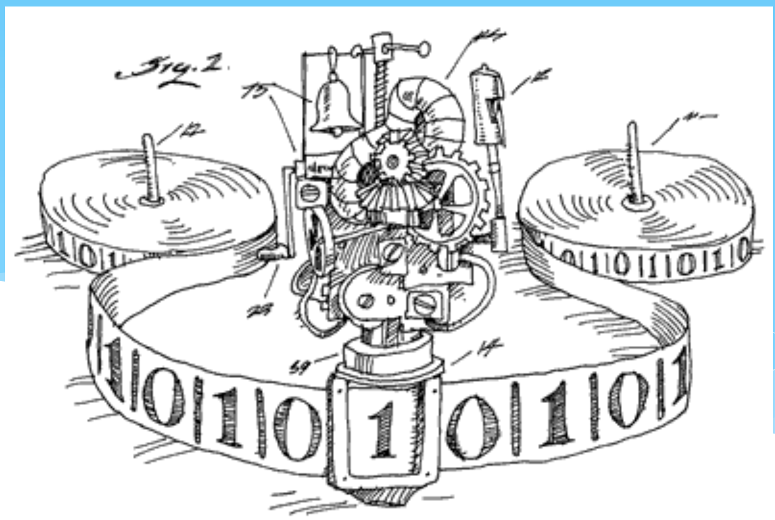
EECS 376: Foundations of Computer Science

Seth Pettie

Lecture 15



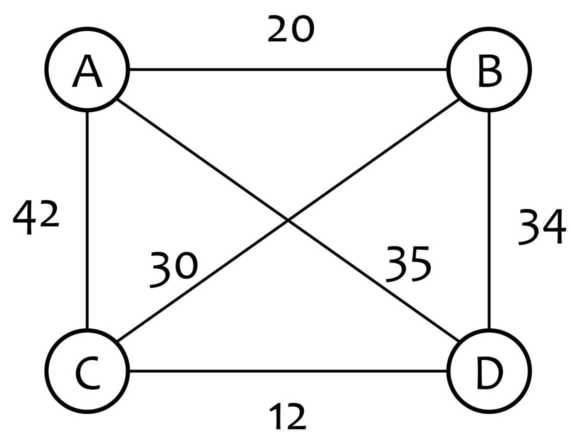
Today's Agenda

1) Recap: NP (efficiently verifiable languages)
2) Cook-Levin Theorem

   The SAT problem is as hard as any problem in NP

Verifiable Computations

* **Example:** Decision version of ***Traveling Salesperson Problem (TSP)*** Given 4 cities and pair-wise distances between them, is there a tour of length at most 100 that visits all the cities?
* **Remark:** Here we only care about feasibility, not the actual tour.
* **Certificate:** $A \to B \to C \to D \to A$. Cost: 20+30+12+35 = 97.
* **Reply:** We can verify and are convinced.



Verifiable Computations

* **Example 3:** Subset Sum
* Given integers $a_1, \dots, a_n$ and target t, is there a subset of numbers that sums to t?

* **Certificate:** The subset of numbers.
* **Reply:** We can verify 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*** such that:

1. $V(x, c)$ is efficient with respect to x (polynomial time in $|x|$).
2. If $x \in L$, then there is <u>*some*</u> ***certificate*** c such that $V(x, c)$ accepts.
3. If $x \notin L$, then $V(x, c)$ rejects <u>*all*</u> ***certificates*** c.

* **Definition:** The class **NP** = the class of efficiently <u>*verifiable*</u> languages

P and NP

* **Formally:** Let L be a language.
* $L \in$ **P** if there exists a polynomial time in $|x|$ algorithm $M(x)$ such that:
  * $x \in L \Longrightarrow M(x)$ accepts
  * $x \notin L \Longrightarrow M(x)$ rejects

* $L \in$ **NP** if there exists a polynomial time in $|x|$ algorithm $V(x, c)$ such that:
  * $x \in L \Longrightarrow V(x, c)$ accepts for at least one c
  * $x \notin L \Longrightarrow V(x, c)$ rejects for every c

* **Note: P ⊆ NP** (V: ignore c and just run M on x)

What is P vs NP about?

Informally: Verifying an answer (given a hint/solution) seems much easier than figuring out the answer.

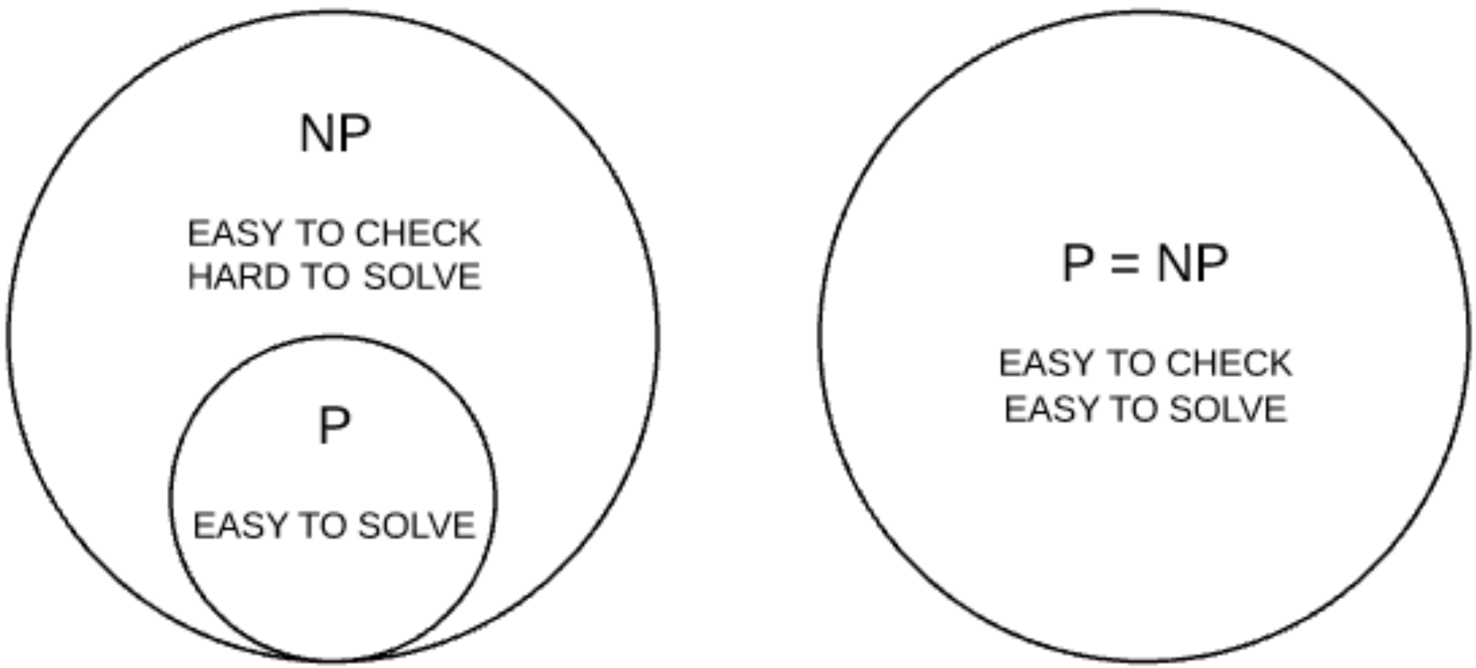
E.g. TSP, Ham-cycle, independent set, clique, subset sum … all lie in NP (but we don't know if they are in P)

Major open question (P vs NP?): Is NP strictly bigger than P?

Pictorially

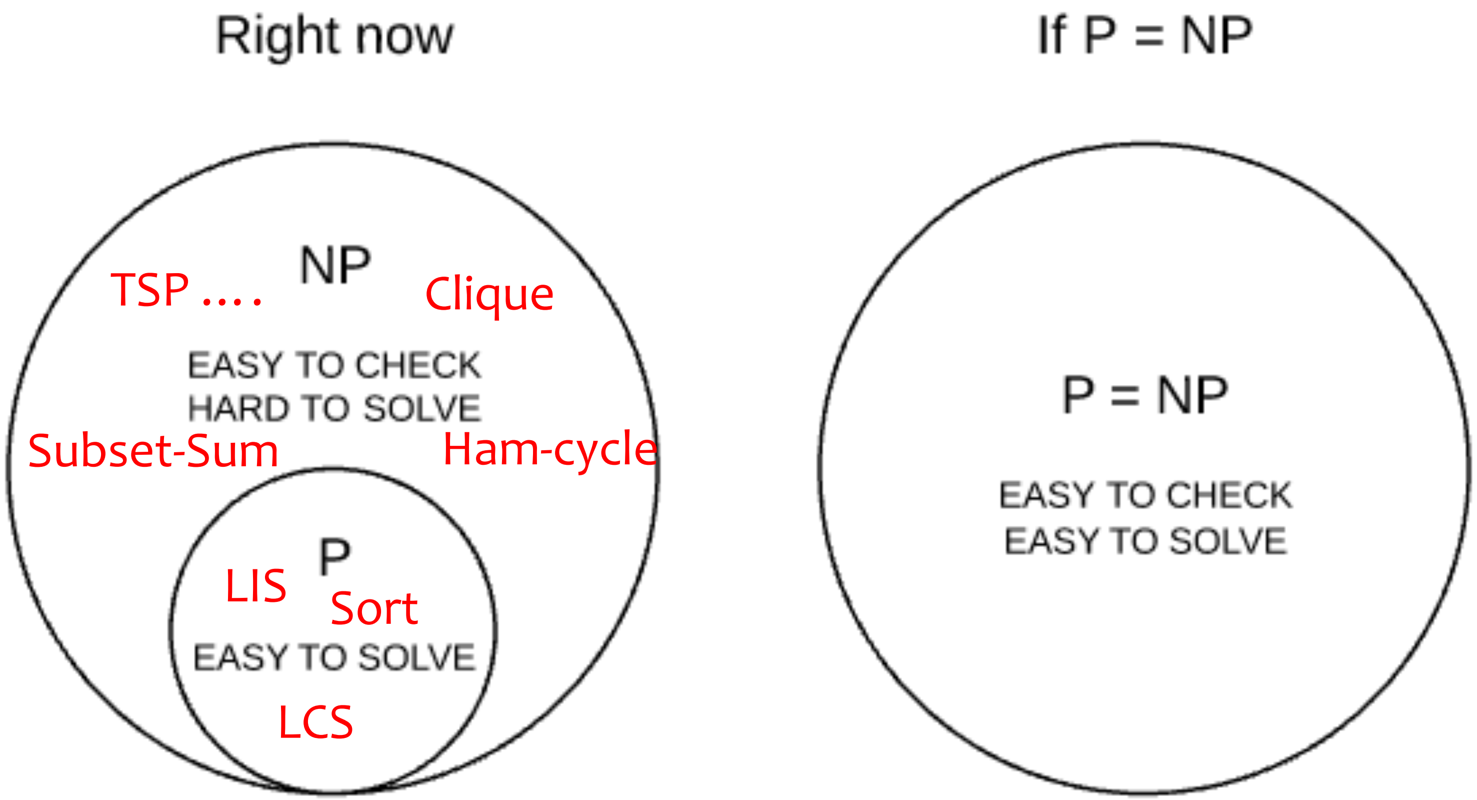
Right now

If P = NP



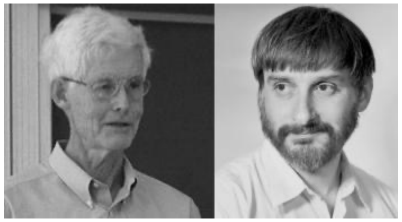
Pictorially

Problems beyond NP we won't study here



Two amazing results (got Turing awards)

Cook-Levin (1971): ***SAT is NP-hard.***



- SAT: stands for Satisfiability problem, will see next.
- NP-hard: means if SAT in P, then all of NP will be in P.

  (i.e. just need to show SAT in P to show P=NP)

Karp (1972): Actually TSP, Ham-cycle, clique, Subset Sum, … are "equivalent" to SAT. (next couple lectures)

(i.e. if any of them in P, then P=NP)

A "Hard" Language for NP

* **Informal Definition:** A language L is called **NP-Hard** if $L \in$ **P** implies that **NP** = **P**.

* **In other words:** A poly-time algorithm for L can be converted to yield poly-time algorithms for all efficiently verifiable languages! That is, for every language in **NP**.

Cook-Levin: ***SAT is NP-Hard***

Satisfiability Problem (SAT)

- Boolean ***variables*** x,y,z … taking values true or false (1 or 0)
- A Boolean ***literal*** is a variable (x) or its negation ($\neg x$ or $\overline{x}$)
- 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$\}$

EECS M ECE CSE

Satisfiability Problem (SAT)

- **Example 1:** $\phi(x,y) = \neg x \wedge y$
- **Question:** What is $\phi(1,0)$ and $\phi(0,0)$?
- **Example 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.)

(i.e., if ϕ is satisfiable, is there an efficiently verifiable certificate?)

Why is Satisfiability Important?

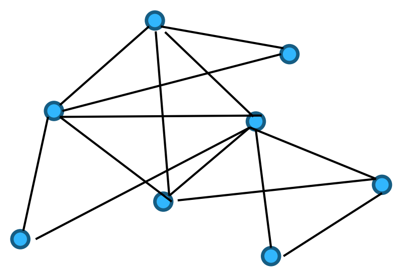
- **Theorem [Cook-Levin]:** SAT is NP-Hard.

- Let $L \in$ **NP** and let V be a verifier for L.
- **NP-Hard means:** If SAT $\in$ **P** then $L \in$ **P**
- **Proof Strategy:** Make an efficient algorithm for L
  - M on input x:
    - Construct formula $\phi_{V,x}$ (using the efficient algorithm of Cook-Levin)
    - Accept iff $\phi_{V,x} \in$ SAT (using the assumed efficient decider for SAT)

Goal: Proof (main idea) of Cook-Levin Theorem

First: A concrete example

k-clique problem: Given a graph G and an integer k, is there a clique of size k (a subset S of k vertices, so that every two vertices in S are adjacent)



Q: Is the problem in NP?

What is an efficiently verifiable certificate?

What does the verifier need to do?

1) check if every pair of vertices in S has an edge in G,
2) check that size of S is k.

Goal: Reduce finding clique certificate to SAT

Key insight:

Given graph G, the question of whether a certificate exists for k-clique can be reduced to solving a SAT instance.

Formally: Given instance G of k-clique,

design a formula Φ_G (in poly-time), s.t.

Φ_G is satisfiable iff G has a k-clique

Φ_G is unsatisfiable iff all cliques in G have size <k.

How?

Recall: Certificate = subset S of vertices
Verifier: (i) Check edge between each pair in S. (ii) Check that $|S| > k$

Let us view the certificate c as 0-1 string y of length n

(i) Formula for checking each pair in S has an edge.

$$\Phi_1 = \bigwedge_{(i,j)\ not\ an\ edge} \overline{y_i} \vee \overline{y_j}$$

(for each non-edge, one of its vertices is not in S)

(ii) Formula for checking $\sum_i y_i \geq k$

if k=1 $\Phi_2 = \vee_i\, y_i$ (union over y_i)

Can you think of a formula for k=2 ?

(the naïve idea does not scale well for general k, you get about n^k terms, but it is possible to construct shorter formulas using *logical circuits*)

EECS M ECE CSE

Summary: Finding clique certificate via SAT

Key insight:

Given graph G, the question of whether a certificate exists for k-clique can be reduced to solving a SAT instance.

Formally: Given instance G of k-clique design a formula Φ_G (in poly time), s.t.

Φ_G is satisfiable iff G has a k-clique

Φ_G is unsatisfiable iff all cliques in G have size $< k$.

$\Phi_G = \Phi_1 \wedge \Phi_2$

Φ_1 = Formula for checking each pair in S has an edge in G.

Φ_2 = Formula for checking $\sum_i y_i \geq k$

$\Phi_G \in SAT$ iff G has a clique of size k

Proving Cook-Levin Theorem

* **Theorem [Cook-Levin]:** SAT is NP-Hard.
* Let $L \in$ **NP** and let V be a verifier for L.

* **High-level Idea:** There is a poly-time algorithm that given a string x, constructs a Boolean formula $\phi_{V,x}$ such that:
  * $x \in L \Rightarrow \phi_{V,x}$ is satisfiable
  * $x \notin L \Rightarrow \phi_{V,x}$ is unsatisfiable

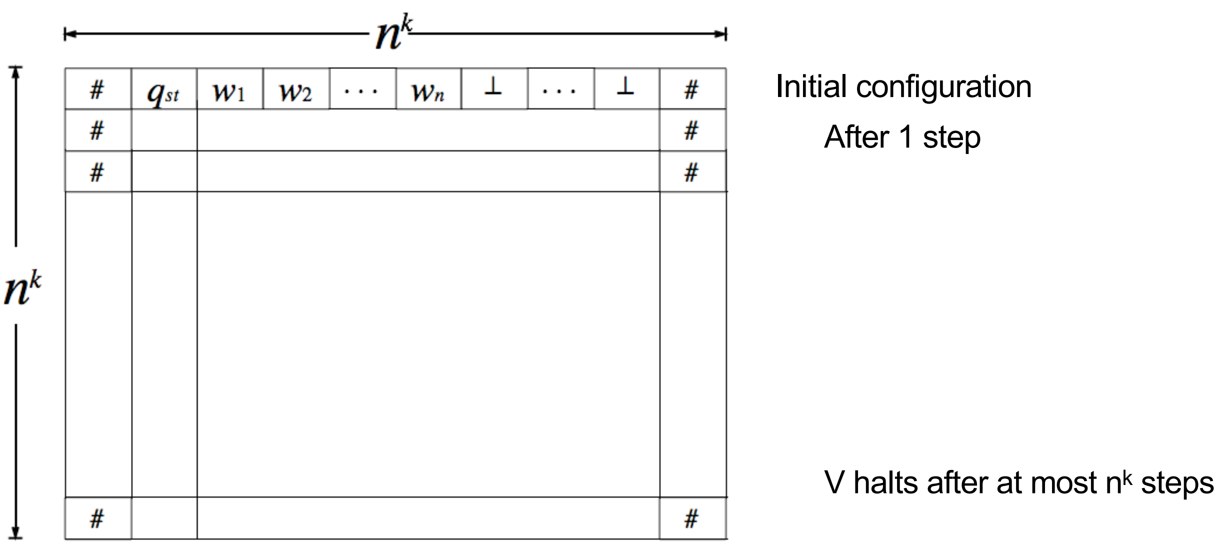
$\phi_{V,x}$ depends on the logic of V and on the instance x.

SAT is NP-Hard: Setup

* Let $V(x, c)$ be a verifier (i.e. a TM) for some $L \in \boldsymbol{NP}$.
* For every input x and a certificate c:
  * V makes at most $|x|^k$ steps (for some fixed k).
    $\Rightarrow V$ can affect only the first $|x|^k$ cells of the tape
* **Goal:** Design a Boolean formula that is satisfiable
  iff some certificate c causes $V(x, c)$ to accept in $|x|^k$ steps
* Turing machine engineering!
* **Definition:** A **configuration** of V represents the tape content of V, state of V, and location of V's head. **Example:** $011q_50001$:
  * V's tape content is $0110001\perp\perp\perp\ldots$
  * 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 # symbols)
  * Symbols can be from S = {0,1} ∪ Q ∪ {#, $, ⊥}
  * Successive rows correspond to configurations



Proof Overview

* Given an input x, construct a Boolean formula $\phi_{V,x}$ that represents every valid tableau such that
  * V accepts x for some certificate c $\Rightarrow \phi_{V,x} \in \text{SAT}$
  * V rejects x for all certificates c $\Rightarrow \phi_{V,x} \notin \text{SAT}$

* $\phi_{V,x} = \phi_{V,x,start} \wedge \phi_{V,x,cell} \wedge \phi_{V,x,accept} \wedge \phi_{V,x,move}$

1. ϕ_{start} enforces the ***starting configuration*** at the top line
2. ϕ_{cell} ensures that every cell contains ***exactly one*** symbol
3. ϕ_{accept} ensures that V reaches an ***accepting configuration***
4. ϕ_{move} ensures that each configuration ***follows from the previous configuration***, according to the code of *V*

The Starting Configuration

ϕ_{start} enforces the starting configuration

#	**q_0**	**x**	**$**	**c**	**⊥**	**⊥**	**#**

* Initial state q_0,
* Input x, $|x| = n$; certificate c, $|c| = m$
* $ - a special symbol that separates x and c
* **WE DO NOT KNOW c (!!)**, 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

cannot contain two different symbols

contains at least one symbol

every cell

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

What do these parts mean, in English?

Accepting Configurations

ϕ_{accept} ensures that V reaches an accepting configuration

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

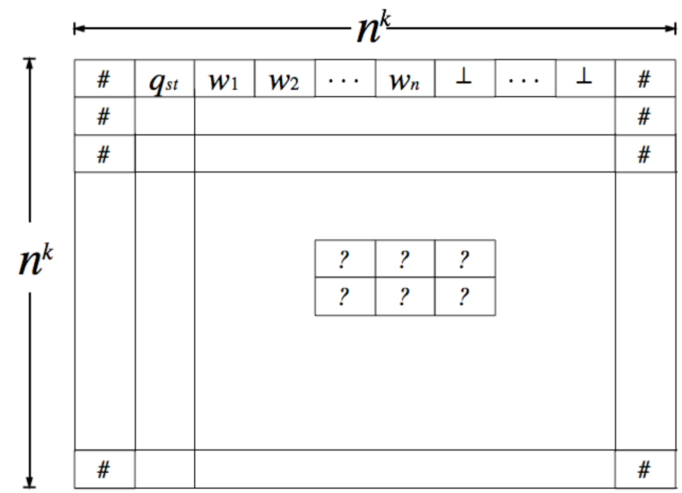
Logical Transitions

ϕ_{move} ensures that each configuration follows from the previous configuration according to the δ function

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

(Basically enforcing valid execution of V)

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



P=NP Conclusion

- **Conclusion: P** = **NP** iff there is an efficient algorithm for testing satisfiability of Boolean formulae.
- **Common *Belief*:** There is no efficient algorithm for deciding satisfiability, in which case **P** $\neq$ **NP**.