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
- To fill in

2 SRE Takeaways

- Previously, we have seen an example of a reduction of 3-SAT to a graph problem using variable and clause gadgets (IndependentSet). VectorSubsetSum is an example of a more numerical problem that is also NP-complete; this SRE shows that a variety of different kinds of problems can be shown to be $\mathsf{NP}_{\mathsf{search}}$-complete via a reduction from 3-SAT.
- To show a problem is $\mathsf{NP}_{\mathsf{search}}$-hard, it suffices to reduce from an $\mathsf{NP}_{\mathsf{search}}$-hard problem.

3 Search vs. Decision

In Lecture 19, we saw that $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{EXP}_{\mathsf{search}}$ using an exhaustive search algorithm for any computational problem in $\mathsf{NP}_{\mathsf{search}}$, which had runtime exponential in the (polynomial) length of valid answers. One might wonder from this whether the seeming hardness of $\mathsf{NP}_{\mathsf{search}}$ comes from the fact that there are many possible answers on a given instance. This motivates studying analogues of $\mathsf{NP}_{\mathsf{search}}$ for *decision problems*, where there are only two possible answers. This is fact how the theory of NP-completeness is usually presented (including in the MacCormick and Sipser texts). Here we discuss that formulation and its relation to what we have discussed about search problems.

Definition 3.1. A computational problem $\Pi = (\mathcal{I}, \mathcal{O}, f)$ is a *decision problem* if $\mathcal{O} = \{\mathtt{yes}, \mathtt{no}\}$ 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 $\mathcal{O} = \{\mathtt{accept}, \mathtt{reject}\}$. But it is convenient to standardize the names, since in the definition of NP below we will treat `yes` and `no` asymmetrically.

By definition,

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

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

Definition 3.2 (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:

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

Examples:

- SAT-Decision: Given CNF formula φ, is φ satisfiable?
- GraphColoring-Decision: Given a graph G and a number k, does G have an independent set of size at least k?

Another view of NP: decision problems Π where a $\texttt{yes}$ 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 3.3. $\mathsf{P} \subseteq \mathsf{NP}$.

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 3.2. 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 $\Pi \in \mathsf{NP}$. $\square$

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

Proof of Theorem 3.4. Suppose that $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$. For any $\Pi = (\mathcal{I}, \{\texttt{yes}, \texttt{no}\}, f) \in \mathsf{NP}$, let $\Gamma = (\mathcal{I}, \mathcal{O}, g) \in \mathsf{NP}_{\mathsf{search}}$ be as in Definition 3.2. By assumption, $\Gamma \in \mathsf{P}_{\mathsf{search}}$ so there is a polynomial-time algorithm solving Γ. This can be converted into a polynomial-time algorithm that solves Π by replacing every non-$\perp$ output with $\texttt{yes}$ and every $\perp$ output with $\texttt{no}$. This shows that $\Pi \in \mathsf{P}$.

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

$\square$

Lemma 3.5. *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 whether setting x_i to be 0 or 1 preserves satisfiability.

```
1 $R(\varphi)$ :
  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 $O(\varphi(\alpha_0, \ldots, \alpha_{i-1}, 0, x_{i+1}, \ldots, x_{n-1})) = yes$ 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 3.6. *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.

4 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).

5 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 (for theorems that have short mathematical proofs) is easy.
- Breaking cryptography is easy.
- Machine learning is easy.
- Every problem in NP is NP-complete.

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 $\mathsf{P} \neq \mathsf{NP}$).