then that would have consequences of the greatest significance. Namely, this* ***would clearly mean that the thinking of a mathematician in the case of yes-or-no questions could be completely replaced by machine …***

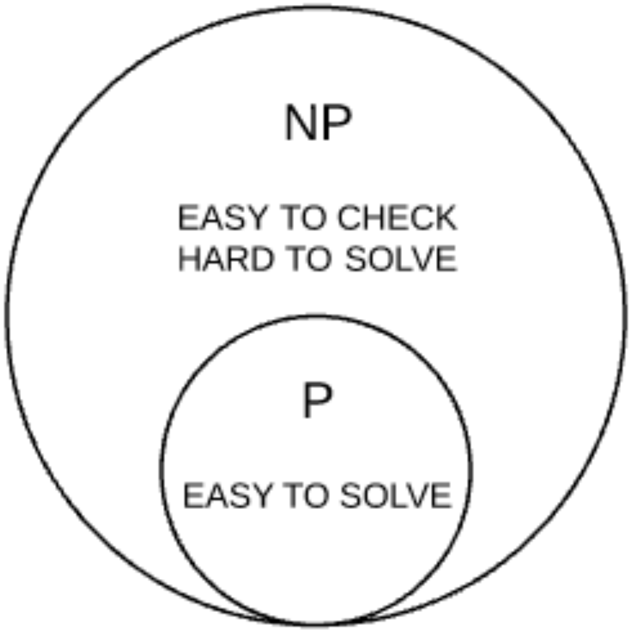
  - *Kurt Godel's letter to von Neumann in 1956 (15 years before P vs NP was formalized!)*

* *"****If P = NP****, then the world would be a profoundly different place than we usually assume it to be. There would be* ***no special value in 'creative leaps'****, no fundamental gap between solving a problem and recognizing the solution once it's found."*

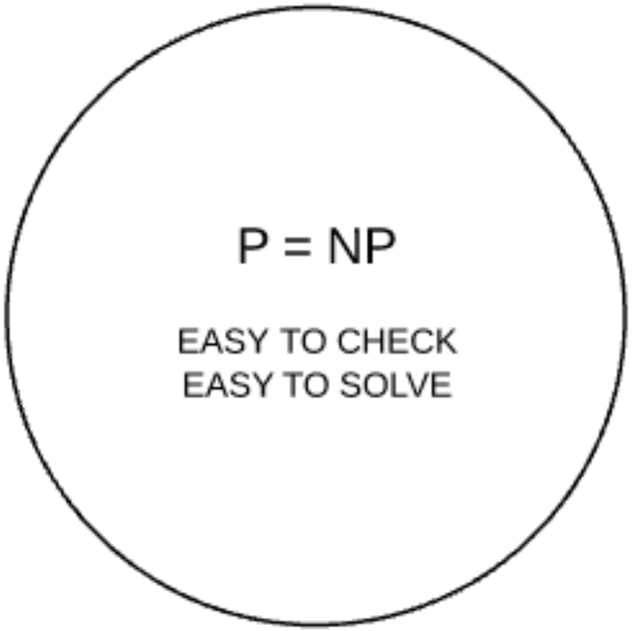
  \- Scott Aaronson

Pictorially

Believed: $P \neq NP$



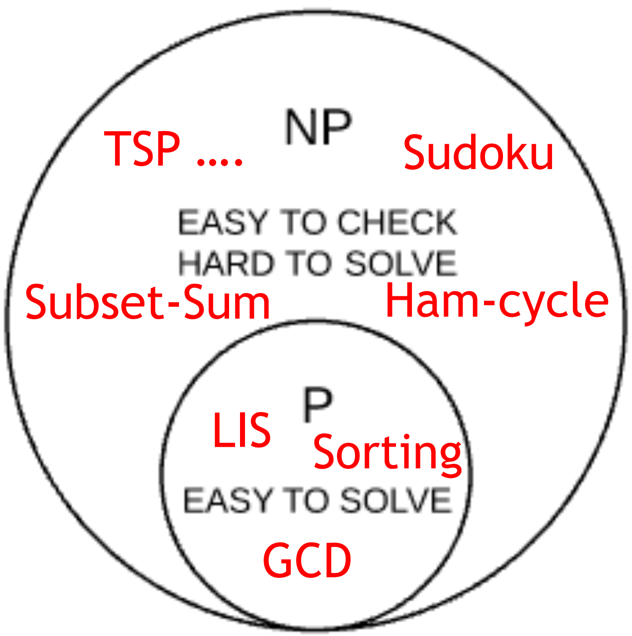
If P = NP



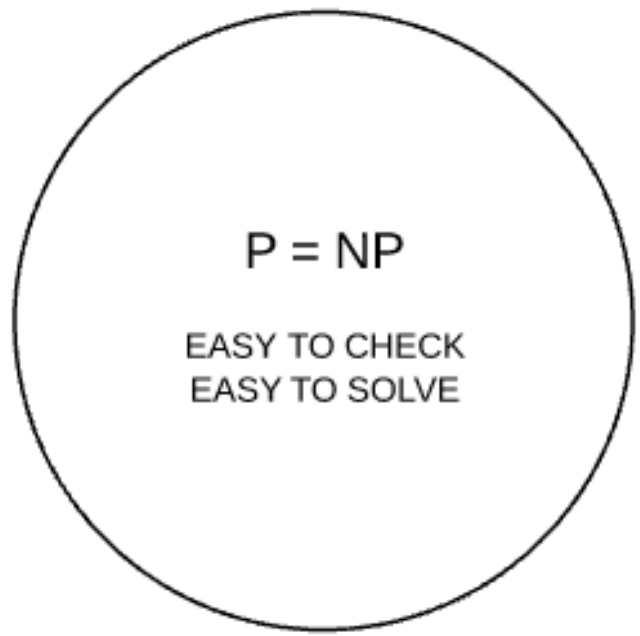
Pictorially

Problems beyond NP
(won't study here)

Believed: P $\neq$ NP

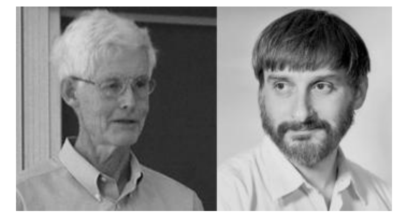


If P = NP

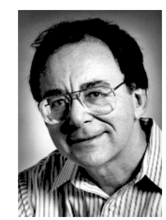


Two Amazing Works (Given Turing Awards)

Cook-Levin (1971): SAT is "NP-hard." In particular:
If SAT is in P, then all of NP is in P, i.e., P=NP.
(Easy: if SAT is not in P, then $P \neq NP$.)



So, to resolve P vs. NP, we "just" need to determine the status of SAT!



Karp (1972): TSP, Ham-Cycle, Subset Sum, …
all of these are "equivalent" to SAT.

Either all of them are in P (so P=NP), or none are (so $P \neq NP$).

A “Hard” Language for NP

Cook-Levin Theorem:

Any poly-time algorithm for SAT

would yield, via efficient reduction,

a poly-time algorithm for *any* language in NP
(i.e., any efficiently verifiable language)

Satisfiability

* Boolean ***variables*** x,y,z … taking values true or false (1 or 0)
* A Boolean ***literal*** is a variable or its negation z, $\neg$z
* A Boolean ***operator*** is AND, OR ($\wedge, \vee$)
* A Boolean ***formula*** is a formula involving Boolean literals and operators, e.g., $\phi = (\neg x \wedge y) \vee (x \wedge \neg z)$

* A ***satisfying assignment*** for ϕ is a true/false assignment to the variables such that ϕ evaluates to true.
* ϕ is ***satisfiable*** if it has a satisfying assignment
* **SAT** = $\{\phi : \phi$ is a satisfiable Boolean formula$\}$

Satisfiability

* **Ex 1:** $\phi(x,y) = \neg x \wedge y$
* **Question:** What is $\phi(T,F)$ and $\phi(F,F)$?
* **Ex 2:** $\phi(x,y,z) = (\neg x \vee y) \wedge (\neg x \vee z) \wedge (y \vee z) \wedge (x \vee \neg z)$
* **Question:** Are these ϕ satisfiable?

* **Question:** Is SAT $\in$ **NP** ?

  I.e., is SAT efficiently verifiable?
  Can the satisfiability of a given ϕ be efficiently verified?

Cook-Levin Outline

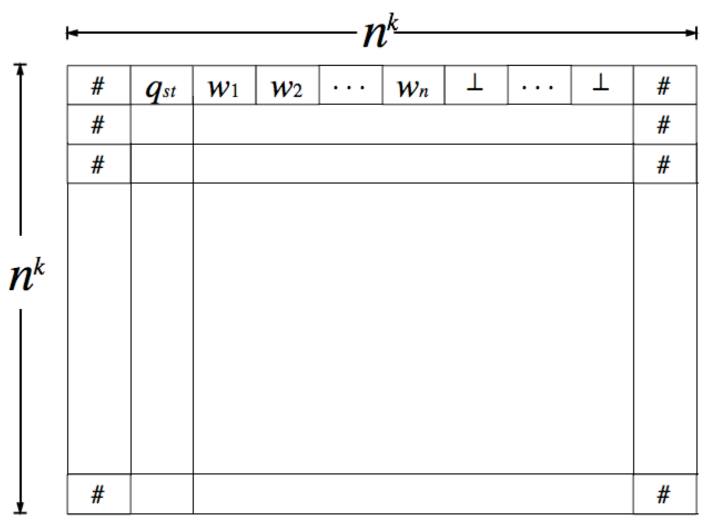
* **Theorem [Cook-Levin]:** If SAT $\in$ **P**, then **NP** $\subseteq$ **P**.
* Let D_{SAT} be an efficient decider for SAT.
* Let L $\in$ **NP**, so L has an efficient verifier V.
* Goal: L $\in$ **P** via efficient decider D_L that uses D_{SAT} & V.
* $D_L(x)$:
  * Cleverly construct a poly-sized Boolean formula $\phi_{V,x}$ so that:
  * $x \in L \iff \phi_{V,x}$ is satisfiable.
  * Output $D_{\mathsf{SAT}}(\phi_{V,x})$.

Initial Observations

* V(.,.) is an efficient verifier (i.e., a TM) for L $\in$ **NP**.
* For every input x and certificate c:
  * V runs for at most $|x|^k$ steps, for some constant k.
    $\Rightarrow$ V can read/affect only the first $|x|^k$ tape cells
* Goal: Given some x, design a Boolean formula $\phi_{V,x}$ that is satisfiable
  iff some c causes V(x,c) to accept in at most $|x|^k$ steps.
* Solution: Turing machine inspection!
* **Definition:** A configuration of V represents V's tape contents, state, head location.
* **Example:** $011q_50001$ means:
  * V's tape has $0110001\perp\perp\perp$…
  * V is in state q_5; V's head points to the 4th cell

A Configuration Tableau

- A ***tableau*** is an array of symbols:
  - Rows represent configurations (flanked by #s)
  - Symbols can be from S = {0,1} ∪ Q ∪ {#, \$, ⊥}
  - Successive rows correspond to configurations after each step of V



Initial configuration

After 1 step

V halts after at most n^k steps

Proof Overview

* Given an input x, construct a Boolean formula $\phi_{V,x}$ that represents an accepting V-tableau with input x (and unspecified c):
* $$V(x, c) \text{ accepts for some } c \iff \phi_{V,x} \text{ is satisfiable.}$$
* Variables: $t_{i,j,s}$ denotes whether tableau cell (i,j) has symbol s.
* $\phi_{V,x} = \phi_{start} \wedge \phi_{cell} \wedge \phi_{accept} \wedge \phi_{step}$

1. ϕ_{start} enforces the starting config on input x (w/ unspecified c)
2. ϕ_{cell} ensures that every cell contains exactly one symbol
3. ϕ_{accept} ensures that V reaches an accepting configuration
4. ϕ_{step} ensures that each configuration follows from the previous one, according to the code of V

The Starting Configuration

ϕ_{start} enforces the starting configuration:

#	q_0	x	\$	c	$\perp$	$\perp$	#

* Initial state q_0,
* Input x, $|x| = n$; certificate c, $|c| = n^k$
* \$ - a special symbol ("comma") that separates x and c
* Certificate c is unspecified, so we leave a "placeholder"

$$\left.\phi_{start} = t_{1,1,\#} \wedge t_{1,2,q0} \wedge t_{1,3,x1} \wedge t_{1,4,x2} \wedge \ldots \wedge t_{1,n+2,xn} \wedge t_{1,n+3,\$} \wedge \right\}\text{This fixes the first n+3 symbols}$$

$$\underbrace{(t_{1,n+4,1} \vee t_{1,n+4,0} \vee t_{1,n+4,\perp})}_{(c_1 \text{ can be either 1 or 0 or } \perp)} \wedge (t_{1,n+5,1} \vee t_{1,n+5,0} \vee t_{1,n+5,\perp}) \wedge \ldots$$

Cell Consistency

* ϕ_{cell} ensures that every cell contains **exactly** one symbol:

every cell

cannot contain two different symbols

$$\bigwedge_{1 \le i,j \le n^k} \left[\bigvee_{\sigma \in S} t_{i,j,\sigma} \wedge \bigwedge_{\sigma \ne \tau \in S} \left(\overline{t_{i,j,\sigma} \wedge t_{i,j,\tau}} \right) \right]$$

contains at least one symbol

Accepting Configurations

ϕ_{accept} ensures that V reaches an accepting configuration:

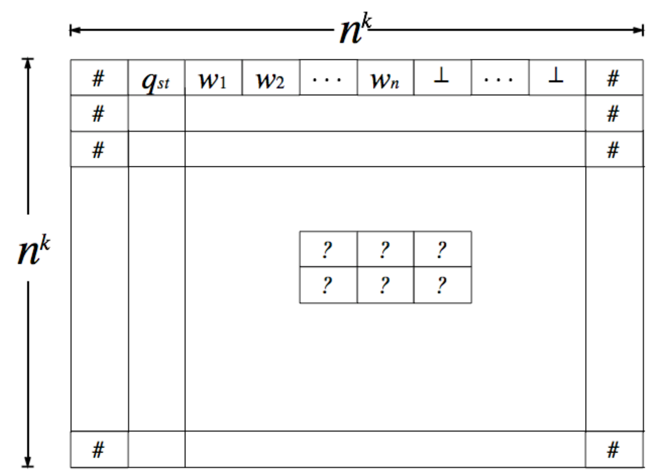
$$\bigvee_{1 \le i, j \le n^k} t_{i,j,qaccept}$$

Computational Steps

ϕ_{step} ensures that each configuration follows from the previous configuration according to V's transition function:

Definition: A 2x3 "window" is valid if it could appear in a valid tableau

Theorem: The whole tableau is valid if and only if every 2x3 window is valid.



Can write a small formula for window validity: e.g., q01 —> 1q'1 if there is a q —>q' transition (0—>1, R).

Conclusion

- **Conclusion:** **P** = **NP** iff there is an efficient algorithm for SAT (determining satisfiability of Boolean formulae)

- **Common Belief:** There is no efficient algorithm for determining satisfiability, so **P** ≠ **NP**.