CS120: Intro. to Algorithms and their Limitations	Hesterberg & Vadhan

Lecture 21: The P vs. NP Problem

Harvard SEAS - Fall 2022 *Nov. 15, 2022*

1 Announcements

Recommended Reading:

- MacCormick §14.4, 14.6, 14.8

2 Search vs. Decision

The theory of NP-completeness is usually presented (including in the MacCormick text) as focusing on decision problems. Here we discuss that formulation and its relation to what we have discussed about search problems.

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 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 $\mathsf{NP}_{\mathsf{search}}$: NP consists of the problems that amount to deciding whether an instance of an $\mathsf{NP}_{\mathsf{search}}$ problem has a solution or not. Formally:

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 sketch. Let $\Pi = (\mathcal{I}, \{\texttt{yes}, \texttt{no}\}, f)$ be an arbitrary computational problem in P. Then define $\Gamma = (\mathcal{I}, \{\texttt{yes}\}, g)$ by $g(x) = f(x) \cap \{\texttt{yes}\}$.

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 accepts x.) Thus, we meet the requirements of Definition 2.2 and conclude that $\Pi \in \mathsf{NP}$. □

In contrast, as we have commented earlier (and you may show on ps9), $\mathsf{P}_{\mathsf{search}}$ 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 (but there may be others that are hard or even undecidable to verify).

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* $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$.

Q: Does this theorem remind you of anything you've seen?

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

For the converse, assume that $\mathsf{P} = \mathsf{NP}$. Then,

□

The use of IndependentSet in the above proof is not crucial, and the same can be proven using other $\mathsf{NP}_{\mathsf{search}}$-complete problems, such as SAT:

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

Proof sketch. The idea is to find a satisfying assignment one variable at a time, using the Satisfiability-Decision oracle to determine whether setting $x_i = 0$ or $x_i = 1$ preserves satisfiability.

```
R(φ) :
  Input  : A CNF formula φ(x_0, ..., x_{n−1}) (and access to an oracle O solving
           Satisfiability-Decision)
  Output : A satisfying assignment α to φ, or ⊥ if none exists.
if O(φ) = no then return ⊥;
foreach i = 0, ..., n − 1 do
  | if O(φ(α_0, ..., α_{i−1}, 0, x_{i+1}, ..., x_{n−1})) = yes then α_i = 0;
  | else α_i = 1;
return α = (α_0, ..., α_{n−1})
```

□

In most textbooks, the theory of NP-completeness focuses on decision problems. In that case, 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: