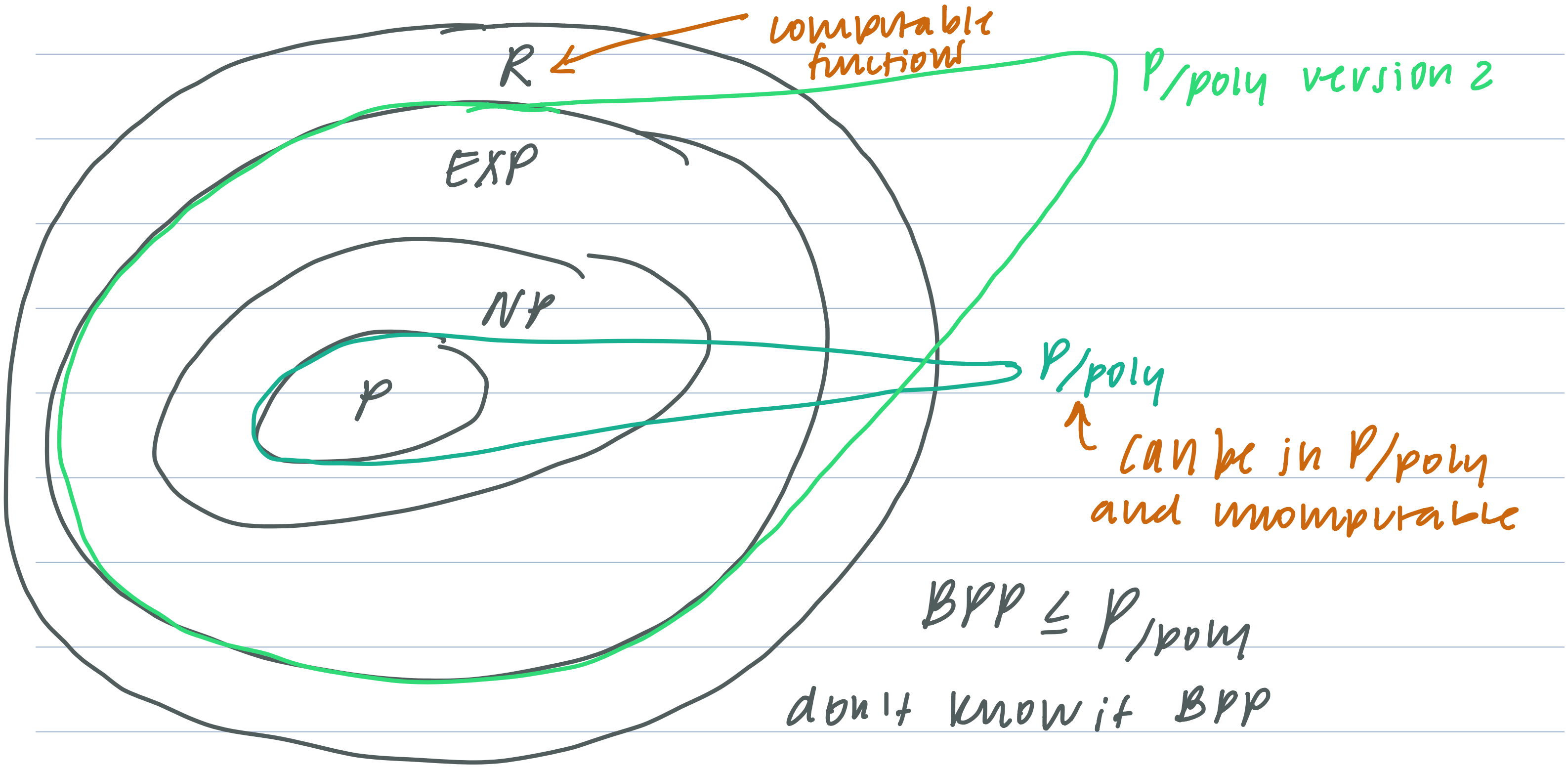
CS121 Section 8

November 6

P/poly Information

$F \in P/poly$ if $\forall n$, $F_n \in SIZE(n^c)$

$F_n : \{0,1\}^n \to \{0,1\}^*$



$BPP \leq P/poly$

don't know if BPP

What if P=NP

Search-to-Decision Reduction

F_ISET:

$ISET(G, k) \to$ largest independent set of size $k = 37$

$ISET(G/v_1, k) \to$ still 37? v_1 not part of independent set

rm vertex

⋮

max # runs = $|V|$

not 37? v_1 is elem of independent set

delete neighbors

binary search, alter vertices, etc.

Polynomial-time Reductions

- If it has a 3: 3SAT
- Graph: MAXCUT, ISET
- Others: VERTEXCOVER

Reduction Format

1. $F \in NP$
    - Show there is polynomial-time verification algo for F

2. F is NP-hard

    Reduce H to F → $H(x) = F(R(x))$

    1. Transformation R
        - describe transformation
        - show it is in poly-time
    2. Correctness
        - Soundness $H(X) = 1 \rightarrow F(R(X)) = 1$
        - Completeness $F(R(X)) = 1 \rightarrow H(X) = 1$

Stat Recap

Event space: Ω

random variable X: function mapping from $\Omega \to \mathbb{R}$

expectation: $\mathbb{E}$

variance: $V(X) = \mathbb{E}(X - \mathbb{E}(X))^2 = \mathbb{E}(X^2) - \mathbb{E}(X)^2$

$\mathbb{E}[Y|X]$ is a function of X

Bounds:

- Markov's Inequality: $P(X > \alpha) \leq \frac{E(X)}{\alpha}$
- Chernoff Bounds:

$$I_{X \geq 1} \leq X$$

$$\underset{\substack{\parallel \\ P(X \geq 1)}}{\mathbb{E}[I_{X \geq 1}]} \leq \mathbb{E}[X]$$

$$P(|X - E[X]| > \alpha) = P((X - E[X])^2 > \alpha^2) \leq \frac{\mathbb{E}(X - E(X))^2}{\alpha^2} = \frac{V(X)}{\alpha^2}$$

$$P(e^{tX} > e^{t\alpha}) \leq \frac{\mathbb{E}(e^{tX})}{e^{t\alpha}} = \frac{M_X(t)}{e^{t\alpha}}$$

Midterm

- 1 long: NP complete problem
- 1 wild card problem: maybe P=NP
- bonus: complexity class relationships?
- T/F: regular expressions

CS 121: Introduction to Theoretical Computer Science Fall 2019

Section 8

Prof. Boaz Barak

1 A brief review of the classes P and EXP

Remember that the class of functions P is defined as

$$\mathsf{P} \equiv \bigcup_{c \in \mathbb{N}} \mathsf{TIME}\left(n^c\right).$$

In other words, a function $F : \{0,1\}^* \to \{0,1\}$ is in P if it can be computed in polynomial time by a reasonable computational model (RAM machine, Turing machine, $NAND-TM$). The class EXP is defined as

$$\mathsf{EXP} \equiv \bigcup_{c \in \mathbb{N}} \mathsf{TIME}\left(2^{n^c}\right),$$

meaning that a function is in EXP if it can be computed in exponential time in the size of its input.

N.B. Note that our definitions of both P and EXP treats n as the size of the input. For complexity analysis, you must be clear about what n is – for example, stating that a graph algorithm is $O(n)$ with respect to the number of vertices is not the same as saying that it is $O(n)$ with respect to the number of bits that the algorithm takes as input.

2 The class NP

2.1 Getting familiar with NP

The class P tells us "which functions are efficiently solvable." Similarly, we may want to give a definition of functions that we can "check efficiently." This is what NP gives us.

Recall the definition of NP: a function $F : \{0,1\}^* \to \{0,1\}$ is in NP if $\exists a, b \in \mathbb{N}$ and $V : \{0,1\}^* \to \{0,1\}$ such that $V \in \mathsf{P}$ and

$$\forall x \in \{0,1\}^n, F(x) = 1 \iff \exists_{w \in \{0,1\}^{an^b}} \text{ such that } V(x,w) = 1.$$

What is the intuition behind this construction? Since the definition of F is from $\{0,1\}^*$ to $\{0,1\}$, we can think of $F(x)$ as evaluating binary questions. In this case w can be thought of as an example that the answer to the question we are asking is 'yes', and V is the "verifier" of these examples (since it runs in polynomial time, we say it's efficient). In the function $3SAT$, which maps 3CNF (conjunctive normal form) formulas to a single bit indicating whether or not they are satisfiable, is in NP because if a given boolean formula φ is in $3SAT$, we can provide a string w that is a satisfying assignment of the variables in φ, and V can efficiently check that φ is indeed satisfied. Similarly, the problem of telling whether there is a clique of size k in a given undirected graph is in NP because for any size k and graph G, we can give a w which is a clique of size k in graph G; the verifier V can then check the existence of all the requisite $\binom{k}{2}$ edges.

How does the $\Leftrightarrow$ fit into this? The $\Rightarrow$ says that if $F(x) = 1$, so the answer to the original question is "yes", then there must be a satisfying example w, and our verifier should successfully verify this example, yielding $V(x,w) = 1$. In the other direction, this is saying that if there is a w such that $V(x,w) = 1$ (so in english, there is a satisfying example for x), then it must be true that the answer to the original question $F(x) = 1$. This just means our verifier shouldn't find satisfying examples for problems that are actually not solvable!

There is one last detail to be aware of in this definition. We see that $w \in \{0,1\}^{an^b}$. This seems like a random detail, but definitions should not seem random. Whenever you see something like this that seems odd in a definition, you should ask yourself why this is. $w \in \{0,1\}^{an^b}$ means that our satisfying example should not be "too much longer than x." This is important because we are trying to have our verifier V be efficient, but all we enforced upon it is that $V \in P$. If $|w| = 2^n$ and our verifier takes linear time, then our verifier will actually take time exponential to $|x|$, which isn't really a reasonable definition of efficiency. One of the practice problems is proving thata V is efficient with respect to the length of x.

From our definition of NP it should be clear that $\mathsf{P} \subseteq \mathsf{NP}$. After all, if a function F is computable in polynomial time, we can take V, our verifier, to disregard w and simply run F on its input. Then V will still run in polytime, as desired. The (literal) million dollar question is whether $\mathsf{P} = \mathsf{NP}$. It is strongly suspected (but not proven) that

$$P \overset{?}{\subsetneq} \mathsf{NP}$$

— that there are problems in NP that cannot be solved in polytime.

A brief note: NP does not stand for 'not polynomial' time! Rather it stands for nondeterministic polynomial time. This is because a nondeterministic Turing machine (or other computational model) can compute all functions in NP in polynomial time. It does this by 'guessing' a certificate, and then verifying the input with the certificate, which by definition takes polynomial time. So we see that this definition of NP is equivalent to the one above.

2.2 Problems

1. Give a simple argument for why $\mathsf{NP} \subseteq \mathsf{EXP}$ — consider how you can use the existence of the verifier G.
2. For each of the following, say whether the problem is in P, NP, is undecidable, or whether we don't know.
   (a) Given an integer x, find all the factors of x.
   (b) Given an undirected graph graph, determine whether it is possible to partition its vertices into two sets, with at least k edges crossing between sets.
   (c) Given a program Q, an input x, and a string 1^t, determine whether Q halts on x within t steps.
3. Define $F \in \mathsf{coNP}$ iff $\overline{F} \in \mathsf{NP}$, where $\overline{F}$ denotes the negation of the output of F (for example, if $F(00) = 1$, then $\overline{F}(00) = 0$). Prove that if $\mathsf{P} = \mathsf{NP}$, then $\mathsf{coNP} = \mathsf{NP}$.
4. Let $V : \{0,1\}^* \to \{0,1\}$ be defined as taking two inputs x, w such that there exists $a, b \in \mathbb{N}$ such that $w \in \{0,1\}^{a|x|^b}$. $V \in P$. Prove that $V \in TIME(|x|^c)$ for some c.

---

3 Universality and $3SAT$ — The Cook-Levin Theorem

The Cook-Levin Theorem states that

$$\forall F \in \mathsf{NP}, F \leq_{\mathrm{p}} 3SAT.$$

This is a very important result! It says that $3SAT$ is at least as hard as every problem in NP, or that solving $3SAT$ in polytime would allow us to solve any NP problem in polytime as well. This means that $3SAT$ is NP-hard.

Definition: $G : \{0,1\}^* \to \{0,1\}$ is NP-hard if $\forall F \in \mathsf{NP}$, $G \leq_p F$. A function $H : \{0,1\}^* \to \{0,1\}$ is NP-complete if it is NP-hard and itself in NP.

→ has a polynomial-time verifier

Therefore, from what we saw previously, $3SAT$ is also NP-complete (this is one way of stating the Cook-Levin Theorem).

3.1 Polynomial-Time Reductions

Recall that a function F is polynomially reducible to a function G is a polynomial-time algorithm for computing F that makes use of G. We write that $F \leq_{\mathrm{p}} G$. Reductions as such are helpful because they impliy that F is a problem that is "no harder" than G – because we know that there exists at least one way of computing F, which is to make use of G. To show that a problem G is NP-hard, it suffices to reduce a known NP-hard problem F to G, since solving G (in an efficient manner) implies that we can solve F too.

3.2 Problems

1. Given an undirected graph $G = (V, E)$, a clique is a subset $C \subseteq V$ such that $(v_1, v_2) \in E$ for all $v_1, v_2 \in C$. Consider the function $CLIQUE(G, k) = 1$ iff G has a clique of size k, and 0 otherwise. Show that $3SAT \leq_{\mathrm{p}} CLIQUE$, and that $CLIQUE$ is NP-complete.

2. Define $F \in \mathsf{coNP}$ iff $\overline{F} \in \mathsf{NP}$, where $\overline{F}$ denotes the negation of the output of F (for example, if $F(00) = 1$, then $\overline{F}(00) = 0$). Consider the following function $TAUTOLOGY$: if ϕ is a 3DNF formula (clauses of three 'and'ed variables, 'or'ed together), $TAUTOLOGY(\phi) = 1$ iff for all assignments x of the variables of ϕ, we have $\phi(x) = 1$. Otherwise $TAUTOLOGY(\phi) = 0$. Prove that $TAUTOLOGY$ is coNP-complete.

   We say $TAUTOLOGY$ is coNPcomplete if $TAUTOLOGY \in \mathsf{coNP}$ and $\forall F \in \mathsf{coNP}, TAUTOLOGY \leq_p F$. Hint: $3SAT$ is NP-complete. Try to relate the $3SAT$ problem to $TAUTOLOGY$.

3. Given n sets $S_1, S_2, \ldots, S_n$ such that

   $$\bigcup_{i=1}^{n} S_i = A$$

   the set cover of size k over these sets is a collection C of k of these sets such that

   $$\bigcup_{i \in C} S_i = A$$

   Given a collection of sets and an integer k, SET-COVER returns if there exists a valid set cover of a most size k over the given collection of sets. Prove that SET-COVER is NP-complete.

---

4 What if $\mathsf{P} = \mathsf{NP}$?

Here we go through a few consequences if $\mathsf{P} = \mathsf{NP}$.

4.1 Search-to-Decision

When considering a problem–such as cliques of size k in graph G–there are two relevant questions to ask. The first are decision problems, which ask if there is a solution. In this case, does there exist a clique of size k in graph G? The second type of problem is a search problem. In this case, the question would be, find a clique of size k in graph G if it exists.

An interesting results is that if $\mathsf{P} = \mathsf{NP}$ then search and decision problems become equally hard. That is, if we can solve a decision problem in polynomial time, we can solve the accompanying search problem in polynomial time too.

More formally, if $\mathsf{P} = \mathsf{NP}$, then for every polynomial-time algorithm V and $a, b \in \mathbb{N}$, there is a polynomial-time algorithm $FIND_V$ such that for every $x \in \{0,1\}^n$, if there exists $y \in \{0,1\}^{an^b}$ satisfying $V(xy) = 1$, then $FIND_V(x)$ finds some string y' satisfying this condition.

As with most proofs in this course, understanding the idea of the proof is far more important than memorizing all the details. As such, we'll give a more intuition-based explanation of the proof. The key insight is that we can define a function $STARTSWITH_V$ that on input $x \in \{0,1\}^*$ and $z \in \{0,1\}^l$ is 1 if and only if there exists a $y \in \{0,1\}^{a|x|^b}$ such that the first l bits of y are equal to z and $V(xy) = 1$. If $\mathsf{P} = \mathsf{NP}$ (we prove that this function is in NP in the practice problems), then we can search for the solution by asking if a solution exists starting with 0 or 1 (if neither are true then there is no solution), and dependent on that, pick one to add to our solution, and ask whether the next bit is a 0 or 1. This results in running a polynomial time algorithm a linear number of times, overall polynomial!

4.2 Optimization

Optimization (finding both max and argmax) for a polynomial time function also becomes polynomial time if $\mathsf{P} = \mathsf{NP}$.

In more formal terms, if $\mathsf{P} = \mathsf{NP}$, then for every polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}$ there is a polynomial-time algorithm OPT such that on input $x \in \{0,1\}$, $OPT(x, 1^m) = \max_{y \in \{0,1\}^m} f(x, y)$ (where we identify the output of $f(x)$ with a natural number via the binary representation).

The general idea of the proof involves creating a function that tells us if the optimal value is above k ($G(x, 1^m, k) = \exists_{y \in \{0,1\}^m} F(x, y) \geq k$). We can show this is in NP, and then we can use this to do a binary search on the possible values of k. Since F is polynomial, it can return a maximal value of $2^{p(n)}$, so a binary search takes $\log(2^{p(n)}) = p(n)$.

4.3 Problems

1. Prove that for $V \in P$ $STARTSWITH_V$ is in NP.

2. Using the optimization and search-to-decision results, prove that for any $F \in P$, we can compute $OPTARG(x, 1^m) = \text{argmax}_{y \in \{0,1\}^m} F(x, y)$ (again identifying the output of F with a natural number via the binary representation).