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
- To fill in

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 $\mathcal{O} = \{\texttt{yes}, \texttt{no}\}$ and for every $x \in \mathcal{I}$, $|f(x)| = 1$.

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,

$$\begin{aligned} \mathsf{P} &= \{\Pi : \Pi \in \mathsf{P}_{\mathsf{search}} \text{ and } \Pi \text{ is a decision problem}\} \\ \mathsf{EXP} &= \{\Pi : \Pi \in \mathsf{EXP}_{\mathsf{search}} \text{ and } \Pi \text{ is a decision problem}\}. \end{aligned}$$

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:

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

Examples:

- Given 3-CNF formula φ, is φ satisfiable?
- Given a graph G and a number k, does G have an independent set of size k?

Another view of NP: decision problems Π where a `yes` answer has a short, efficiently verifiable proof. Indeed, we can prove that $f(x) = \{\texttt{yes}\}$ by giving a solution $y \in g(x)$, which is of at most polynomial length and is verifiable in polynomial time.

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? On Problem Set 5, you showed the equivalence of IndependentSet-ThresholdSearch and IndependentSet-ThresholdDecision.

Proof of Theorem 2.4. Suppose that $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$. Then, $\mathsf{P} = \mathsf{NP}$ because a polynomial-time algorithm that solves a search problem Π can be converted into a polynomial-time algorithm that solves Π-decision by replacing every non-$\perp$ output with `yes` and every $\perp$ output with `no`.

For the converse, assume that $\mathsf{P} = \mathsf{NP}$. Then, Since IndependentSet-ThresholdDecision in NP, we have that IndependentSet-ThresholdDecision is also in P. By Problem Set 5, we have IndependentSet-ThresholdSearch $\leq_p$ IndependentSet-ThresholdDecision, so IndependentSet-ThresholdSearch is in $\mathsf{P}_{\mathsf{search}}$. Last time, we proved that IndependentSet-ThresholdSearch is $\mathsf{NP}_{\mathsf{search}}$-complete, so we conclude that $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$.

□

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.

```
1 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.
2 if O(φ) = no then return ⊥;
3 foreach i = 0, ..., n − 1 do
4 |   if O(φ(α_0, ..., α_{i−1}, 0, x_{i+1}, ..., x_{n−1})) = yes then α_i = 0;
5 |   else α_i = 1;
6 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:

- SAT, 3SAT
- IndependentSet
- 3-D Matching
- SubsetSum
- 3-Coloring
- LongPath
- ProgrammingTeam
- Problems from economics: Combinatorial Auctions
- Problems from biology: Protein Folding
- Problems from math: Finding short proofs of theorems!

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:

- Searching for solutions is never much harder than verifying solutions.
- Optimization is easy.
- Finding mathematical proofs is easy.
- Breaking cryptography is easy.
- Machine learning is easy.
- Every problem in NP is NP-complete (ps9).

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

- None of the NP-complete problems have (worst-case) polynomial-time algorithms. Have to settle for superpolynomial-time algorithms, heuristics that perform well on average/real-world instances (like SAT solvers), or approximation algorithms (which don't necessarily find optimal solutions).
- There are problems in NP that are neither NP-hard nor in P, and similarly for search problems. Natural candidates: Factoring, and finding Nash Equilibria of 2-player games.
- There is *hope* for secure cryptography (but this seems to require assumptions stronger than $\mathsf{P} \neq \mathsf{NP}$).