Harvard CS 121 and CSCI E-121

Lecture 21: Nondeterministic Polynomial Time

Harry Lewis

November 14, 2013

- Reading: Sipser §7.3.

Problems about arithmetic: are they in P?

- Given two numbers N, M, compute their product.
  - What is the "size" of the numbers?
- Given a number N, decide if N is prime.
- Given a number N, compute N's prime factorization.

A bogus polynomial-time algorithm

Consider the following algorithm on input an n-bit number z:

- Repeat n times: let $z \leftarrow z \times z$ (using grade-school multiplication algorithm)

"Proof" that this algorithm is polynomial time:

- The loop has n iterations.
- Each time we multiply, which takes time $O(n^2)$.
- Total time $= n \cdot O(n^2) = O(n^3)$.

Where is the error?

Another way of looking at P

- Multiplicative increases in time or computing power yield multiplicative increases in the size of problems that can be solved

- If L is in P, then there is a constant factor $k > 1$ such that
  - If you can solve problems of size s within a given amount of time
  - and you are given a computer that runs twice as fast, then
  - you can solve problems of size $k \cdot s$ on the new machine in the same amount of time.

- E.g. if L is decidable in $O(n^d)$ time, then with twice as much time you can solve problems $2^{1/d}$ as large

Exponential time

- E $= \cup_{c>0}$TIME(c^n)
- For problems in E, a multiplicative increase in computing power yields only an *additive* increase in the size of problems that can be solved.
- If L is in E, then there is a constant k such that
  - If you can solve problems of size s within a given amount of time
  - and you are given a computer that runs twice as fast, then
  - you can solve problems only of size $k + s$ on the new machine using the same amount of time.

"Nondeterministic Time"

We say that a nondeterministic TM M decides a language L iff for every $w \in \Sigma^*$,

1. Every computation by M on input w halts (in state q_{accept} or state q_{reject});
2. $w \in L$ iff there exists at least one accepting computation by M on w.
3. $w \notin L$ iff every computation by M on w rejects (or dies, with no applicable transitions).

M decides L in nondeterministic time $t(\cdot)$ iff for every w, every computation by M on w takes at most $t(|w|)$ steps.

More on Nondeterministic Time

1. Linear speedup holds.

2. "Polynomial equivalence" holds among nondeterministic models

   e.g. L decided in time T by a nondeterministic multitape TM

   $\Rightarrow L$ decided in time $O(T^2)$ by a nondeterministic 1-tape TM

Definition:

NTIME$(t(n)) =$
$\{L : L$ is decided in time $t(n)$ by some nondet. multitape TM$\}$

NP = $\bigcup_{\text{polynomial } p}$ NTIME$(p) = \bigcup_{k \geq 0}$ NTIME(n^k).

P vs. NP

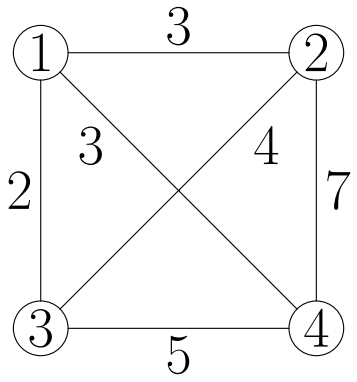
- Clearly P $\subseteq$ NP. But there are problems in NP that are not obviously in P ($\neq$ "obviously not")

- TSP = TRAVELLING SALESMAN PROBLEM.
  - Let $m > 0$ be the number of cities,
  - $D : \{1, \ldots, m\}^2 \to \mathcal{N}$ give the distance $D(i, j)$ between city i and city j, and
  - B be a distance bound

  Then TSP $=$

  $\{\langle m, D, B\rangle : \exists$ tour of all cities of length $\leq B\}$.

Traveling Salesman Problem: Example



$n = 4$

$B \geq 15 \Rightarrow \langle m, D, B \rangle \in \text{TSP}$
$B \leq 14 \Rightarrow \langle m, D, B \rangle \notin \text{TSP}$

"tour" = visits every city and returns to starting point

There are many variants of TSP, eg require visiting every city exactly once, triangle inequality on distances...

TSP$\in$ NP

- Why is TSP $\in$ NP?

  Because if $\langle m, D, B\rangle \in$ TSP, the following nondeterministic strategy will accept in time $O(n^3)$, where $n =$ length of representation of $\langle m, D, B\rangle$.

  – nondeterministically write down a sequence of cities $c_1, \ldots, c_t$, for $t \leq m^2$. ("guess")

  – trace through that tour and verify that all cities are visited and the length is $\leq B$. If so, halt in q_{accept}. If not, halt in q_{reject}. (and "check")

  If $\langle m, D, B\rangle \notin$ TSP, above has no accepting computations.

  But any obvious deterministic version of this algorithm takes exponential time.

A useful characterization of NP

- A verifier for a language L is an algorithm V such that

$$L = \{x : V \text{ accepts } \langle x, y \rangle \text{ for some string } y\}.$$

- A polynomial-time verifier is one that runs in time polynomial in $|x|$ on input $\langle x, y \rangle$.
- A string y that makes $V(\langle x, y \rangle)$ accept is a "proof" or "certificate" that $x \in L$.
- **Example:** TSP

  certificate y = ?

  $V(\langle x, y \rangle)$ = ?

- Without loss of generality, $|y|$ is at most polynomial in $|x|$.

NP is the class of easily verified languages

- **Theorem:** NP equals the class of languages with polynomial-time verifiers.

  Proof:

  $\Rightarrow$

  $\Leftarrow$

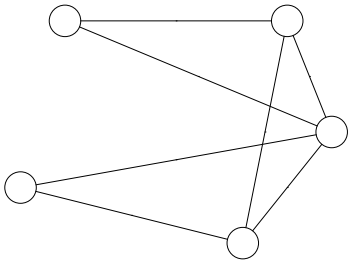
- "L is in NP iff members of L have short, efficiently verifiable certificates"

More problems in NP

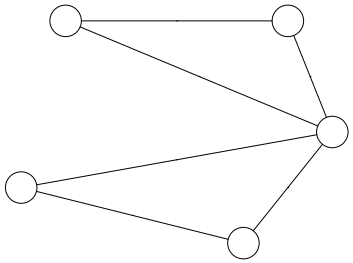
- HAMILTONIAN CIRCUIT

  HC $= \{G : G$ is an undirected graph with a circuit that touches each node exactly once$\}$.

HC



No HC



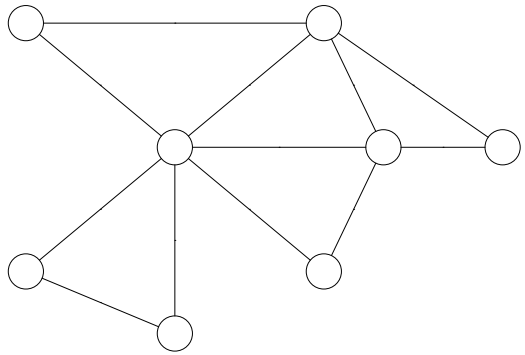
Really just a special case of TSP. (why?)

- We are not fussy about the precise method of representing a graph as a string, because all reasonable methods are within a polynomial of each other in length.

A "similar" problem that is in P

- EULERIAN CIRCUIT

  $$\text{EC} = \{G : G \text{ is an undirected graph with a circuit that passes through each edge exactly once}\}.$$

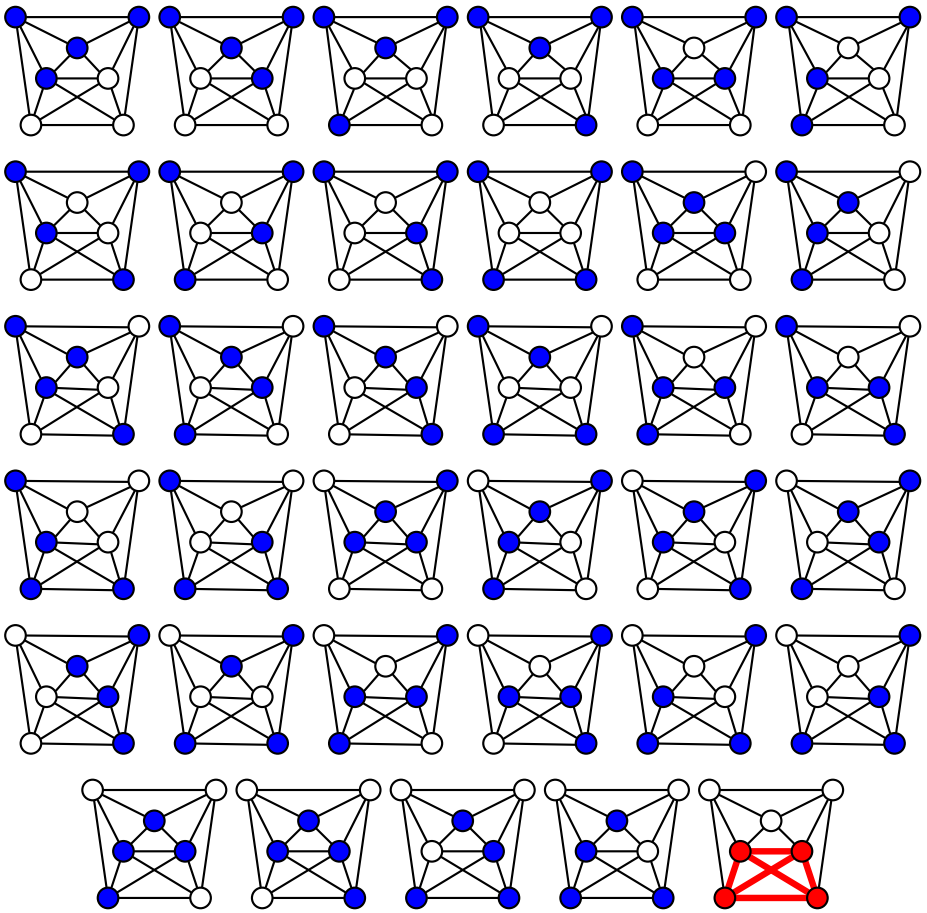


It is easy to check if G is Eulerian...

So EC $\in$ P.

CLIQUE $\in$ NP

- Given graph G and number n, are there n mutually connected nodes in G?



Composite Numbers

- COMPOSITES $= \{w : w$ a composite number in binary $\}$.

  COMPOSITES $\in$ NP

  Not obviously in P, since an exhaustive search for factors can take time proportional to the value of w, which grows as $2^n =$ exponential in the size of w.

  Only recently (2002), it was shown that COMPOSITES $\in$ P (equivalently, PRIMES $\in$ P).

Boolean logic

Boolean formulas

Def: A Boolean formula (B.F.) is either:

- a "Boolean variable" $x, y, z, \ldots$
- $(\alpha \vee \beta)$ where α, β are B.F.'s.
- $(\alpha \wedge \beta)$ where α, β are B.F.'s.
- $\neg\alpha$ where α is a B.F.

e.g. $(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$

[Omitting redundant parentheses]

Boolean satisfiability

Def: A truth assignment is a mapping
a : Boolean variables $\rightarrow \{0, 1\}$. [$0 =$ false, $1 =$ true]

The $\{0, 1\}$ value of a B.F. γ on a truth assignment a is given by the usual rules of logic:

- If γ is a variable x, then $\gamma(a) = a(x)$.
- If $\gamma = (\alpha \vee \beta)$, then $\gamma(a) = 1$ iff $\alpha(a) = 1$ or $\beta(a) = 1$.
- If $\gamma = (\alpha \wedge \beta)$, then $\gamma(a) = 1$ iff $\alpha(a) = 1$ and $\beta(a) = 1$.
- If $\gamma = \neg\alpha$, then $\gamma(a) = 1$ iff $\alpha(a) = 0$.

a satisfies γ (sometimes written $a \models \gamma$) iff $\gamma(a) = 1$.

In this case, γ is satisfiable. If no a satisfies γ, then γ is unsatisfiable.

Boolean Satisfiability

SAT $= \{\alpha : \alpha$ is a satisfiable Boolean formula$\}$.

Prop: SAT $\in$ NP

A "similar" problem in P: 2-SAT

A 2-CNF formula is one that looks like

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

i.e., a conjunction of clauses, each of which is the disjunction of 2 literals (or 1 literal, since $(x) \equiv (x \vee x)$)

2-SAT $=$ the set of satisfiable 2-CNF formulas.

e.g. $(x \vee y) \wedge (\neg x \vee \neg y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \notin$ SAT

2-SAT $\in$ P

Method (resolution):

1. If x and $\neg x$ are both clauses, then not satisfiable

   e.g. $(x) \wedge (z \vee y) \wedge (\neg x)$

2. If $(x \vee y) \wedge (\neg y \vee z)$ are both clauses, add clause $(x \vee z)$ (which is implied).

3. Repeat. If no contradiction emerges $\Rightarrow$ satisfiable.

$O(n^2)$ repetitions of step 2 since only 2 literals/clause.

Proof of correctness: omitted

P vs. NP

- We would like to solve problems in NP efficiently.
- We know P $\subseteq$ NP.
- Problems in P can be solved “fairly” quickly.
- What is the relationship between P and NP?

NP and Exponential Time

Claim: $\text{NP} \subseteq \bigcup_k \text{TIME}(2^{n^k})$

Of course, this gets us nowhere near P.

Is $\text{P} = \text{NP}$?

i.e., do all the NP problems have polynomial time algorithms?

It doesn't "feel" that way but as of today there is no NP problem that has been proven to require exponential time!

The Strange, Strange World if P = NP

- Thousands of important languages can be decided in polynomial time, e.g.
  - SATISFIABILITY
  - TRAVELLING SALESMAN
  - HAMILTONIAN CIRCUIT
  - MAP COLORING
  - ⋮

If P = NP, then searching becomes easy

- Every "reasonable" search problem could be solved in polynomial time.
  - "reasonable" $\equiv$ solutions can be recognized in polynomial time (and are of polynomial length)
  - SAT SEARCH: Given a satisfiable boolean formula, find a satisfying assignment.
  - FACTORING: Given a natural number (in binary), find its prime factorization.
  - NASH EQUILIBRIUM: Given a two-player "game", find a Nash equilibrium.
  - ⋮

If P $=$ NP, Optimization becomes easy

- Every "reasonable" optimization problem can be solved in polynomial time.
  - Optimization problem $\equiv$ "maximize (or minimize) $f(x)$ subject to certain constraints on x"
  - "Reasonable" $\equiv$ "f and constraints are poly-time"
  - MIN-TSP: Given a TSP instance, find the shortest tour.
  - SCHEDULING: Given a list of assembly-line tasks and dependencies, find the maximum-throughput scheduling.
  - PROTEIN FOLDING: Given a protein, find the minimum-energy folding.
  - CIRCUIT MINIMIZATION: Given a digital circuit, find the smallest equivalent circuit.

If P $=$ NP, Secure Cryptography becomes impossible

- **Cryptography:** Every encryption algorithm can be "broken" in polynomial time.
  - "Given an encryption z, find the corresponding decryption key K and message m" is an NP search problem.
  - Take CS127.

If P = NP, Artificial Intelligence becomes easy

- **Artificial Intelligence:** "Learning" is easy.
  - Given many examples of some concept (e.g. pairs (image1, "dog"), (image2, "person"), ...), classify new examples correctly.
  - Turns out to be equivalent to finding a short "classification rule" consistent with examples.
  - Take CS228.

If P = NP, Even Mathematics Becomes Easy!

- **Mathematical Proofs:** Can always be found in polynomial time (in their length).
  - SHORT PROOF: Given a mathematical statement S and a number n (in unary), decide if S has a proof of length at most n (and, if so, find one).
  - An NP problem!
  - cf. letter from Gödel to von Neumann, 1956.

Library of Congress

Gödel's Letter to Von Neumann, 50 years ago

[$\phi(n) =$ time required for a TM to determine whether a formula has a proof of length n]

...

If there really were a machine with $\phi(n) \sim k \cdot n$ (or even $\sim k \cdot n^2$) this would have consequences of the greatest importance. Namely, it would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine. ...

It would be interesting to know, for instance, the situation concerning the determination of primality of a number and how strongly in general the number of steps in finite combinatorial problems can be reduced with respect to simple exhaustive search. ...

The World if P $\neq$ NP?

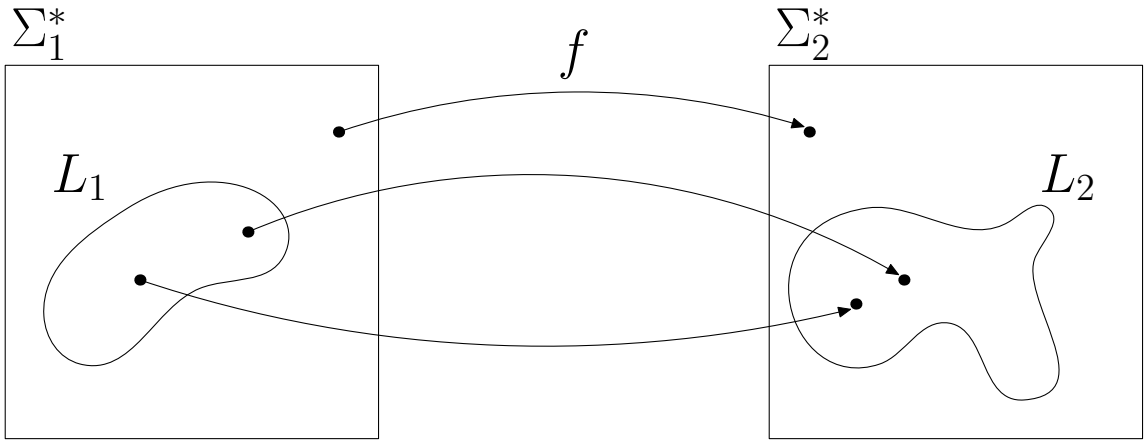
- **Q:** If P $\neq$ NP, can we conclude anything about any specific problems?

- **Idea:** Try to find a "hardest" NP language.
  - Just like A_{TM} was the "hardest" Turing-recoginizable language.
  - Want $L \in$ NP such that $L \in$ P iff every NP language is in P.

Polynomial-time Reducibility

- **Def**: $L_1 \leq_{\mathsf{P}} L_2$ iff there is a polynomial-time computable function $f : \Sigma_1^* \to \Sigma_2^*$ s.t. for every $x \in \Sigma_1^*$, $x \in L$ iff $f(x) \in L_2$.
- **Proposition:** If $L_1 \leq_{\mathsf{P}} L_2$ and $L_2 \in \mathsf{P}$, then $L_1 \in \mathsf{P}$.
- **Proof:**

$L_1 \leq_{\mathsf{P}} L_2$



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

$x \notin L_1 \Rightarrow f(x) \notin L_2$

f computable in polynomial time

$L_2 \in \mathsf{P} \Rightarrow L_1 \in \mathsf{P}$.

NP-Completeness

- **Def**: L is NP-complete iff
  1. $L \in$ NP and
  2. Every language in NP is reducible to L in polynomial time. ("L is NP-hard")

- **Prop:** Let L be any NP-complete language. Then P $=$ NP *if and only if* $L \in$ P.

Cook–Levin Theorem (Stephen Cook 1971, Leonid Levin 1973)

- **Theorem:** SAT (Boolean satisfiability) is NP-complete.
- **Proof:** Need to show that every language in NP reduces to SAT (!) Proof later.