Introduction to Complexity

CS 336: Design and Analysis of Algorithms.
© Konstantin Makarychev

Hard Problems

- $f: \{0,1\}^* \to \mathbb{N}$ is a function; $\{0,1\}^*$ are all $0/1$-words.
- n is the size of the input i.e. $n = |x|$.
- **Def:** Program A computes f if $A(x) = f(x)$ for all x.
- Is there a function f such that the running time of any program A computing f is

$$\gg 2^n, \gg 2^{2^n}, \gg 2^{2^{2^n}}$$

(for some $x \in \{0,1\}^*$)

Hard Problems

- $f: \{0,1\}^* \to \mathbb{N}$ is a function; $\{0,1\}^*$ are all $0/1$-words.
- n is the size of the input i.e. $n = |x|$.
- **Def:** Program A computes f if $A(x) = f(x)$ for all x.
- Is there a function f such that the running time of any program A computing f is

$$\gg 2^n, \gg 2^{2^n}, \gg 2^{2^{2^n}}$$

(for some $x \in \{0,1\}^*$)

Yes: $f(x) = 2^{2^{2^{2^{|x|}}}}$

Hard Problems

- $f: \{0,1\}^* \to \{0,1\}$ is a function or decision problem.
- $L \subset \{0,1\}^*$ is a language.

$$f(x) = 1 \iff x \in L$$

- **Def:** Program A *recongizes* L if

$$A(x) = \begin{cases} 1, x \in L \\ 0, x \notin L \end{cases}$$

- Is there a language L such that the running time of any program recognizing L is

$$\gg 2^n, \gg 2^{2^n}, \gg 2^{2^{2^n}}$$

Hard Problems

- There exist undecidable languages/functions/problems:
  - language L is *decidable* if there exists a program that recognizes L; and
  - language L is *undecidable*, overwise.
- There exist undecidable languages:

Hard Problems

- There exist undecidable languages/functions/problem:
  - language L is *decidable* if there exists a program that recognizes L; and
  - language L is *undecidable*, overwise.
- There exist undecidable languages:
  - Number of languages = number of subsets of natural numbers = number of real numbers.
  - Number of programs = number natural numbers.

number natural numbers < number of real numbers

Hard Problems

- There exist undecidable languages/functions/problems:
  - language L is *decidable* if there exists a program that recognizes L; and
  - language L is *undecidable*, overwise.
- Halting problem is undecidable:

$$\text{HALT} = \{A\colon\ program\ A\ terminates\}$$

- Why?

Halting Problem

Given: a program A
Goal: decide whether A terminates

Proof Idea:

This statement is true if and only it is false.

$$f(x) = \begin{cases} 1, \text{if } f(x) = 0 \\ 0, \text{if } f(x) \neq 0 \end{cases}$$

Halting Problem

Given: a program A
Goal: decide whether A terminates

Proof Idea:

Program A
Let $a = \text{run}\ (A)$;

Return $\begin{cases} 1, \text{if } a = 0 \\ 0, \text{otheriwse} \end{cases}$

Halting Problem

Given: a program A
Goal: decide whether A terminates

Proof Idea:

Program A
  Let $a = \text{run}\ (A)$;
  Return $\begin{cases} 1, \text{if } a = 0 \\ 0, \text{otheriwse} \end{cases}$

Then, $A() \neq A()$? Contradiction? No, because $A()$ is not defined.

Halting Problem

Given: a program A
Goal: decide whether A terminates

Proof: Suppose that function *halts(X)* determines whether X terminates.

Program A
  if (*halts*(A))
    Let $a = \text{run}\ (A)$;
  else
    Let $a = 0$;
  Return $\mathbf{1}$ if $a = 0$ else $\mathbf{0}$;

Then, $A() \neq A()$.
Because A always returns a value, and $A()$ equals $1 - A()$.

Hard Problems

For every function $t(n)$ there exists a language L or decision problem f such that

- for every program A that recognizes L or computes f
- the running time of f is at least $t(n)$ for infinitely many $x \in \{0,1\}^*$.

Class NP

Optimization Problems

Consider optimization problem:

$$f(x) = \max\{value(s): s \text{ is a fesible solution for } x\}$$

Decision version of the problem:

$$g(x,k) = \begin{cases} 1, \text{if } f(x) \geq k \\ 0, \text{otherwise} \end{cases}$$

Note: if we can compute g we can also compute f.

How? What is the running time?

Optimization Problems

Consider optimization problem:

$$f(x) = \max\{value(s): s \text{ is a fesible solution for } x\}$$

Decision version of the problem:

$$g(x,k) = \begin{cases} 1, \text{if } f(x) \geq k \\ 0, \text{otherwise} \end{cases}$$

Solving problem f might be hard. However, for all problems we have seen so far checking whether s is a feasible solution is easy. For these problems, there exists a proof that $g(x,k) = 1$ if that's the case.

Classes P and NP

P is the class of all decision problems computable in polynomial time: $f \in \boldsymbol{P}$ if there exists a polynomial-time algorithm A that computes f.

NP is the class of all decision problems f for which $f(x) = 1$ can be proved in polynomial-time i.e., $f \in \mathbf{NP}$ if there exists a polynomial-time algorithm* A (verifier) such that

$$f(x) = 1$$

if and only if

$$\exists \text{witness/certificate } w \in \{0,1\}^* \text{ s.t. } A(x, w) = 1$$

*The running time should be polynomial in the size of x.

Proofs: Soundness and Completeness

Soundness: If there exists a witness w such that

$$A(x, w) = 1$$

then $f(x) = 1$ or $x \in L$.

Completeness: If $f(x) = 1$ or $x \in L$, then there exists a witness w such that

$$A(x, w) = 1$$

Completeness and Soundness

Soundness: If there exists a witness w such that

$$A(x, w) = 1$$

then $f(x) = 1$ or $x \in L$.

Completeness: If $f(x) = 1$ or $x \in L$, then there exists a witness w such that

$$A(x, w) = 1$$

Mathematics:

- Soundness: If we have a proof for a statement x, then x is always true.
- However, if x is true, then there may not exist a proof of x.

Completeness and Soundness

Soundness: If there exists a witness w such that

$$A(x, w) = 1$$

then $f(x) = 1$ or $x \in L$.

Completeness: If $f(x) = 1$ or $x \in L$, then there exists a witness w such that

$$A(x, w) = 1$$

Criminal Justice in the Ideal World:

- Soundness: If we have a proof that x is guilty, then x is guilty.
- Completeness: If x is guilty, then there exists a proof that x is guilty.

Examples of problems in NP

- Composite numbers: Is an integer number x composite?
  Witness: numbers a and b such that $x = ab$.
- Can we schedule a given set of jobs on m machines?
  Witness: a feasible schedule.

Both of these problems are in **P**.

P ⊂ NP

Thm: Any decision problem f in P also belongs to NP.

Proof: f is in P, thus there exists a polynomial-time algorithm A that decides f i.e., $f(x) = 1 \Leftrightarrow f(x) = 1$. The witness w can be the empty string.

Question: Is P equal NP?

P ⊂ NP

Thm: Any decision problem f in P also belongs to NP.

Proof: f is in P, thus there exists a polynomial-time algorithm A that decides f i.e., $f(x) = 1 \Leftrightarrow f(x) = 1$. The witness w can be the empty string.

Question: Is P equal NP?

Answer: We don't know, but it is generally believed that

$$P \neq NP$$

P vs NP

The precise statement of the **P** versus **NP** problem was introduced in 1971 by Stephen Cook in his seminal paper

"The complexity of theorem proving procedures"

and is considered to be the most important open problem in computer science.

It is one of the seven ***Millennium Prize Problems*** selected by the Clay Mathematics Institute to carry a ***$1,000,000*** prize for the first correct solution.

---

*from Wikipedia.

More examples

SAT: Determine whether a given Boolean formula $\phi(x_1, \dots, x_n)$ is satisfiable i.e., if there exists $x_1^*, \dots, x_n^* \in \{0,1\}$ such that $\phi(x_1^*, \dots, x_n^*) = 1$.

Hamiltonian Path: Is there a path p in graph G that visits every ***vertex*** exactly once.

Eulerian Path: Is there a path p in graph G that visits every ***edge*** exactly once.

More examples

SAT: Determine whether a given Boolean formula $\phi(x_1, \dots, x_n)$ is satisfiable i.e., if there exists $x_1^*, \dots, x_n^* \in \{0,1\}$ such that $\phi(x_1^*, \dots, x_n^*) = 1$. Is not* in P.

Hamiltonian Path: Is there a path p in graph G that visits every ***vertex*** exactly once. Is not* in P.

Eulerian Path: Is there a path p in graph G that visits every ***edge*** exactly once. Is in P.

---

*Unless $\mathrm{P} = \mathrm{NP}$.

More examples

MAX CUT: Given a graph $G = (V, E)$ find a cut (L, R) of maximum size. The size of the cut is the number of edges (u, v) with $u \in L$ and $v \in R$.

Decision version of MAX CUT: Given a graph G and number k determine whether the value of max cut in G is $\geq k$.

MAX CUT is not* in P.

---

*Unless $\mathrm{P} = \mathrm{NP}$.

EXP – Exponential Running Time

EXP is the class of decision problems f that can be decided in time $\exp(n^c)$: specifically, $f \in EXP$ if there exists a program A with running time at most $O(\exp(n^c))$ for some constant c.

Thm: $\mathrm{P} \subseteq \mathrm{EXP}$ and $\mathrm{P} \neq \mathrm{EXP}$

Proof idea:

- Proof of $\mathrm{P} \subseteq \mathrm{EXP}$ is trivial;
- Proof of $\mathrm{P} \neq \mathrm{EXP}$ follows Cantor's diagonal argument.

NP vs EXP

EXP is the class of decision problems f that can be decided in time $\exp(n^c)$: specifically, $f \in EXP$ if there exists a program A with running time at most $O(\exp(n^c))$ for some constant c.

Thm: NP ⊆ EXP. It is believed that NP ≠ EXP.

NP vs EXP

Thm: NP ⊆ EXP

Proof: Consider a decision problem $f \in NP$. For this problem, there exists a verifier i.e., a polynomial-time algorithm A s.t.

$$f(x) = 1 \Leftrightarrow \exists\, w \text{ s.t. } A(x, w) = 1$$

Suppose the length of w is upper-bounded by a polynomial $p(|x|)$.

Algorithm B

For each string $w \in \{0,1\}^{p(|x|)}$:
  if $A(x, w) = 1$, then return 1 //yes

return 0 //no

Thm: NP ⊆ EXP

Algorithm B

For each string $w \in \{0,1\}^{p(|x|)}$:
  if $A(x, w) = 1$, then return 1 //yes

return 0 //no

Claim 1: The running time of B is exponential in $|x|$.

Claim 2: $B(x) = 1 \Leftrightarrow f(x) = 1$

Proof ⇒ If $B(x) = 1$, then for some $w \in \{0,1\}^{p(|x|)}$, we have $A(x, w) = 1$

Thus (soundness), $f(x) = 1$.

Thm: NP ⊆ EXP

Algorithm B

For each string $w \in \{0,1\}^{p(|x|)}$:
  if $A(x,w) = 1$, then return 1 //yes

return 0 //no

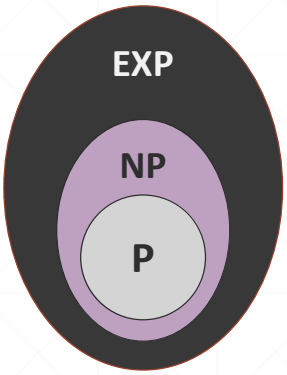
Claim 2: $B(x) = 1 \Leftrightarrow f(x) = 1$

Proof ⇒ If $f(x) = 1$, then for some $w \in \{0,1\}^{p(|x|)}$, we have $A(x,w) = 1$ (completeness). The algorithm B will find this w or another w' such that $A(x,w) = 1$ and return 1.

Classes of Decision Problems

- **NP –** class of problems that have verifiable solutions.
- **P** – class of problems that we can solve.
- Common belief:

$$\mathbf{P \neq NP}$$



Classes P and co-NP

NP is the class of all decision problems f for which $f(x) = 1$ can be proved in polynomial-time i.e., $f \in \mathbf{NP}$ if there exists a polynomial-time algorithm* A (verifier) such that

$$f(x) = 1 \Leftrightarrow \exists w \in \{0,1\}^{p(|x|)} \text{ s.t. } A(x,w) = 1$$

co-NP is the class of all decision problems f for which $f(x) = 0$ can be proved in polynomial-time i.e., $f \in \mathbf{coNP}$ if there exists a polynomial-time algorithm* B (verifier) such that

$$f(x) = 0 \Leftrightarrow \exists w \in \{0,1\}^{p(|x|)} \text{ s.t. } B(x,w) = 0$$

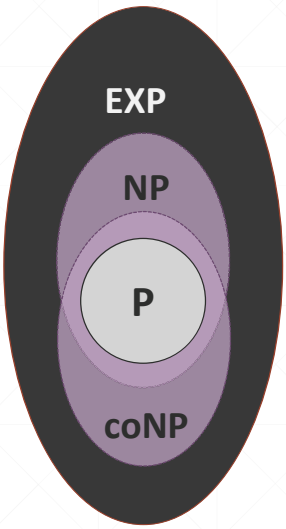
$f \in \text{NP}$ if and only if $(1 - f) \in \text{coNP}$

$L \in \text{NP}$ if and only if $\bar{L} \in \text{coNP}$

Classes of Decision Problems

- **NP –** class of problems that have verifiable solutions.
- **P** – class of problems that we can solve.
- Common belief:

$$\mathbf{P} \neq \mathbf{NP}$$



Reductions

Reductions

Can we compare two problems f and g in terms of hardness?

1st attempt: f is easier than g if the following statement holds:

if we can solve g, we can solve f

As is, this definition is not useful though.

Turing/Cook Reductions

Can we compare two problems f and g in terms of hardness?

Def: Problem f polynomial Turing/Cook - reduces to g if there exists a polynomial-time algorithm A that solves f given a "black box" or "oracle" access to g.

That is, suppose that A can call function $g(\cdot)$ and those calls are counted as one operation.

Thm: If $f \leq_T g$ and $g \in P$, then $f \in P$.

Question: If $f \leq_T g$ and $g \in NP$, is it true that $f \in NP$?

Turing/Cook Reductions

Can we compare two problems f and g in terms of hardness?

Def: Problem f polynomial Turing/Cook - reduces to g if there exists a polynomial-time algorithm A that solves f given a "black box" or "oracle" access to g.

That is, suppose that A can call function $g(\cdot)$ and those calls are counted as one operation.

Thm: If $f \leq_T g$ and $g \in P$, then $f \in P$.

Question: If $f \leq_T g$ and $g \in NP$, is it true that $f \in NP$? No*

---

*unless NP=co-NP

Karp or Many-one Reductions

Can we compare two problems f and g in terms of hardness?

Def: Problem f polynomial Karp-reduces to g if there exists an polynomial time algorithm (reduction) R such that

$$f(x) = 1 \Leftrightarrow g\big(R(x)\big) = 1$$

Lemma: If $f \leq_m g$, then $f \leq_T g$.

Corollary: If $f \leq_m g$ and $g \in P$, then $f \in P$.

Karp or Many-one Reductions

Can we compare two problems f and g in terms of hardness?

Def: Problem f polynomial Karp-reduces to g if there exists an polynomial time algorithm (reduction) R such that

$$f(x) = 1 \Leftrightarrow g\big(R(x)\big) = 1$$

Question: If $f \leq_m g$ and $g \in \mathrm{NP}$, is it true that $f \in \mathrm{NP}$?

Karp or Many-one Reductions

Can we compare two problems f and g in terms of hardness?

Def: Problem f polynomial Karp-reduces to g if there exists an polynomial time algorithm (reduction) R such that

$$f(x) = 1 \Leftrightarrow g\big(R(x)\big) = 1$$

Thm: If $f \leq_m g$ and $g \in \mathrm{NP}$, is it true that $f \in \mathrm{NP}$?
Proof: $f(x) = 1 \Leftrightarrow g\big(R(x)\big) = 1 \Leftrightarrow \exists w\, A(R(x), w) = 1.$

Thus, $A(R(\cdot),\cdot\,)$ is the verifier for the problem f.

NP-complete problems

Is there a hardest problem in NP?

A problem $g \in NP$ is NP-complete if for every $f \in NP$,

$$f \leq_m g$$

The Cook–Levin Thm: There exist NP-complete decision problems. **SAT** is one of them.

Problem SAT: Determine whether a given Boolean formula $\phi(x_1, \dots, x_n)$ is satisfiable i.e., if there exists $x_1^*, \dots, x_n^* \in \{0,1\}$ such that $\phi(x_1^*, \dots, x_n^*) = 1$.

NP-complete Problems

Observation: Most problems we need to solve in practice either belong to P or are NP-complete*.

Thm: If a problem f is NP-complete and $\mathrm{P} \neq \mathrm{NP}$, then $f \notin \mathrm{P}$.

Proof: Suppose that $f \in P$. Consider a problem $g \in NP$. Since f is NP-complete, we have $g \leq_m f$. Hence, $g \in P$. Consequently, $\mathrm{P} = \mathrm{NP}$.

---

*This is not a formal statement.

NP-complete Problems

3-dimensional matching	Feedback arc set	Max independent set
3-partition	Graph homomorphism	Minimum k-cut
3-coloring	Graph coloring	Modularity maximization
Capacitated MST	Hamiltonian path	Partition problem
Max Clique	Integer programming	Pathwidth
Dominating set	Knapsack problem	Set cover
Bandwidth problem	Longest path problem	Traveling salesman
Feedback vertex set	Maximum cut	Vertex cover

Circuit Satisfiability

Circuit Satisfiability

CIRCUIT-SAT: Given a combinational circuit built from AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?

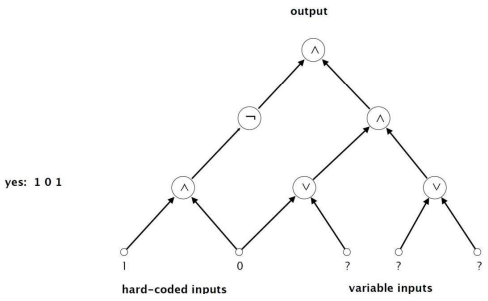


figure by Kevin Wayne

CIRCUIT-SAT is in NP

Proof:

An assignment satisfying the circuit is the witness, thus

CIRCUIT-SAT $\in$ NP

CIRCUIT-SAT is in NP-complete

- Any algorithm that takes a fixed number (n) of bits as input and produces a *yes* or *no* answer can be represented by a circuit.
- Consider any decision problem $f \in \text{NP}$. It has a polynomial time verifier $A(x, w)$. Let's represent A as a circuit C.
- Reduction $R(x)$: Return circuit C with hard-coded values of x and input variables $w_1, \ldots, w_k$.
- If $C(x,\cdot)$ is satisfiable if and only if there exists a witness w^* such that $C(x, w^*) = 1$, but $\text{A}(x, w^*) = C(x, w^*) = 1$, hence $f(x) = 1$ if and only if $C(x,\cdot)$ is satisfiable.

How to prove NP-completeness?

- To show that a problem f is NP-complete prove that

$$g \leq_m f$$

for a known NP-complete problem g.

Lemma: $\leq_m$ is a transitive relation i.e.,

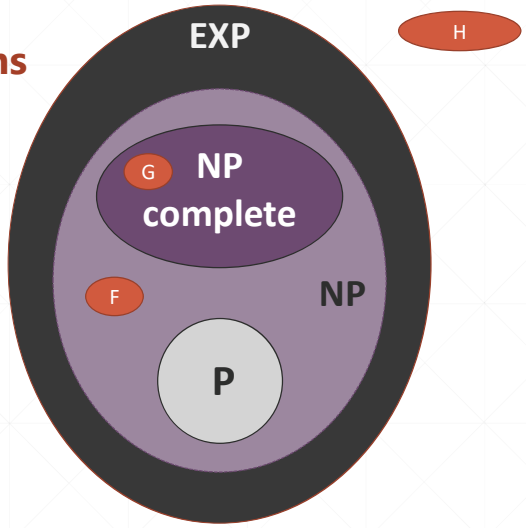
$$h \leq_m g \text{ and } g \leq_m f \text{ then } h \leq_m f$$

Corollary: $g \leq_m f$ and g is NP complete, the f is NP complete.

Classes of Decision Problems

- $f \leq_m g$ if there is a poly time reduction R_1 s.t.
  $$f(x) = g(R_1(x))$$
- $g \leq_m h$ if there is a poly time reduction R_2 s.t.
  $$g(x) = h(R_2(x))$$

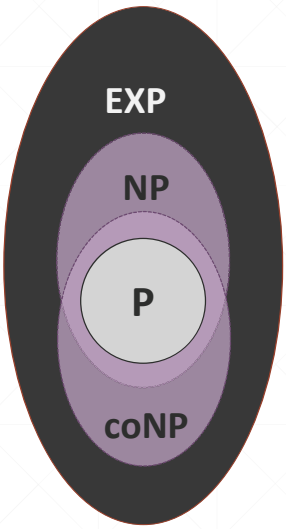
Then, $f(x) = h(R_2(R_1(x)))$.



Classes of Decision Problems

- **NP –** class of problems that have verifiable solutions.
- **P** – class of problems that we can solve.
- Common belief:

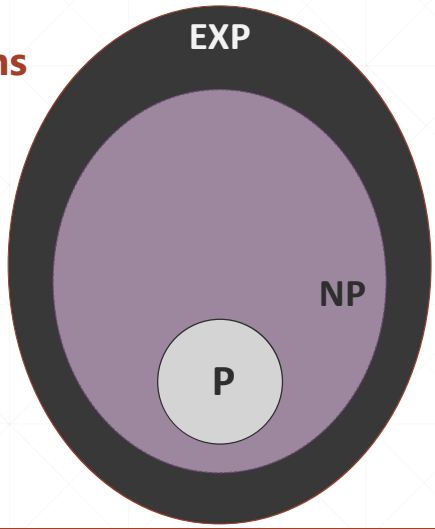
$$P \neq NP$$



Classes of Decision Problems

- **NP –** class of problems that have verifiable solutions.
- **P** – class of problems that we can solve.
- Common belief:

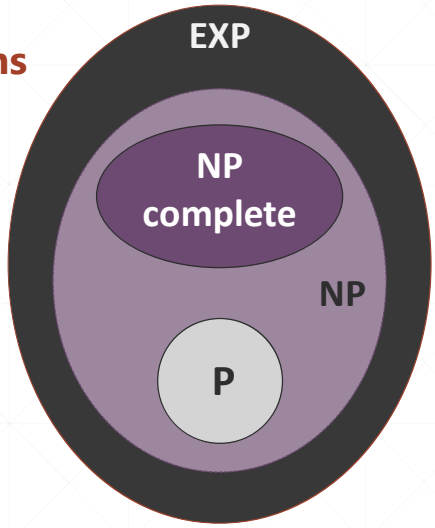
$$P \neq NP$$



Classes of Decision Problems

- **NP –** class of problems that have verifiable solutions.
- **P** – class of problems that we can solve.
- Common belief:

$$P \neq NP$$



Classes of Decision Problems

- $f \leq_m g$ if there is a poly time reduction R_1 s.t.
  $$f(x) = g(R_1(x))$$
- $g \leq_m h$ if there is a poly time reduction R_2 s.t.
  $$g(x) = h(R_2(x))$$

Then, $f(x) = h(R_2(R_1(x)))$.