The complexity-class P

$$P = \bigcup_{k \geq 0} TIME(n^k)$$

The set of all decision problems solvable by a deterministic TM/algorithm in "polynomial" time

Informally: all decision problems that are efficiently solvable.

Example problems in P

- Every CFL is in P
  The CYK algorithm parses in $O(n^3)$-time in general
- Sort = $\{\langle S_1, S_2 \rangle \mid S_1$ is a list of integers S_2 is this list in increasing order$\}$
- RELPRIME = $\{\langle x, y \rangle \mid x, y \in \mathbb{Z}$ and x, y are relatively prime$\}$
  Extended Euclid's algorithm computes $gcd(x,y)$ in poly-time.

HAMPATH = $\{\langle G, s, t\rangle \mid$ G is a directed graph with a Hamiltonian path from s to t $\}$

Brute-force decider:

1. enumerate all paths $s \to t$
2. see if they are Hamiltonian

- No known Poly-time decider
- What about a verifier?

Def

A verifier for a language A is a TM V where

$\forall w \in A$ V accepts $\langle w, c\rangle$ for some string c.

* V is a "polynomial-time" verifier if it accepts in $O(|w|^k)$

implies that the certificate c is also short

NP Def #1

The complexity class NP is the class of languages / decision problems that have Poly-time verifiers.

P vs. NP

is it fundamentally easier to solve a problem than it is to recognize a solution.

$NTIME(t(n)) = \{L \mid \exists \text{ NTM } N \text{ that decides } L \text{ in } O(t(n)) \text{ time}\}$

NP Definition #2

$$NP = \bigcup_{k \geq 0} NTIME(n^k)$$

Show that NP definitions 1 and 2 are equivalent

#1 $\Rightarrow$ 2

Assume poly-time verifier V for A

Show $\exists$ NTM N

$N =$ "On input $\langle w \rangle$

1. Non-deterministically choose c s.t. $|c| = O(n^k)$
2. Run $V(\langle w, c \rangle)$ and accept iff V outputs 1"

$2 \Rightarrow 1$

Assume poly-time non-deterministic TM N

Show a V that verifies:

V: "On input $\langle w, c \rangle$:
1. Simulate $N(\langle w \rangle)$ using the choices given in c
2. Return 1 iff N ACCEPTS."

Point

- To show a decision problem $L \in P$ we must give a Poly-time algorithm ($O(n^k)$) for the problem.
- To show a decision problem $L \in NP$ we must give a poly-time verification algorithm for the problem.

CLIQUE = $\{\langle G,k\rangle \mid G$ is a graph that contains a k-clique$\}$

What is the certification?

the vertices in the clique

V = "On input $\langle\langle G,k\rangle, c\rangle$:

1. if $\exists$ edges in G that connect each vertex in c to every other vertex in c ACCEPT
2. Otherwise REJECT"

3SAT = $\{\langle \varphi \mid \varphi$ is a 3CNF formula that has a satisfying assignment$\}$

CNF: conjunctive normal form

$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_4 \vee x_3)$

What is the certification?

an assignment to each literal

Being in NP is no big distinction

But there are certain $L \in NP$ that are very important, because they "represent" or "express" all languages/problems in NP.

↳ represents means that all NP problems are reducible to these L.

$A \leq_p B$ means that A is reducible to B in polynomial time.

$\exists$ poly-time computable f s.t.

$$w \in A \iff f(w) \in B$$

if $A \leq_p B$ and $B \in P$, then $A \in P$

Proof

Let M_B be a poly-time decider TM for B

M_A = "On input w:

1. Compute $f(w)$
2. Run M_B on $f(w)$, ACCEPT iff M_B does."

The Cook-Levin Theorem

Theorem: SAT is NP-Complete

$SAT = \{ \phi \mid \phi \text{ is a satisfiable boolean formula} \}$

Claim $SAT \in NP$

The certificate is a satisfying assignment

$x_1, x_2 \ldots x_n$ s.t. $x_i \in \{0,1\}$

The verifier can check the satisfying assignment in poly-time (linear even)

Claim SAT is NP-Hard

$\forall L \in NP$, L is efficiently reducible to SAT

$L \leq_p SAT$

Idea:

Given NTM M_L for L. take a string w and produce a boolean formula $\phi_{M_L, w}$ that simulates the run of M_L on input w. An assignment to $\phi_{M_L, w}$ will be a satisfying assignment iff M_L accepts w.

Proof:

- Let M be a NTM that decides L in cn^k time.
- Define a tableau for M to be a $cn^k \times cn^k$ table whose rows are consecutive configurations of a computational path of M on input w.
- The first row is the starting configuration of M on w.
- each row must the previous by M's δ function
- A tableau is accepting if any of its rows is an accepting configuration.
- If M accepts w there exists an accepting tableau

$$\# q_0 w_1 w_2 \ldots w_n \sqcup \ldots \sqcup \#$$
$$\# \qquad\qquad\qquad\qquad \#$$

- Let's start by describing ϕ's variables.
  - let $C = Q \cup \Gamma \cup (\#)$
  - for each i, j between 1 and cn^k and $s \in C$ we have one boolean variable $x_{i,j,s}$
  - Each of the (cn^{2k}) entries $[i,j]$ of the tableau is called a cell.

    $x_{i,j,s} = 1$ iff cell $[i,j]$ has the value s.

- A valid accepting tableau must satisfy the following conditions
  1. Is each cell in the tableau legal? (i.e. contains only one value type)
  2. Is the starting configuration legal?
  3. Are the transitions between consecutive rows legal?
  4. Is there an accepting row?
- Since all 4 must be true we have

$$\phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$$

- Let's start with ϕ_{cell}
  - there must be exactly one value from C in each cell.

$$\phi_{cell} = \bigwedge_{i,j} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{s,t \in C} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right]$$

all cells — one variable is true — two symbols not in the same cell.

- ϕ_{start} - initial configuration is legal

$$\phi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \ldots$$

- ϕ_{accept} asserts that there is an accepting configuration

$$\phi_{accept} = \bigvee_{i,j \leq cn^k} x_{i,j,q_{accept}}$$

- ϕ_{move} makes sure each configuration results from the previous one legally.
  - ensure each window is legal
    - a window is a 2×3 cell block of a tableau

example

$$\delta(q_1, a) = \{(q_1, b, R)\}, \quad \delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$$

a	q_1	b
q_2	a	c

(a)

a	q_1	b
a	a	q_2

(b)

a	a	q_1
a	a	b

(c)

- If the top row is legal and every window is legal then each row in the table is a configuration that legally follows from the previous one.
  - not proven here

- ϕ_{move}
  - each window contains 6 cells that can be set only a constant number of ways.
    - the set of transitions is bounded by the δ function.

$$\phi_{move} = \bigwedge_{i,j \in [n^k]} (\text{window } [i,j] \text{ is legal})$$

- legal window

$$\bigvee_{\substack{\text{all legal} \\ \text{combinations}}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6} \right)$$

Complexity

- Clearly the running time of the reduction is proportional to ϕ
  - total number of variables

$$(cn^k)^2 \cdot |G| \in O(n^{2k})$$

  - size of $\phi_{cell} \in O(n^{2k})$
    - contains $|G|$ variables per cell

  - $\phi_{start} \in O(n^k)$

  - ϕ_{move} uses a constant number of variable per window

$$\phi_{move} \in O(n^{2k})$$

Def

A language/decision problem L is NP-Complete if

① $\forall\ L' \in NP \quad L' \leq_p L$ ← if you only have this L is NP-hard

② $L \in NP$

Theorem

if $L \in$ NP-complete and $L \in P$ then $NP \subseteq P \Rightarrow P = NP$

Cook-Levin Theorem

SAT is NP-Complete

the language of boolean satisfiability can represent any NP problem.

Basic intuition

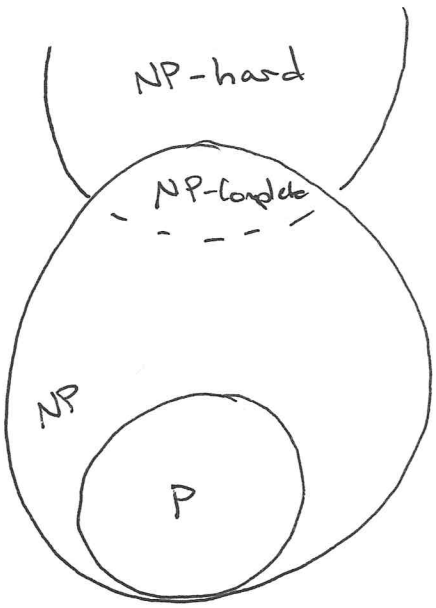
For any function $f : \{0,1\}^{\ell} \rightarrow \{0,1\}$

$\exists$ a CNF formula ϕ s.t. $\phi(u) = f(u)$

$\forall\ u \in \{0,1\}^{\ell}$

$$P \subseteq NP \subseteq \begin{matrix} PSPACE \\ = \\ NPSPACE \end{matrix} \subseteq EXPTIME$$

if $P \neq NP$



if $P = NP$

- To show $A \in P$ we must give an algorithm that finds a solution in polynomial time.
- To show $A \in NP$ we must give an algorithm that verifies a solution in polynomial time.

NP - Complete

In order for a problem L to be NP-complete 2 things must be true

1. $\forall L' \in NP$

   $L' \leq_p L$

2. $L \in NP$

Why is the NP-complete class of problems important?

if an NP-complete problem is reducible to some unknown problem the we have shown the unknown problem can only be in P iff P=NP

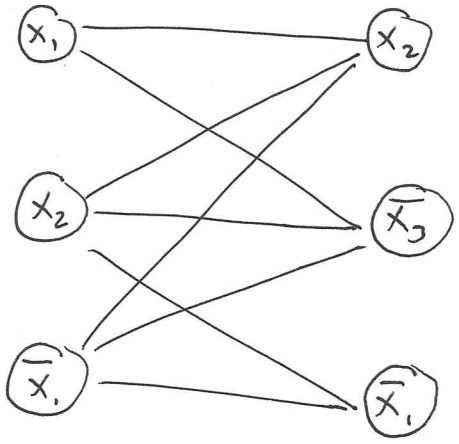
3SAT $\leq_p$ CLIQUE

Goal: turn a 3SAT instance ϕ into a $\langle G, k \rangle$ where G is a graph with a k-clique iff ϕ is satisfiable.

k = # of clauses in ϕ

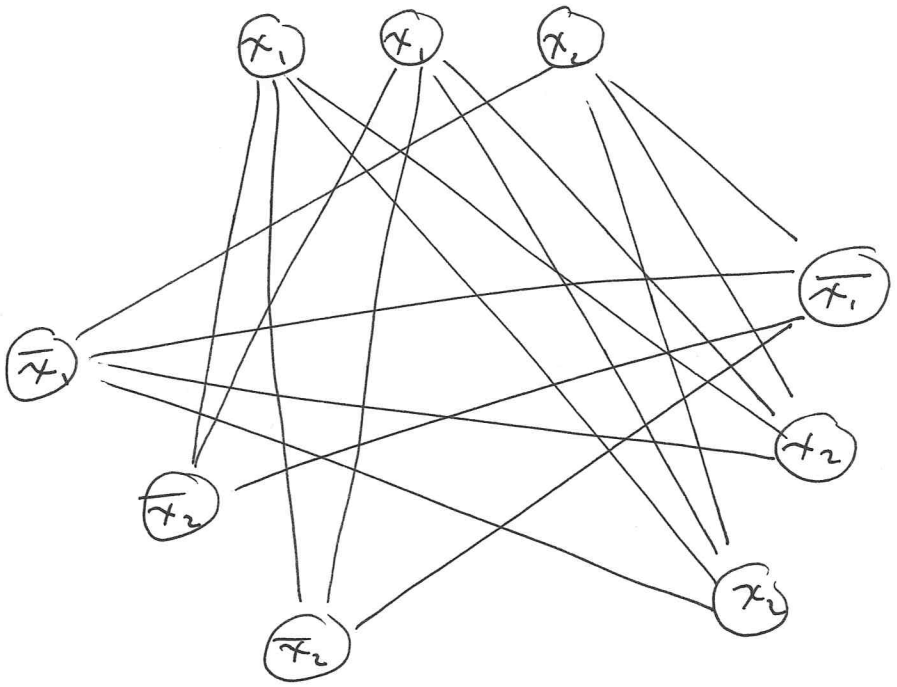
$$\phi = (x_1 \vee x_2 \vee \overline{x_1}) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_1})$$

map to a bipartite graph

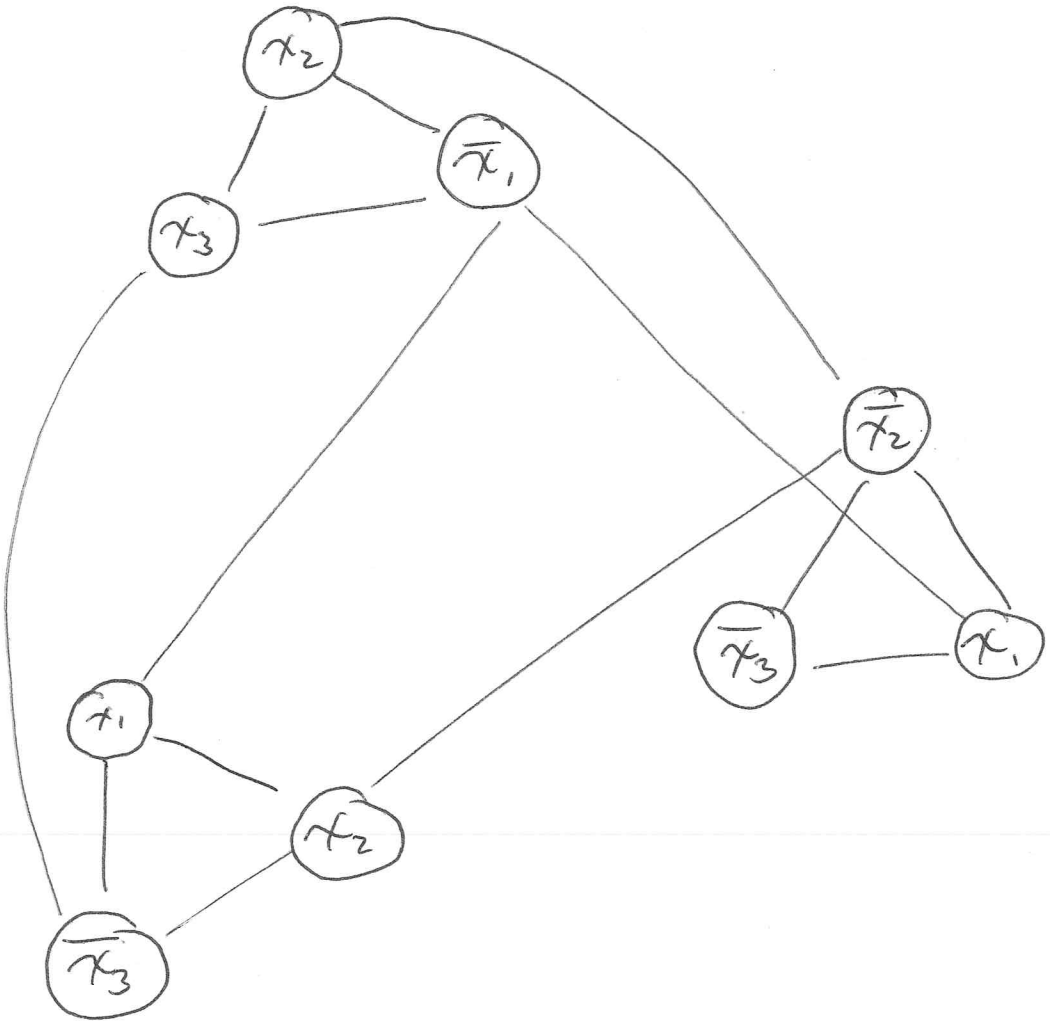


$A \leq_p B$ must be in polynomial-time

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee x_2)$$



$$(x_2 \vee x_3 \vee \overline{x_1}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_1) \wedge (x_1 \vee x_2 \vee \overline{x_3})$$



Independant set is NP-Complete

$IS = \{ \langle G, k \rangle \mid G$ is a graph containing an independant set of size $k \}$

$3SAT \leq_p IS$

Proof

- Given instance $\Phi = C_1 \wedge C_2 \wedge \dots \wedge C_n$ where C_i is the disjunction of 3 variables drawn from $x_1, x_2, \dots, x_n$ and their negations $\overline{x_1}, \overline{x_2}, \dots \overline{x_n}$
- Create graph G as follows
  - For each variable in each clause create a node labeled with the name of the variable
  - For each clause add an edge between the 3 nodes corresponding to the variables
  - Add an edge between ~~the~~ every pair of nodes with one labeled x_i and the other $\overline{x_i}$

- If Φ has a satisfying assignment then at least one variable in each clause is true.
- Let S be the independent set of size n
  - Define S by selecting one of the satisfying nodes in each clause gadget
- If there is an independent set of size n then there is a satisfying assignment.
  - at most one node in each clause is in S
  - Since S is independant, at most one node is from each clause and no x_i and $\overline{x_i}$ pair has been included.

TSP is NP-Complete

$HamPath \leq_p TSP$

- Let $G = (V, E)$ be an instance of HamPath
- Create $G' = (V, E')$

  where $E' = \{(i,j) \mid i,j \in V \text{ and } i \neq j\}$

- Define the cost function as

$$c(i,j) = \begin{cases} 1 \text{ if } (i,j) \in E \\ 2 \text{ if } (i,j) \notin E \end{cases}$$

- Suppose a Hamiltonian Cycle exists

  then a solution to TSP will have weight of $|V|$

BPP Def

Bounded-Error Probabilistic Polynomial-Time

Def #1

The class of Decision problems solvable by a probabilistic Turing Machine in poly-time with an error probability bounded away from $1/3$ in all instances.

Def #2

The class of decision problems solvable by a Nondeterministic Turing Machine such that

- if the answer is 'yes' then at least $2/3$ of the paths accept
- if the answer is 'no' then at most $1/3$ of the paths accept

BPP

often identified as the class of feasible problems for a computer with access to a genuine random-number source.

2/3 is completely arbitrary

The idea is that there is a probability of error, but if the algorithm is run many times, the chance the majority of the runs are wrong drops off exponentially.

Chernoff bounds

P ⊆ BPP

why?

Many problems were known to be in BPP but not known to be in P. The number of such problems is decreasing.

The conjecture is now BPP = P (still unproven)

Primality testing was known to be in BPP for years before 2002 AKS primality test showed it to be in P.

$P \subseteq BPP$

$BPP \overset{?}{\subset} NP$

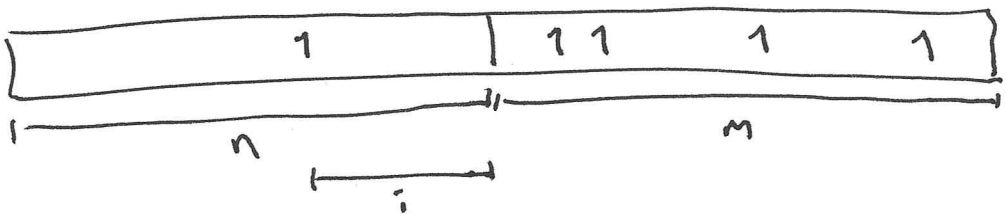
$NP \overset{?}{\subset} BPP$

$BPP = NP$

SubsetSum = $\{ \langle L, k \rangle \mid L$ is a list of integers that contains a subset that sums to $k \}$

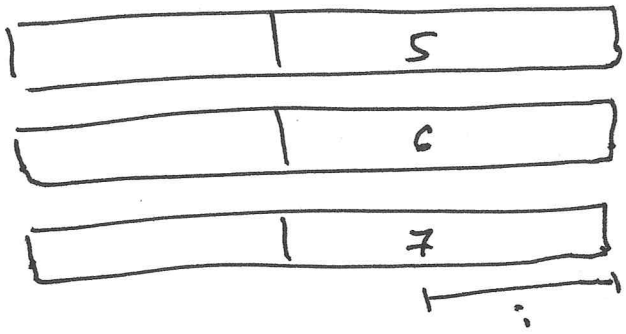
3SAT $\leq_p$ Subset Sum

- let ϕ be an instance of 3SAT with m clauses and n variables.
- Construct L to contain $3m + 2n$ values each is of length $m + n$ in base-32
  - integers for ~~variables~~ literal



  - 1 in the ith position if this integer represents x_i or $\overline{x_i}$
  - 1's in all positions where this literal makes the clause true.
  - 0 in all other positions

- Integers for Clauses



- for the i^{th} Clause

Example: C_1 C_2

$$(x_1 \vee y_2 \vee z_3) \wedge (x \vee \bar{y} \vee \bar{z})$$

$m = 2$ $n = 3$

$x = 00111$

$\bar{x} = 00100$

$y = 01001$

$\bar{y} = 01010$

$z = 10001$

$\bar{z} = 10010$

$C_1 = 00005$
00006
00007

$C_2 = 00050$
00060
00070

Budget

$$K = 8\left(1 + 32 + 32^2 + \dots + 32^{m-1}\right) + 32^c\left(1 + 32 + 32^2 + \dots + 32^{n-1}\right)$$

base 32 ensures no carries between positions.

8 for each clause position and a 1 for each variable position

- Among all integers the sum of digits in the position for a variable is 2.
- For clauses it is $1+1+1+5+6+7=21$
- The budget must be satisfied for a per-digit basis.

- If the set matches the budget it must contain exactly one of x and $\bar{x}$
- For each clause at least one of the integers for literals must have a 7 there so we can choose 5, 6, or 7 to make 8 in that position.

- Each integer can be constructed from ϕ in linear time
  - total reduction is $O((n+m)^2)$
- If ϕ is satisfiable take a satisfying assignment A
  - pick integers for those literals that makes A true

- the selected integers sum to between 1 and 3 for each clause
- For each clause choose the integer with 5, 6, or 7 in that digit to make the sum 8
- These selected integers sum to exactly the budget

- A sum of integers equal to k implies ϕ is satisfiable

Partition Problem

Partition = $\{ \langle S \rangle \mid S$ is a multi set that can be partitioned in S_1 and S_2 such that $\sum_{s \in S_1} s = \sum_{s \in S_2} s \}$

SubsetSum $\leq_p$ Partition

- given instance $\langle L, k \rangle$ of subsetSum compute the sum s of all integers in L.
  - linear
- Output L followed by 2 integers s and 2k
- Example L = 3, 4, 5, 6 k = 7

  Output = 3, 4, 5, 6, 14, 18
- Sum of all integers in output is

  $2(s + k)$
  - each partition must sum to $s + k$
- If the input has a subset that sums to k pick it plugs S to solve the output