P vs. NP Question

Lecture outline

- Basic definitions:
  - P, NP complexity classes
  - the notion of a certificate.
- reductions, various types:
  - Karp, Cook and Levin.
- Search vs Decision problems
  - self-reducibility.
- NP-Complete languages and relations
  - NP Complete relation and self-reducibility.
- Introducing some NP-Complete problems.
- Showing R_{SAT} is NP-hard.

The P, NP complexity classes

Def: A Decision problem for a language $L \subseteq \{0,1\}^*$ is to decide whether a given string x belongs to the language L.

Def: P is the class of languages (decision problems) that can be recognized by a deterministic polynomial time Turing machine.

Def: NP is the class of languages that can be recognized by a non-deterministic polynomial-time Turing-machine.

Polynomially Verifiable Relations

Def: A binary relation R is polynomially bounded if:

$$(x,y) \in R \Rightarrow |y| \leq |x|^{o(1)}$$

Def: R is polynomial-time-decidable if the corresponding language

$$L_R = \{ (x,y) : (x,y) \in R \}$$

is in P.

Def: A relation that is both polynomially-bounded & polynomial-time-decidable is polynomially-verifiable.

NP – Alternative Definition

Def: L is an NP language if there is a polynomially-verifiable relation R_L s.t.

$$X \in L \Leftrightarrow \exists \text{ w for which } (x,w) \in R_L$$

w is a witness or a certificate.

Given a witness, membership can be verified in polynomial time.

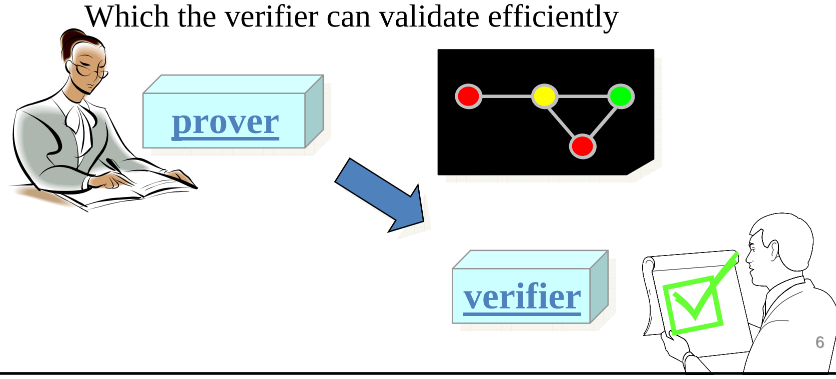
P=NP implies: $\exists$ witness $\Rightarrow$ can be found efficiently

The NP game

The notion of a certificate can be thought of a game between a prover and a verifier

Given input X   The all powerful prover sends a certificate for membership of X

Which the verifier can validate efficiently



NDTM $\Rightarrow$ PolyVerRel

1.2.1

- Assume M_L is a non-deterministic TM that decides L within $P_L(|x|)$ steps
- Let R_L be the set of all pairs (x,y) s.t.
  - x is an input to M_L
  - Y is an accepting, legal computation of M_L on x
- R_L is polynomially-verifiable
- $X \in L \Leftrightarrow \exists$a computation y s.t. $(x,y) \in R_L$

PolyVerRel $\Rightarrow$ NDTM

- Let R_L be the relation for L, decided by a deterministic TM M^*_L
- Def a non-deterministic TM M_L:
  - Guess y of proper polynomial size
  - Call M^*_L to check if $(x,y) \in R_L$
  - Accept x if M^*_L accepts (x,y)
- M_L is a non-deterministic TM for L

Search Problems

- Def: A search problem over a binary relation R finds, given x, a string y s.t. $(x,y) \in R$.
- Given a polynomially-verifiable relation R, define:

$$L(R) = \{\ x : \exists y \text{ s.t. } (x,y) \in R\ \}$$

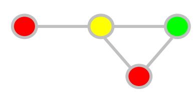
Clearly, finding a solution can only be harder than just deciding whether it exists.

Note: NP = { L(R) : R is polynomially-verifiable }

Example

Problem: 3-coloring graphs

Instance: An undirected graph G=(V,E)



Corresponding relation:

$$R_{3COL} = \{\ (G,\varphi) : \varphi \text{ is a legal 3-coloring of } G\ \}$$

Decision problem: decide the language $L(R_{3COL})$, namely whether G is 3-colorable.

Search problem: find a 3-coloring of G.

Reductions

2.1

- The purpose of a reduction is to show that some problem is at least as hard as some other problem.

- If problem A reduces to problem B, then solving B implies solving A.

B is at least as hard as A, denoted **A ≤ B**

Cook Reduction

Def: An oracle for a problem Π is a magical apparatus that, given an input x to Π, returns $\Pi(x)$ in a single step.

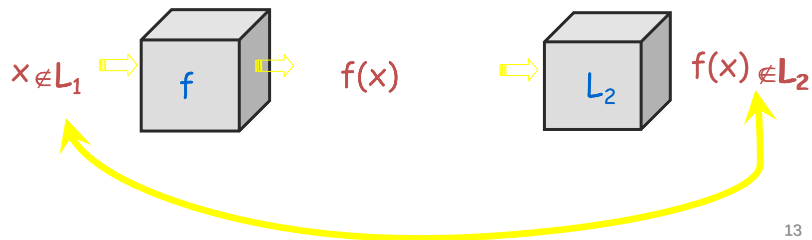
Def: A Cook reduction from problem Π_1 to problem Π_2 is a polynomial-time TM for solving Π_1 on input x utilizing an oracle for Π_2.

Denoted $\Pi_1 \leq_{cook} \Pi_2$.

Karp Reduction

Def: A Karp reduction of L_1 to L_2 is a polynomial-time--computable function f s.t.

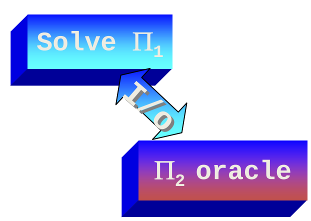
$$x \in L_1 \Leftrightarrow f(x) \in L_2$$



Karp vs. Cook reductions

Cook reduction allows calling the oracle polynomially many times

Karp reduction allows only one call to the oracle, and only at the end of the computation.



- Cook reduction is stronger: given Karp reduction f,
  - On input x compute the value f(x).
  - Present f(x) to the oracle, and output its answer.
- There are (few) examples where a Cook reduction is known, while a Karp reduction is unknown.

Levin Reduction

2.1

Def: a Levin reduction from R_1 to R_2 is 3 poly-time-computable functions f,g,h s.t.

- $x \in L(R_1) \iff f(x) \in L(R_2)$
- $(x,y) \in R_1 \Rightarrow (f(x), g(x,y)) \in R_2$
- $(f(x),z) \in R_2 \Rightarrow (x, h(x,z)) \in R_1$

f translates inputs of the first problem to inputs of the second problem, g & h transform certificates of one to the other.

! Note: A Levin reduction implies Cook / Karp reductions of the corresponding search / decision problems.

Properties of Reductions

2.1

Claim: All reductions are transitive:
A reduces to B, B to C $\Rightarrow$ A reduces to C

Claim: Cook reduction preserves poly-time-computability

So do Karp & Levin reductions

Proof: assume Π_1 Cook-reduces to the poly-time-comp problem Π_2, and M is the reduction algorithm.

- Π_2-oracle can be simulated by a poly-time TM
- Replacing oracle queries in M by the simulation - we get a poly-time TM that solves Π_1.

Search vs. Decision Problem

1.4

Recall: for poly-time verifiable relation R,

$L(R) = \{ x : \exists y \text{ s.t. } (x,y) \in R \}$

Def: A relation R is called self-reducible if solving the search problem for R is Cook-reducible to deciding the language L(R).

The search problem can be solved using the decision problem

An Example: 3-SAT

Input: A CNF formula φ with n variables.

Task: find $\sigma : \{1,..,N\} \to \{0,1\}$ such that

$\varphi (\sigma(1), \ldots, \sigma(n)) = \text{True}$

Corresponding relation:

$R_{SAT} = \{ (\varphi, \sigma) : \varphi (\sigma(1), \ldots, \sigma(n)) = T \}$

$SAT = L(R_{SAT})$

Note: Self-reducibility is a property of a relation, not a language.

[There are many relations R for which SAT = L(R).]

R_{SAT}

1.4.1

<u>Thm</u>: R_{sat} is Self-Reducible.

<u>Proof</u>: Assuming an oracle O for the language SAT = $L(R_{SAT})$. solving the <u>search problem</u>:

Query O whether $\varphi \in$ SAT. If not - stop.

- For k := 1 to n:
  - $\varphi_K(x_{k+1}, .., x_n) := \varphi(\sigma_1, .., \sigma_{k-1}, 1, x_{k+1}, .., x_n)$
  - If $\varphi_k \in$ SAT (Query O), $\sigma_k = 1$
    else $\sigma_k = 0$.

formula obtained by replacing $x_1, .., x_k$ with $\sigma_1, .., \sigma_{k-1}, 1$

- $\sigma(1)=\sigma_1, \ldots, \sigma(n)=\sigma_n$ satisfies φ!

Self-reducibility of SAT

<u>Given</u>: $(\neg x \vee \neg y) \wedge (\neg x \vee z)$

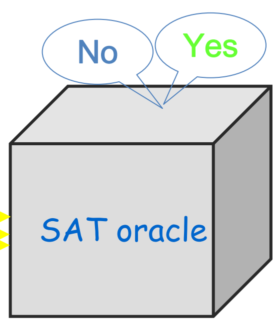
$(\neg x \vee \neg y) \wedge (\neg x \vee z)$ ✓

x = 1: $(\neg 1 \vee \neg y) \wedge (\neg 1 \vee z)$ ✓

y = 1: $(\neg 1 \vee \neg 1) \wedge (\neg 1 \vee z)$ ☒

y = 0: $(\neg 1 \vee \neg 0) \wedge (\neg 1 \vee z)$

z = 1: $(\neg 1 \vee \neg 0) \wedge (\neg 1 \vee 1)$ ✓



Another Example: GI

Graph Isomorphism: given two (simple) graphs, are they isomorphic?

Natural relation: R_{GI} contains all ((G1,G2),π) s.t. π is an isomorphism between G_1 and G_2.

Unlike SAT, GI is not known to be NP-Complete

in fact GI is unlikely to be NP-hard, as we'll see later in the course

R_{GI}

1.4.2

Thm: R_{GI} is Self-Reducible.

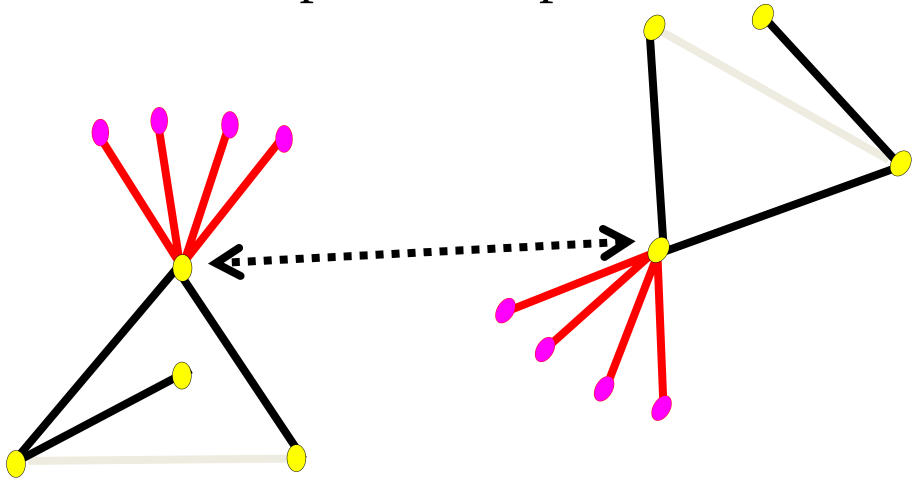
Proof: construct π piecemeal, 1 vertex at a time

check if $u \in G_1$ can be mapped by π to $v \in G_2$:

- Connect both v and u to new n leaf-vertices to obtain 2 new graphs, G'_1 & G'_2
- If G'_1 isomorphic to G'_2 - u must be mapped to v
- Iterating this - deleting matching vertices - determines the isomorphism, a vertex at a time

u and v are distinguished from other vertices so any isomorphism must map u to v

Graph Isomorphism



Non Self-Reducibility

- There are many non-self-reducible relations, but it is hard to find an NP language, whose "natural" relation is non-self-reducible.
- $L_{COMP} = \{ N \ : \ N = n1 \cdot n2 \}$ is poly- time decidable via a <u>randomized</u> algorithm. deterministic!
- The natural choice is

$$R_{COMP} = \{ (N, (n1,n2)) \ : \ N = n1 \cdot n2 \}$$

- It is widely believed the <u>search</u> of R_{COMP} is not poly-time-comp (factoring), which would imply it is not self-reducible (by a random algorithm)

NP-completeness

Def: A language L is NP-Complete if:

1. $L \in NP$.
2. $\forall L' \in NP$: $L' \leq_{Karp} L$.

Generalize: $L' \leq_{Cook} L$

Def: A relation R is NP-Complete if:

1. $L(R) \in NP$.
2. $\forall R'$ s.t. $L(R') \in NP$: $R' \leq_{Levin} R$.

NP-completeness & Self-Reducibility

2.2

Thm: For every relation R,

R is NP-Complete $\Rightarrow$ R is self-reducible.

Proof: Let R be an NP-complete relation.

R_{SAT} is NP-hard under Levin reduction (to be proven later), namely, there is a Levin reduction (f,g,h) from R to R_{SAT}

Since R is NP-complete, there exists a Karp reduction k from SAT to L(R).

NP-completeness & Self-Reducibility

Proof (continue): an algorithm that finds y s.t. $(x,y)\in R$, using the Levin & karp reductions:

1. Query $L(R)$'s oracle whether $x\in L(R)$
2. If "no", announce: $x\notin L(R)$
3. If "yes", translate x into a CNF formula $\varphi=f(x)$ [using Levin's f function]
4. Compute a satisfying assignment $(\sigma_1,\ldots,\sigma_n)$ for φ
5. Translate $(\sigma_1,\ldots,\sigma_n)$ to a witness $y=h(x,(\sigma_1,\ldots,\sigma_n))$ [Using Levin's h function]

$x\in L(R) \Rightarrow f(x)\in L(R_{SAT})$

show later

$f(x,z)\in L(R_{SAT}) \Rightarrow (x,h(x,z))\in L(R)$

NP-completeness & Self-reducibility

Proof (continue): given a partial assignment $(\sigma_1,\ldots,\sigma_i)$, compute a satisfying assignment $(\sigma_1,\ldots,\sigma_n)$ for φ,

- Trying to assign $x_{i+1}=1$:
- check $L(R)$'s oracle to see if $\varphi(\sigma_1,\ldots,\sigma_i,1,x_{i+2},\ldots,x_n)$ is satisfiable
  [By translating the CNF formula $\varphi(\sigma_1,\ldots,\sigma_i,1,x_{i+2},\ldots,x_n)$ to the language $L(R)$, using the Karp function k]
- If the oracle answers "yes" assign $\sigma_{i+1}=1$, otherwise assign $\sigma_{i+1}=0$
- Iterate until $i=n$.

$SAT\leq_{Karp}L(R)$ and since $L(R)$ is NP-complete
$\varphi\in SAT \Leftrightarrow k(\varphi)\in L(R)$

same as in self-reducibility of SAT. instead of SAT's oracle, use L(R)'s oracle

NP-completeness & Self-reducibility

Given: $(\neg x \vee \neg y) \wedge (\neg x \vee z)$

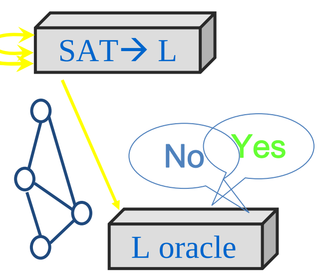
$(\neg x \vee \neg y) \wedge (\neg x \vee z)$ ✓

$x = 1$: $(\neg 1 \vee \neg y) \wedge (\neg 1 \vee z)$ ✓

$y = 1$: $(\neg 1 \vee \neg 1) \wedge (\neg 1 \vee z)$ ☒

$y = 0$: $(\neg 1 \vee \neg 0) \wedge (\neg 1 \vee z)$

$z = 1$: $(\neg 1 \vee \neg 0) \wedge (\neg 1 \vee 1)$ ✓



Bounded-Halting

2.3

Two Equivalent Definitions:

BH $\equiv$ { $\langle <M>,x,1^t \rangle$ | $<M>$ is the description of a non-deterministic TM that accepts input x within t steps. }

BH $\equiv$ { $\langle <M>,x,1^t \rangle$ | $<M>$ is the description of a deterministic TM, and $\exists y$ s.t. $|y| \leq |x|^{O(1)}$ and M accepts (x,y) within t steps }

Bounded-Halting Cont.

Def: Bounded-Halting Relation

$R_{BH} \equiv \{ (\langle <M>,x,1^t \rangle, y) \mid <M>$ is the description of a deterministic machine, which accepts input (x,y) within t steps $\}$

Note: The length of y is bounded by t,

! therefore it is polynomial in the length of the input $\langle <M>,x,1^t \rangle$.

Bounded-Halting is NP-Complete

Claim: BH is NP-Complete.

Proof:

- $BH \in NP$ (immediate from definition)
- Any L in NP, Karp-reduces to BH:
  Let L be in NP, then:
  - There exists a poly-time verifiable witness-relation R_L, recognized by M_L.
  - M_L accepts every (x,y) in $p(|x|)$ steps
- The reduction transforms x to $\langle M_L, x, 1^{p(|x|)} \rangle$

Bounded-Halting is NP-complete

Proof (continue):

$X \in L \Leftrightarrow$ Exists a polynomially bounded witness y such that $(x,y) \in R_L$

$\Leftrightarrow$ Exists a polynomial time computation of M_L accepting (x,y)

$\Leftrightarrow \langle M_L, x, 1^{p(|x|)} \rangle \in BH$

$(x,y) \leftrightarrow (\langle M,x,1t \rangle,y)$

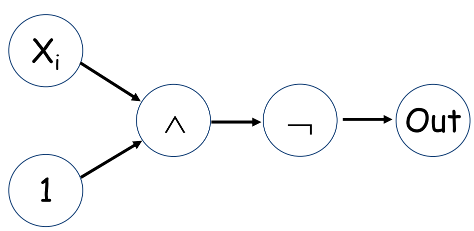
Note: The reduction can be transformed into Levin reduction of R_L to R_{BH} with the identity function supplying the two missing functions.

Circuit-Satisfiability

2.4

Def: a circuit is a directed acyclic graph G=(V,E) with vertices labeled output,$\wedge,\vee,\neg,x_1,\ldots,x_m$,0 s.t.

- A vertex labeled x_i has in-degree 0
- A vertex labeled 0 (or 1) has in-degree 0
- The in-degree of vertices labeled $\wedge$, $\vee$ is 2 (bounded fan-in)
- A vertex labeled $\neg$ has in-degree 1
- $\exists$a single sink (of out-degree 0), of in-degree 1, labeled "output"



Circuit-Satisfiability

Given an assignment $\sigma \in \{0, 1\}^m$ to the variables $x_1, \ldots, x_m$, $C(\sigma)$ will denote the value of the circuit's output

Def: Circuit-Satisfiabilty

$CS \equiv \{\text{circuit } C : \text{exists } \sigma \text{ s.t. } C(\sigma)=1\}$

$R_{CS} \equiv \{(C, \sigma) : C(\sigma)=1\}$

The value is defined by setting the value of each vertex to the natural value imposed by the Boolean operation it is labeled by

CS is NP-complete

2.4,2.6

Claim: CS is NP-Complete.

Proof:

- $CS \in NP$
  - R_{CS} is polynomialy-bounded: the witness σ is an assignment - it has one bit for each x_i
  - Given a pair (C, σ) evaluating one gate takes $O(1)$ steps, therefore total evaluation time is polynomial in $|C|$.

CS is NP-complete

Proof (continue):

- CS is NP-hard (show a reduction from BH):

Given $\langle M,x,1^t \rangle$ the computation of M can be fully described by a $t \times t$ matrix in which entry (i,j) is:

- The content of cell j at time i (constant)
- An indicator to whether the head is on cell j at time i (1 bit).
- In case the head is indeed there: the state of the machine ($O(\log |M|)$).

Each row of the matrix corresponds to a configuration of the machine

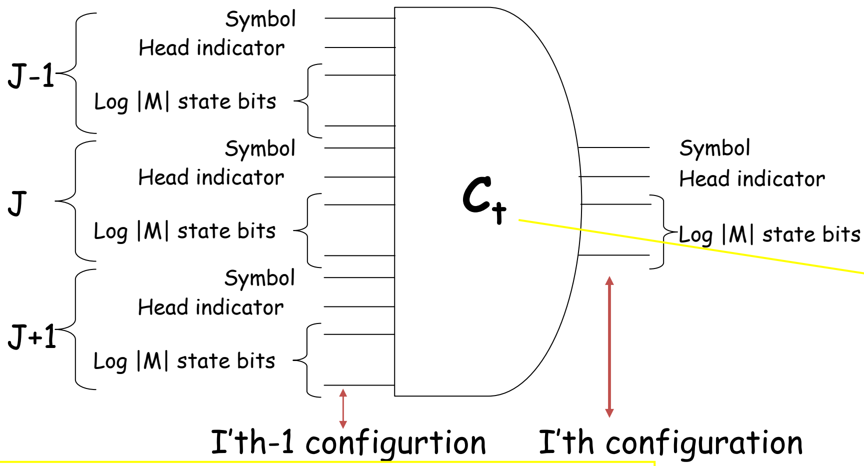
CS is NP-complete

Proof (continue):

- The transition between following configurations can be simulated by a series of t circuits C_t as shown later.
- The reduction constructs a circuit C' from the $t \times t$ C_t circuits, where the i^{th} series is the input of the i+1 series.
- The input to the circuit is built of |x| vertices corresponding to the input x, and t-|x| "variable vertices".
- Finally, we add to C' the ability to identify that a certain configuration has reached accepting state, so that the circuit will output 1.
- It is shown that the whole circuit can be built using $O(n^5)$ gates.

The circuit C_t

- The j'th triple in the i'th configuration, is determined by the j-1, j, j+1 triples in the i-1 configuration. This can be described by boolean functions implemented by C_t.



It is shown that C_t can be described in $O(n^3)$ gates

CS is NP-complete

<u>Proof</u> (continue):

- The accepting states can be encoded into a Boolean function, which will return 1, on input an accepting state. The output of the entire circuit is an OR over all the state outputs of all C_t circuits. This can be described by $O(n^2 \log n)$ gates.

R_{SAT} is NP-hard

2.5

Claim:

R_{SAT} is NP-hard under Levin reduction.

Proof:

Since Circuit-satisfiability is NP-Complete it suffices to show a reduction from R_{CS} to R_{SAT}:

The reduction maps a circuit C to a CNF formula φ_C, and an input y for the circuit to an assignment y'*to the formula and vice versa.

R_{SAT} is NP-hard

Proof (continue): Mapping circuit C to CNF formula φ_C

- Every vertex of the circuit-graph is mapped into a variable.
- For every such variable V_i we define a CNF formula φ_i that forces the variable to have the same value as the gate represented by V_i
- We define $\varphi_c = \varphi_1 \wedge \ldots \wedge \varphi_n$

R_{SAT} is NP-hard

Proof (continue): mapping a gate to a formula:

- For a $\neg$ vertex v with an in-edge from u:

  $\varphi_i (v, u) = (u \vee v) \wedge (\neg u \vee \neg v)$
- For a $\vee$ vertex v with in-edges from u, w:

  $\varphi_i (v, u, w) = (u \vee w \vee \neg v) \wedge (u \vee \neg w \vee v) \wedge (\neg u \vee w \vee v) \wedge (\neg u \vee \neg w \vee v)$
- For a $\wedge$ vertex v with in-edges from u, w:

  $\varphi_i (v, u, w) = (u \vee w \vee \neg v) \wedge (u \vee \neg w \vee \neg v) \wedge (\neg u \vee w \vee \neg v) \wedge (\neg u \vee \neg w \vee v)$
- For the vertex marked ***output*** with an in-edge from u:

  $\varphi_{output}(u) = u$

R_{SAT} is NP-hard

Proof (continue):

- $C \in CS \Leftrightarrow \varphi_c \in SAT$: given an assignment to φ_c it's easy to construct an assignment to C, and vice versa
- Note: The reduction is poly-time - the size of the CNF formula φ_c is linear in the size of C, therefore, it can be constructed in polynomial time

Bibliographic Notes

- Lecture notes for a course by Oded Goldreich.
- M. Bellare and s. Goldwasser, “The Complexity of Decision vs Search”
- M. Sipser, “The History and Status of the P vs NP Problems”
- M. Sipser, “Introduction to the Theory of Computation”