CS1200: Intro. to Algorithms and their Limitations Anshu & Vadhan

Lecture 21: The P vs. NP Problem

Harvard SEAS - Fall 2024 *Nov. 14, 2024*

1 Announcements

- SRE today
- PS8 coming out, due 11/20.

Recommended Reading:

- MacCormick §14.4, 14.6, 14.8

2 Search vs. Decision

Q: Does the seeming hardness of $\mathsf{NP}_{\mathsf{search}}$ come from the fact that there are many possible answers? What if we restrict to decision problems, with only two possible answers?

Definition 2.1. A computational problem $\Pi = (\mathcal{I}, \mathcal{O}, f)$ is a *decision problem* if $\mathcal{O} =$
and for every $x \in \mathcal{I}$, $|f(x)| =$

The choice of the names `yes` and `no` for the 2 elements of $\mathcal{O}$ is arbitrary, and other common choices are $\mathcal{O} = \{1, 0\}$ and $\mathcal{O} = \{\texttt{accept}, \texttt{reject}\}$. But it is convenient to standardize the names, since in the definition of NP below we will treat `yes` and `no` asymmetrically.

By definition,

$$\begin{aligned} \mathsf{P} &= \\ \mathsf{EXP} &= \end{aligned}$$

However, the decision class NP has a more subtle definition in terms of $\mathsf{NP}_{\mathsf{search}}$:

Definition 2.2 (NP)**.** A decision problem $\Pi = (\mathcal{I}, \{\texttt{yes}, \texttt{no}\}, f)$ is in NP if there is a computational problem $\Gamma = (\mathcal{I}, \mathcal{O}, g) \in \mathsf{NP}_{\mathsf{search}}$ such that for all $x \in \mathcal{I}$, we have:

$$\begin{aligned} f(x) = \{\texttt{yes}\} &\Leftrightarrow \\ f(x) = \{\texttt{no}\} &\Leftrightarrow \end{aligned}$$

Examples:

-
-

Another view of NP:

Pursuing this viewpoint, it turns out that there is a deep connection between mathematical proofs and NP, and this is one reason that the P vs. NP question is considered to be a central open problem in mathematics as well as computer science.

One nice feature of focusing on decision problems is that we can show that NP contains P (the class of decision problems solvable in polynomial time):

Lemma 2.3. $\mathsf{P} \subseteq \mathsf{NP}$.

Proof. Let $\Pi = (\mathcal{I}, \{\texttt{yes}, \texttt{no}\}, f)$ be an arbitrary computational problem in P. Our goal is to come up with a computational problem $\Gamma = (\mathcal{I}, \mathcal{O}, g)$ in $\mathsf{NP}_{\mathsf{search}}$ that satisfies the requirements of Definition 2.2. We do this by setting $g(x) = f(x) \cap \{\texttt{yes}\}$ and $\mathcal{O} = \{\texttt{yes}\}$.

Thus, $f(x) = \{\texttt{yes}\}$ iff $g(x) \neq \emptyset$, and it can be verified that $\Gamma \in \mathsf{NP}_{\mathsf{search}}$. (The verifier $V(x, y)$ for Γ can check that $y = \texttt{yes}$ and that the polynomial-time algorithm for Π outputs $\texttt{yes}$ on x.) Thus, we conclude that $\Pi \in \mathsf{NP}$. □

In contrast, as we have commented earlier (and you will show on ps8), $\mathsf{P}_{\mathsf{search}}$ is not a subset of $\mathsf{NP}_{\mathsf{search}}$, since $\mathsf{NP}_{\mathsf{search}}$ requires that *all* solutions are easy to verify, whereas $\mathsf{P}_{\mathsf{search}}$ only tells us that at least one of the solutions is easy to find. There may be solutions that are too long or even undecidable to verify. On the other hand, P tells us that there is only one solution, so the above subtlety does not arise.

The "P vs. NP Question" is usually formulated as asking whether $\mathsf{P} = \mathsf{NP}$ (with the answer widely conjectured to be no).

It turns out that search and decision versions of the P vs. NP question are equivalent:

Theorem 2.4 (Search vs. Decision)**.** *$\mathsf{NP} = \mathsf{P}$ if and only if $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$.*

Proof of Theorem 2.4. Suppose that $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$.

For the converse, assume that $\mathsf{P} = \mathsf{NP}$. Since SAT-Decision is in NP, it is also in P and hence in $\mathsf{P}_{\mathsf{search}}$. By Lemma 2.5 below, SAT $\leq_p$ SAT-Decision, so SAT is also in $\mathsf{P}_{\mathsf{search}}$. Since SAT is $\mathsf{NP}_{\mathsf{search}}$-complete, we have $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$.

□

Lemma 2.5. *SAT $\leq_p$ SAT-Decision.*

Proof sketch. The idea is to find a satisfying assignment one variable at a time, using the SAT-Decision oracle to determine if we should set x_i to be 0 or 1.

```
1 R(φ) :
  Input    : A CNF formula $\varphi(x_0,\ldots,x_{n-1})$ (and access to an oracle $O$ solving
             SAT-Decision)
  Output   : A satisfying assignment $\alpha$ to $\varphi$, or $\perp$ if none exists.
2 if $O(\varphi) =$ no then return $\perp$;
3 foreach $i = 0,\ldots,n-1$ do
4     if $O($                                  $) =$ yes then $\alpha_i = 0$;
5     else $\alpha_i = 1$;
6 return $\alpha = (\alpha_0,\ldots,\alpha_{n-1})$
```

□

In most textbooks, the theory of NP-completeness focuses on decision problems (and names like SAT, k-Coloring, etc. typically refer to the decision versions). As expected, we say a decision problem Π is NP-*complete* if $\Pi \in \mathsf{NP}$, and Π is NP-hard, meaning $\Gamma \leq_p \Pi$ for every problem $\Gamma \in \mathsf{NP}$. All of the $\mathsf{NP}_{\mathsf{search}}$-completeness proofs we have seen are also NP-completeness proofs of the corresponding decision problems:

Theorem 2.6. *SAT-Decision, 3-SAT-Decision, IndependentSet-Decision, SubsetSum-Decision, LongPath-Decision, and 3D-Matching-Decision are* NP*-complete.*

For NP-completeness proofs as in the above theorem, mapping reductions become even simpler; we only need a polynomial-time algorithm R that transforms `yes` instances to `yes` instances, and `no` instances to `no` instances. We don't need the algorithm S that maps solutions to the search problem on $R(x)$ back to solutions to the search problem on x.

3 The Breadth of NP-completeness.

There is a huge variety of NP-complete problems, from many different domains:

The fact that they are all NP-complete means that, even though they look different, there is a sense in which they are really all the same problem in disguise. And they are equivalent in complexity: either they are all easy (solvable in polynomial time) or they are all hard (not solvable in polynomial time). The widely believed conjecture is the latter; $\mathsf{P} \neq \mathsf{NP}$. The lack of polynomial-time algorithms indicates that these problems have a mathematical nastiness to them; we shouldn't expect to find nice characterizations or "closed forms" for solutions (as such characterizations would likely lead to efficient algorithms).

4 Two Possible Worlds

If $\mathsf{P} = \mathsf{NP}$, then:

If $\mathsf{P} \neq \mathsf{NP}$, then: