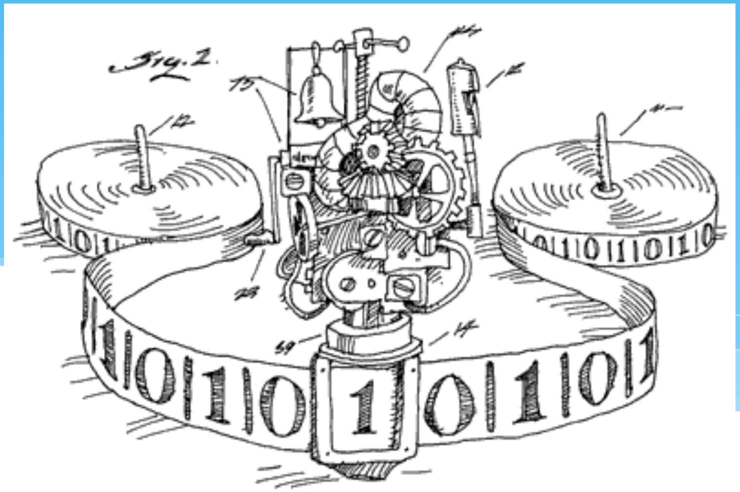
EECS 376: Foundations of Computer Science

Chris Peikert

6 March 2023



Complexity: Agenda

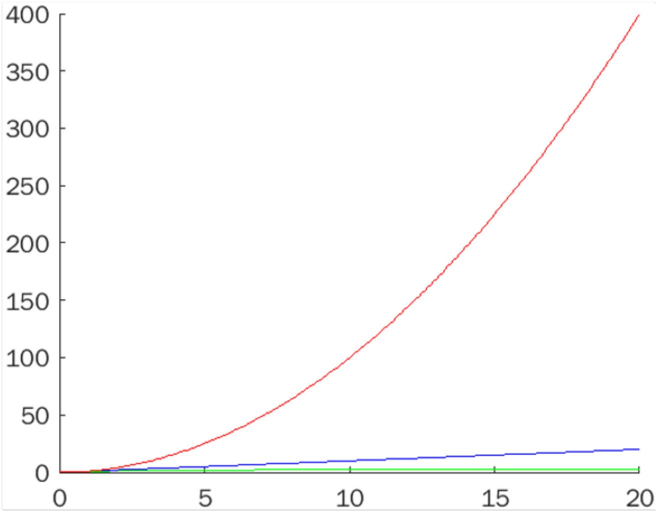
1) What “resources” (time, memory, randomness, …) are needed to solve a problem on a computer?
2) Which problems can be solved **efficiently** on a computer?
3) *What does “**efficient**” even mean?*
4) The hardest way to make $1 Million: P vs. NP.

Why Polynomial?

* “For practical purposes the difference between algebraic [polynomial] and exponential order is often more crucial than the difference between finite and non-finite.”
  – Jack Edmonds
  (defined complexity class P, 1965)

Review: Running Time

- We measure "efficiency" of an algorithm by how its (*worst-case*) ***runtime*** scales with the input "size"
- We express this growth asymptotically: e.g., $O(\log n), O(n), O(2^n), \ldots$ where n denotes the input size.
- Size = # bits to write/represent the input on a computer.



Efficient $\equiv$ "polynomial runtime in input size";
Very robust definition (as we'll see)

Polynomial-Time Algorithms

We have seen polynomial-time algorithms for various problems:

GCD(x,y)	Euclid's algorithm ($O(\log(x+y))$ divisions)
Longest Increasing Subseq.	Dynamic Prog. (time $O(n^2)$)
Longest Common Subseq.	Dynamic Prog. (time $O(n^2)$)
All-Pairs Shortest Paths	Dynamic Prog. (time $O(n^3)$)
Minimum Spanning Tree	Greedy (time $O(n \log n)$)

There are several others: Matching, Max-flow/min-cut, ….

Primality testing [Agarwal, Kayal, Saxena 2002]: Time $O(\log^6 N)$

Based on many clever and ingenious ideas.

Several algorithms still to be discovered!

A Poly-Time Algorithm for Every (Solvable) Problem?

- **Longest Path**: Given G and vertices (s,t), find a *longest* s->t path (w/o cycle).
- **Hamiltonian Cycle**: Given G, find a cycle that visits each vertex exactly once.
- **Independent Set**: Given G, find a largest subset of vertices that have no edges among them.
- **Subset Sum**: Given $a_1, a_2, \ldots, a_n \in \mathbb{Z}$ and target t, find a subset that sums to t
  - (what about the DP solution we saw in the class?)
- …

Most believe that none of these problems has a poly-time algorithm.
We know that if any one of them does, then all of them do!

Theory of "**P vs. NP**": a crown jewel of Computer Science—
with a million dollar reward!

The Class P

* **Definition:** **P** = the set of all languages that can be decided by a TM in (some) polynomial time in the input size.
* In other words: $L \in \mathbf{P}$ if L is ***efficiently decidable***, i.e., it has a polynomial-time decision algorithm.
* **Formally:** $\mathbf{P} = \bigcup_{k \geq 1} DTIME(n^k)$
* **Properties:**
  * Model agnostic: can replace TM with any "realistic" (deterministic) model.
  * Composition: if an efficient M has an oracle, which is instantiated by an efficient M', then the resulting program is also efficient.
  * *Proof idea*: $(n^k)^{k'} = n^{k \cdot k'}$ is also polynomial (for constants k, k').

Search vs. Decision Problems

Definition: P = the set of all languages that can be decided by a TM in (some) polynomial time in the input size.

We are (still) talking about decision problems, with “yes/no” answers.

What about gcd(x,y)? Shortest path s->t? LIS(x)? Etc…
These don’t have yes/no answers…

Not a problem! Can *recast* them as decision problems, and solve the original search problems with only polynomial “slowdown.”

Recasting Search as Decision: GCD

- **Problem:** Given integers x, y, z, is $\gcd(x, y) \geq z$?
  - $L_{GCD} = \left\{ \left(x, y, z\right) : x, y, z \in \mathbb{N} \text{ and } z \geq \gcd(x, y) \right\}$
- **Claim:** L_{GCD} is efficiently decidable, i.e., $L_{GCD} \in \mathbf{P}$.
- **Proof:** Run Euclid's algorithm and answer accordingly.
- **Claim:** Using any L_{GCD}-decider, we can efficiently compute $\gcd(x, y)$ itself.
- **Proof:** binary search on $z \in \{1 \ldots \min(x, y)\}$, asking decider if $\gcd(x, y) \geq z$.
  Runtime: $\log \min(x, y)$ = O(input size) calls to decider.

Recasting: Shortest Paths

- **Problem:** Given an (integer-)weighted graph G, vertices s and t, and $z \in \mathbb{Z}$, is there an $s \to t$ path of length at most z ?
  - $L_{path} = \{\langle G, s, t, z\rangle : G$ has an $s \to t$ path of length $\leq z\}$
- **Claim:** L_{path} is efficiently decidable, i.e., $L_{path} \in \mathbf{P}$.
- **Proof:** run Floyd-Warshall APSP and answer accordingly.
- **Claim:** using <u>any</u> L_{path}-decider, we can compute the length of a shortest $s \to t$ path (with no negative-weight cycles).
- **Proof:** binary search on $z \in \{-Z, \ldots, Z\}$ where $Z = \sum_{(u,v)\in E} |w(u,v)|$.

Recasting Search to Decision In General

For a <u>search problem</u> with a **numerical output** A, recast it as a <u>language</u> with an **extra numerical input** z that asks: is $z \geq A$?

Using any decider, can solve the search problem via binary search. Only O(log(RANGE)) calls to the decider.

For a search problem with an **array output** A, recast it as a language with **extra inputs** z, i that asks: is $z \geq A[i]$?

Using any decider, can solve the search problem via binary search for each index i.

Similarly for **string outputs**, **set outputs**, …

Introducing: The Class NP

Common misconception: NP does **not** stand for "Non/Not Polynomial" !

NP means "**Nondeterministic** Polynomial time."

We won't cover nondeterminism in this course.
Instead, we'll use a conceptually simpler definition of NP.

"A better name would have been **VP**: **verifiable** in polynomial time."
-Clyde Kruskal

Quote of The Day

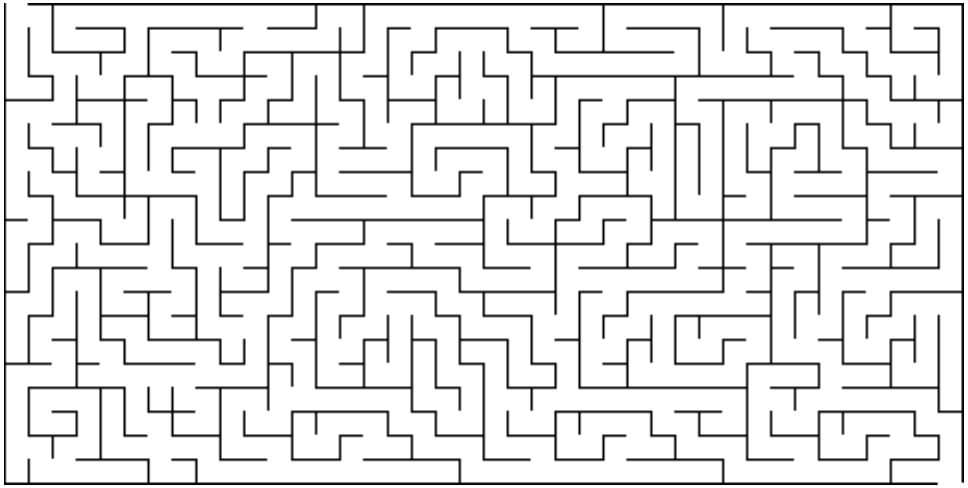
"Doveryai, no proveryai"

(Trust, but verify)

– Old Russian Proverb,
Used by Ronald Reagan

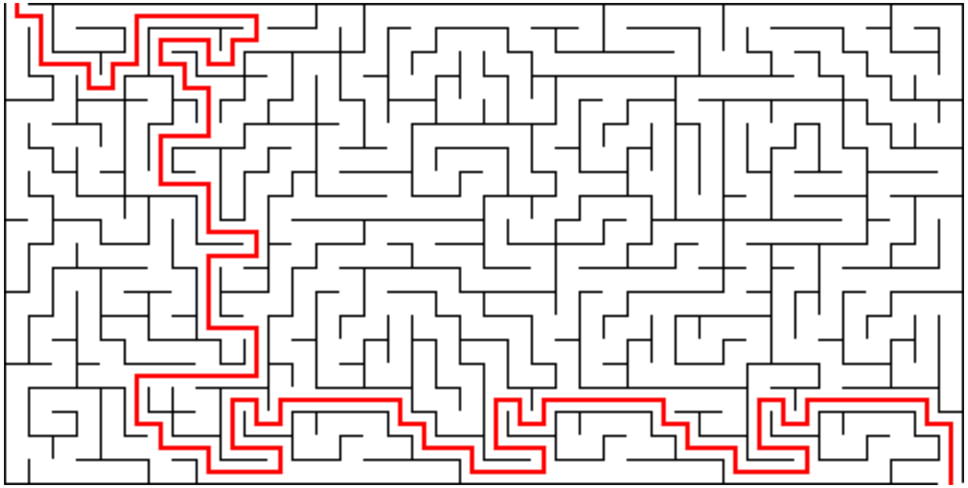
Verifiable Problems

* **Example 1:** Given a maze, is there a route through the maze from the start to finish?
* **Answer:** Yes.
* **Response:** We're not convinced—you could be lying!



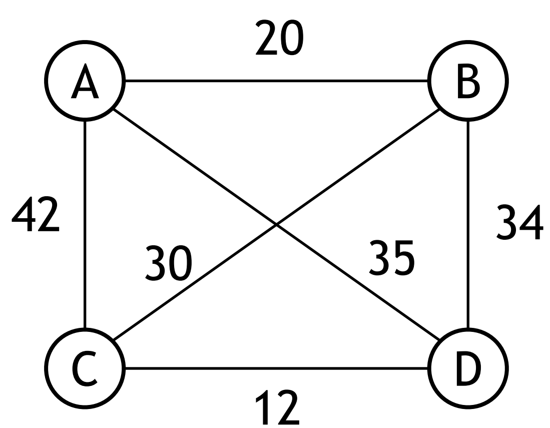
Verifiable Problems

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



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:** 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.



Verifiable Problems

- **Example 3: Subset Sum**
- Given integers $a_1, a_2, \ldots, a_n$ and target t, is there a subset of the a_i that sums to t ?

- **Answer:** Yes; here are indices i of some a_i that do.
- **Response:** We check that all given i are distinct, that the specified a_i sum to t, and are convinced.

Efficiently Verifiable Problems

- **Intuition (informal):** A decision problem is ***efficiently verifiable*** if:

1. When the answer is "yes": we can be efficiently <u>convinced</u> given some <u>*additional information*</u>: "certificate", "proof", "witness".

2. When the answer is "no": <u>no</u> additional information—even maliciously generated—could fool us into thinking that the answer is yes.

- **Asymmetric**: we never wish to be convinced that the answer is "no".

- TSP: If there is a cheap enough tour, we can be convinced (give one). If not, even a "malicious adversary" cannot fool us.
- Subset Sum: if some subset sums to t, we can be convinced (give one). If not, even a "malicious adversary" cannot fool us.

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.
   - (When the answer is "yes", we can be efficiently convinced given some certificate/witness/proof.)
3. For every $x \notin L$, $V(x, c)$ rejects for *all* c.
   - (When the answer is "no", we won't be fooled by any [bogus] proof.)

Definition: the class **NP** = set of all efficiently verifiable languages.

I.e., $L \in \mathbf{NP}$ if L is efficiently verifiable.

Example: TSP

* **TSP:** Given n cities and pair-wise distances, is there a cycle that visits every city *exactly once*, and has length at most k?
  * $\mathrm{TSP} = \{(G, k) : G$ is a weighted graph w/ a ***tour*** of length $\leq k\}$

* **Claim:** TSP is efficiently verifiable, i.e., $\mathrm{TSP} \in \mathbf{NP}$.

Consider certificate c in the form of a sequence of vertices (which is supposed to be a tour of length $\leq k$, if one exists)

bool **verifyTSP**(graph $G = (V, E)$, int k,
certificate $c = (v_1, \ldots, v_m)$):
1. if c is not a permutation of V: reject
2. **return** length$(v_1, \ldots, v_m, v_1) \leq k$

Correctness Analysis of verifyTSP

bool **verifyTSP**(graph $G = (V, E)$, int k,
certificate $c = (v_1, \ldots, v_m)$):
1. if c is not a permutation of V: reject
2. **return** length$(v_1, \ldots, v_m, v_1) \ \leq k$

* **Case 1:** If $(G, k) \in \mathrm{TSP}$, then *some* certificate c makes **verifyTSP**$((G, k), c)$ return *true*.
  * Let $c = (v_1, \ldots, v_m)$ be the vertices of G, ordered according to some tour of length $\leq k$ (which exists by hypothesis).
    * Since c is a permutation of V, line 1 does not reject.
  * Then, length$(v_1, \ldots, v_m, v_1) \leq k$ by assumption and **verifyTSP**(x, c) returns *true* on line 2, as desired.

Correctness Analysis of verifyTSP

bool **verifyTSP**(graph $G = (V, E)$, int k,
certificate $c = (v_1, \ldots, v_m)$):
1. if c is not a permutation of V: reject
2. **return** length$(v_1, \ldots, v_m, v_1) \leq k$

- **Case 2:** If $(G, k) \notin \mathrm{TSP}$, then **verifyTSP**$(x, c)$ rejects for *any* certificate c.
  - If c is not a permutation of V, then line 1 rejects, as desired.
  - Now suppose $c = (v_1, v_2, \ldots, v_m)$ is a permutation of V.
  - By assumption, G has no tour of length $\leq k$.
    - Therefore, length$(v_1, \ldots, v_m, v_1) > k$.
    - Thus, line 2 rejects, as desired.

Runtime Analysis of verifyTSP

bool **verifyTSP**(graph $G = (V, E)$, int k,
certificate $c = (v_1, \ldots, v_m)$):
1. if c is not a permutation of V: reject
2. **return** length$(v_1, \ldots, v_m, v_1) \leq k$

* Suppose the input (G, k) has (bit) length n.
* There are at most n vertices in G.
* Line 1 takes $\mathrm{O}(n)$ time: use a boolean array to check if each vertex appears exactly once in c.
* Line 2 takes $\mathrm{O}(n)$ time: sum the weights of the edges on the tour.

* Therefore, the runtime is polynomial in n, as desired.

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! (next slide)

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

Aside: Certificate for Primes

L_{Primes} = {n : n is prime}

Theorem (Pratt 75): L_{Primes} is efficiently verifiable.

Proof: Key property of primes: n is prime if there is an x such that $x^{n-1} = 1 \pmod{n}$ but $x^e \neq 1 \pmod{n}$ for all $e = (n-1)/p_i$,

where p_i are the prime divisors of $n - 1$.

Certificate: $(x, p_1, \ldots, p_k,$ certificate that each p_i is prime$)$

By induction: Certificate has size $O(\log n)$.

EECS M ECE CSE

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
* $L \in$ **NP** if there is a polynomial-time (in $|x|$) algorithm V where:
  * $x \in L \implies V(x, c)$ accepts for some c
  * $x \notin L \implies V(x, c)$ rejects for every c
* **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 *verifiable* problem
  also efficiently *solvable*?
  Seems not… but we don't know for sure!

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

The Major Open Problem of Computer Science