CS120: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Lecture 25: The P vs. NP Problem

Harvard SEAS - Fall 2021 *Dec. 2, 2021*

1 Announcements

Recommended Reading:

- MacCormick §14.4, 14.6, 14.8
- Salil OH after class
- Review session next Tuesday 9:45-11

2 The Breadth of NP-completeness.

There is a huge variety of $\mathsf{NP}_{\mathsf{search}}$-complete problems, from many different domains:

- SAT, 3SAT
- IndependentSet, 3-Coloring
- SubsetSum
- LongPath
- ProgrammingTeam
- 3-dimensional Matching (like bipartite matching, but where we match triples like donor-patient-hospital)
- Problems from economics: Combinatorial Auctions
- Problems from biology: Protein Folding
- Problems from math: Finding short proofs of theorems!

The fact that they are all $\mathsf{NP}_{\mathsf{search}}$-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; $\mathsf{NP}_{\mathsf{search}} \not\subseteq \mathsf{P}_{\mathsf{search}}$. 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).

3 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 3.1. Let Π be a $\perp$-proper computational problem. Then Π-*decision* is the computational problem $(\mathcal{I}, g)$ such that for all $x \in \mathcal{I}$, we have:

$$\begin{aligned} f(x) \neq \{\perp\} &\Rightarrow g(x) = \{\texttt{accept}\} \\ f(x) = \{\perp\} &\Rightarrow g(x) = \{\texttt{reject}\} \end{aligned}$$

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?

We remark that the same decision problem Γ can arise from several different search problems Π, some of which might be easier than others. For example: consider the Factoring search problem $\Pi = (\mathcal{I}, f)$, where $= \{x \in \mathbb{N} : x \geq 2\}$ and

$$f(x) = \{(p_1, p_2, \ldots, p_k) : k \geq 1, p_0, p_1, \ldots, p_{k-1} \text{ prime numbers s.t. } x = p_0 p_1 \cdots p_{k-1}\}.$$

Then Π-decision is a trivial problem where the answer is always `accept`, since every integer $x \geq 2$ has a prime factorization. And Π-decision $= \Pi'$-decision where $\Pi' = (\mathcal{I}, f')$ is a trivial problem where $f'(x) = \{\varepsilon\}$ for all x.

Definition 3.2. $\mathsf{NP} = \{\Gamma : \Gamma = \Pi\text{-decision for some } \Pi \in \mathsf{NP}_{\mathsf{search}}\}$.

NP stands for "nondeterministic polynomial time," due to an alternate characterization of NP in terms of "nondeterministic" algorithms (which you can find described in the MacCormick text).

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 3.3. $\mathsf{P} \subseteq \mathsf{NP}$.

In contrast, $\mathsf{P}_{\mathsf{search}}$ 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).

Proof of Lemma 3.3. Let $\Gamma = (\mathcal{I}, g)$ be a decision problem in P. Let $\Pi = (\mathcal{I}, f)$ be the search problem given by:

$$\begin{aligned} g(x) = \{\texttt{accept}\} &\Rightarrow f(x) = \{\varepsilon\} \\ g(x) = \{\texttt{reject}\} &\Rightarrow f(x) = \{\perp\} \end{aligned}$$

By inspection, $\Gamma = \Pi$-decision, so we just need to verify that $\Pi \in \mathsf{NP}_{\mathsf{search}}$. The solutions to Π are certainly polynomially bounded (they are all either ε or $\perp$), so we just need to give a polynomial-time verifier V. Since $\Gamma \in \mathsf{P}$, there is a polynomial-time algorithm A solving Γ. We can use A to design our verifier V for Π as follows:

$$V(x,y) = \begin{cases} \texttt{accept} & \text{if } (A(x) = \texttt{accept} \text{ and } y = \varepsilon) \text{ or } (A(x) = \texttt{reject} \text{ and } y = \perp) \\ \texttt{reject} & \textit{otherwise} \end{cases}$$

□

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

Theorem 3.4 (Search vs. Decision)**.** $\mathsf{NP} = \mathsf{P}$ *if and only if* $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$.

Which is the easier direction of Theorem 3.4? If we can find solutions quickly, then we can determine if there is a solution quickly, so the "only if" direction is direct.

We illustrate the other direction with the special case of Satisfiability:

Lemma 3.5. *If Satisfiability-Decision is in* P *then Satisfiability is in* $\mathsf{P}_{\mathsf{search}}$.

Proof sketch. We give a reduction, showing that Satisfiability $\leq_p$ Satisfiability-Decision. 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) = reject 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})) = accept then α_i = 0;
5 |   else α_i = 1;
6 return α = (α_0, ..., α_{n−1})
```

□

Now we can complete the proof of Theorem 3.4.

Proof of Theorem 3.4. 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 `accept` and every $\perp$ output with `reject`.

For the converse, assume that $\mathsf{P} = \mathsf{NP}$. Then Satisfiability-Decision is in P, so by Lemma 3.5, Satisfiability is in $\mathsf{P}_{\mathsf{search}}$. By $\mathsf{NP}_{\mathsf{search}}$-completeness of Satisfiability, we conclude that $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$. □

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 (ps11).

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

- None of the NP-complete problems have (worst-case) polynomial-time algorithms. Have to settle for exponential-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 $\mathsf{P} \neq \mathsf{NP}$).