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

Lecture 19: NP and NP-completeness

Harvard SEAS - Fall 2024 *2024-11-07*

1 Announcements

- Salil OH 11-12pm; Anurag zoom OH Fri 1:30-2:30 pm
- Next SRE moved to *Thursday* 11/14.

Recommended Reading:

- MacCormick §12.0–12.3, Ch. 13

2 Polynomial-Time Reductions

Definition 2.1. For computational problems Π and Γ, we write $\Pi \leq_p \Gamma$ if there is a *polynomial-time* reduction R from Π to Γ. That is, there is a constant $c \geq 0$ such that R runs in time at most $O(N^c)$ on inputs of length N, counting oracle calls as one time step. Equivalently, there is a constant d such that $\Pi \leq_{O(N^d),O(N^d)\times O(N^d)} \Gamma$.

Some examples of polynomial-time reduction that we've seen include:

- 3-Coloring $\leq_p$ SAT (Lecture 15)
- LongPath $\leq_p$ SAT (SRE 5)
- IntervalScheduling-Decision $\leq_p$ Sorting (Lecture 4). In this case a simpler polynomial time reduction is to solve the IntervalScheduling-Decision in time $O(n^2)$, call the Sorting oracle on some input and then ignore the oracle's output. What made the reduction from Lecture 4 useful that it ran in *linear* time (so not obvious how to solve without sorting), something that is lost by only referring to it as a polynomial-time reduction.

Using polynomial-time reductions to compare problems fits nicely with the study of the classes $\mathsf{P}_{\mathsf{search}}$ and P, since they are "closed" under such reductions:

Lemma 2.2. *Let Π and Γ be computational problems such that $\Pi \leq_p \Gamma$. Then:*

1. *If $\Gamma \in \mathsf{P}_{\mathsf{search}}$, then $\Pi \in \mathsf{P}_{\mathsf{search}}$.*
2. *If $\Pi \notin \mathsf{P}_{\mathsf{search}}$, then $\Gamma \notin \mathsf{P}_{\mathsf{search}}$.*

Proof. 1. Since $\Pi \leq_p \Gamma$, we have $\Pi \leq_{T_R,q\times h} \Gamma$ for $T_R(N), q(N), h(N) = O(N^d)$ for some constant d. Suppose that $\Gamma \in \mathsf{P}_{\mathsf{search}}$, i.e. Γ can be solved in time $T_\Gamma(N) = O(N^b)$ for some $b \geq 0$. Then by Lemma 3.2 from Lecture 4 (restated in Lection 17), Π can be solved in time

$$O\left(T_R(N) + q(N) \cdot T_\Gamma(h(N))\right) = O\left(N^d + N^d \cdot \left(N^d\right)^b\right) = O\left(N^{d\cdot(b+1)}\right),$$

So $\Pi \in \mathsf{P}_{\mathsf{search}}$.

2. Contrapositive of Item 1

□

This lemma means that we can use polynomial-time reductions both positively—to show that problems are in $\mathsf{P}_{\mathsf{search}}$— and negatively—to give evidence that problems are not in $\mathsf{P}_{\mathsf{search}}$. For example, under the *assumption* that 3-Coloring is not in $\mathsf{P}_{\mathsf{search}}$, it follows that SAT is not in $\mathsf{P}_{\mathsf{search}}$, by the above lemma and the fact that 3-Coloring $\leq_p$ SAT (SRE5). As always, *the direction of the reduction is crucial!*

Another very useful feature of polynomial-time reductions is that they compose with each other:

Lemma 2.3. *If* $\Pi \leq_p \Gamma$ *and* $\Gamma \leq_p \Theta$ *then* $\Pi \leq_p \Theta$.

This follows from Problem 2 in Problem Set 2, and then using the definition of polynomial time reduction.

3 NP

Figure 1: Can you find a cat?

Roughly speaking, $\mathsf{NP}_{\mathsf{search}}$ consists of the computational problems where valid outputs can be *verified* in polynomial time. This is a very natural requirement; what's the point in searching for something if we can't recognize when we've found it?

Definition 3.1. A computational problem $\Pi = (\mathcal{I}, \mathcal{O}, f)$ is in $\mathsf{NP}_{\mathsf{search}}$ if the following conditions hold:

1. All valid outputs are of polynomial length: There is a polynomial p such that for every $x \in \mathcal{I}$ and every $y \in f(x)$, we have $|y| \leq p(|x|)$, where $|z|$ denotes the bitlength of z.

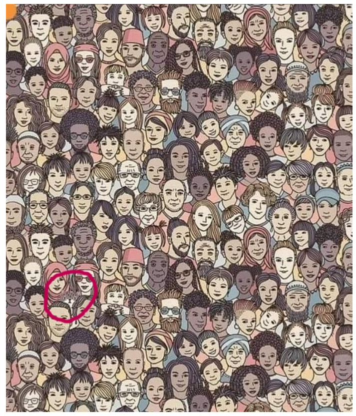


Figure 2: Can you verify that the cat is in the red circle?

2. All valid outputs are verifiable in polynomial time: There's a polynomial-time verifier V that, given $x \in \mathcal{I}$ and a potential output y,[1] decides whether $y \in f(x)$.

(Remark on terminology: $\mathsf{NP}_{\mathsf{search}}$ is often called FNP in the literature, and is closely related to, but slightly more restricted than, the class $\mathsf{PolyCheck}$ defined in the MacCormick text.)

Examples:

1. Satisfiability:
$$\mathcal{I} = \{\text{Boolean formulas } \varphi(z_1, \ldots, z_n), n \in \mathbb{N}\}$$
$$\mathcal{O} = \{\text{Assignments } \alpha \in \{0,1\}^n, n \in \mathbb{N}\}$$
$$f(\varphi) = \{\alpha : \varphi(\alpha) = 1\}$$
We can verify if a potential assignment α' satisfies φ in polynomial time by (a) checking that α' is indeed a valid assignment (i.e. an array of 0's and 1's), and (b) substituting α' into φ and checking whether $\varphi(\alpha') = 1$. Note that $|\alpha'| = n \leq |\varphi|$ so the solutions are of polynomial length.

2. GraphColoring:
$$f(G,k) = \{c : V \to [k] \text{ a proper } k \text{ coloring}\}$$
Our verifier takes in G, k and $c' : V \to [k]$ and checks that for every edge (u,v), $c'(u) \neq c'(v)$, which runs in time $O(m)$. Equivalently, we can check that every color class defines an independent set. Furthermore, $|c'| = n\lceil \log k \rceil \leq |(G,k)|^2$, so the solution is not too long.

3. IndependentSet-ThresholdSearch:
$$f(G,k) = \{S \subset V : |S| \geq k, \text{ S is an independent set}\}$$
Verifier takes G, k and S' and checks that for every pair of vertices $u, v \in S'$, there is no edge between u, v. Further, it checks that $|S'| \geq k$.

[1] Note that we do not assume $y \in \mathcal{O}$, so the verifier should reject if $y \notin \mathcal{O}$, i.e. y is ill-formed.

Potential non-example:

1. IndependentSet-OptimizationSearch:

$$f(G) = \{S \subseteq V : S \text{ is an independent set in } G \text{ of maximum } \textit{size}\}$$

Even though this problem does not appear to be in $\mathsf{NP}_{\mathsf{search}}$ (its still an open question in the theory of computing!), it reduces in polynomial time to IndependentSet-ThresholdSearch, which is in $\mathsf{NP}_{\mathsf{search}}$ (to be discussed next week in the course).

The following proposition shows that every problem in $\mathsf{NP}_{\mathsf{search}}$ can be solved in exponential time.

Proposition 3.2. $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{EXP}_{\mathsf{search}}$.

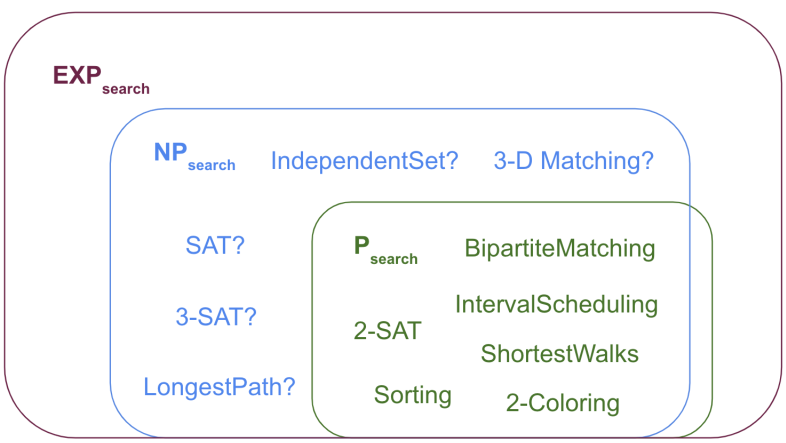
Proof.
Exhaustive search! We can enumerate over all possible solutions and check if any is a valid solution.

ExhaustiveSearch
Input : $x \in \mathcal{I}$
for $y \in \mathcal{O}$ *such that* $|y| \leq p(|x|)$ **do**
  if $V(x, y) = 1(\textit{accept})$ **then**
    return y
return $\perp$

This has runtime $O(2^{p(n)} \cdot (n + p(n))^c)$ which is bounded by the exponential $O(2^{n^d})$, where $d = \deg(p) + 1$.

$\square$

So now our diagram of complexity classes looks like this:



Remarks:

- **$\mathsf{P}_{\mathsf{search}}$ vs $\mathsf{NP}_{\mathsf{search}}$:** Somewhat counterintuitively, $\mathsf{P}_{\mathsf{search}} \not\subseteq \mathsf{NP}_{\mathsf{search}}$. This due to artificial examples that you may see later in the course, but most of the natural problems in $\mathsf{P}_{\mathsf{search}}$ are also in $\mathsf{NP}_{\mathsf{search}}$ (like all of the green problems in the above diagram).
- **Class NP:** Every problem in $\mathsf{NP}_{\mathsf{search}}$ has a corresponding decision problem (deciding whether or not there is a solution). The class of such decision problems is called NP. We will discuss the class NP more next week.

We still have question marks next to all of the blue problems; we don't know whether they (and thousands of other important problems in $\mathsf{NP}_{\mathsf{search}}$) are in $\mathsf{P}_{\mathsf{search}}$ or not. We will now try to get a handle on these questions.

4 $\mathsf{NP}_{\mathsf{search}}$-Completeness

Unfortunately, although it is widely conjectured, we do not know how to prove that $\mathsf{NP}_{\mathsf{search}} \not\subseteq \mathsf{P}_{\mathsf{search}}$. As we will see next week, this is an equivalent formulation of the famous P vs. NP problem, considered one of the most important open problems in computer science and mathematics. However, even without resolving the P vs. NP conjecture, we can give strong evidence that problems are not solvable in polynomial time by showing that they are $\mathsf{NP}_{\mathsf{search}}$-*complete*:

Definition 4.1 (NP-completeness, search version)**.** A problem Γ is $\mathsf{NP}_{\mathsf{search}}$-*complete* if:

1. Γ is in $\mathsf{NP}_{\mathsf{search}}$
2. Γ is $\mathsf{NP}_{\mathsf{search}}$-*hard*: For every computational problem $\Pi \in \mathsf{NP}_{\mathsf{search}}$, $\Pi \leq_p \Gamma$.

We can think of the NP-complete problems as the "hardest" problems in NP. Indeed:

Proposition 4.2. *Suppose* Γ *is* $\mathsf{NP}_{\mathsf{search}}$*-complete. Then* $\Gamma \in \mathsf{P}_{\mathsf{search}}$ *iff* $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$.

Proof. We first show that if $\Gamma \in \mathsf{P}_{\mathsf{search}}$, then $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$. For every problem $\Pi \in \mathsf{NP}_{\mathsf{search}}$, we have that $\Pi \leq_p \Gamma$. Lemma 2.2 now ensures that $\Pi \in \mathsf{P}_{\mathsf{search}}$. Thus, $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$.

On the other hand, if $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$, then $\Gamma \in \mathsf{P}_{\mathsf{search}}$, using the fact that $\Gamma \in \mathsf{NP}_{\mathsf{search}}$. This completes the proof. $\square$

In other words, if any $\mathsf{NP}_{\mathsf{search}}-$complete problem is in $\mathsf{P}_{\mathsf{search}}$, then all problems in $\mathsf{NP}_{\mathsf{search}}$ are in $\mathsf{P}_{\mathsf{search}}$. Remarkably, there are natural NP-complete problems. The first one is CNF-Satisfiability:

Theorem 4.3 (Cook–Levin Theorem)**.** *SAT is* $\mathsf{NP}_{\mathsf{search}}$*-complete.*

This can be interpreted as strong evidence that SAT is not solvable in polynomial time. If it were, then *every* problem in $\mathsf{NP}_{\mathsf{search}}$ would be solvable in polynomial time. We will return to a proof of the Cook–Levin Theorem later in the course.

5 More $\mathsf{NP}_{\mathsf{search}}$-complete Problems

Once we have one $\mathsf{NP}_{\mathsf{search}}$-complete problem, we can get others via reductions from it. Consider the computational problem 3-SAT, which is obtained when we restrict the number of literals in each clause of SAT.

Input	**:** A CNF formula φ on n variables $z_0, \dots z_{n-1}$ in which each clause has width at most 3 (i.e. contains at most 3 literals)
Output	**:** An $\alpha \in \{0,1\}^n$ such that $\varphi(\alpha) = 1$ (if one exists)

Computational Problem 3-SAT

Theorem 5.1. *3-SAT is* $\mathsf{NP}_{\mathsf{search}}$*-complete.*

Proof. The full proof is deferred to Lecture 20. The proof follows in two steps.

1. 3SAT is in $\mathsf{NP}_{\mathsf{search}}$: Our verifier can check if an assignment α satisfies the 3CNF formula (the same verifier as for SAT).

2. 3SAT is $\mathsf{NP}_{\mathsf{search}}$-hard: Since every problem in $\mathsf{NP}_{\mathsf{search}}$ reduces to SAT (Theorem 4.3), all we need to show is SAT $\leq_p$ 3SAT (since reductions compose - Lemma 2.3).

The reduction algorithm from SAT to 3SAT has the following components (Figure 3). First, we give an algorithm R which takes a SAT instance φ to a 3SAT instance φ'.

$$\text{SAT instance } \varphi \xrightarrow{\text{polytime R}} \text{3SAT instance } \varphi'$$

Then we feed the instance φ' to our 3SAT oracle and obtain a satisfying assignment β to φ' or $\perp$ if none exists. If we get $\perp$ from the oracle, we return $\perp$, else we transform β into a satisfying assignment to φ using another algorithm S.

$$\text{SAT assignment } \alpha \xleftarrow{\text{polytime S}} \text{3SAT assignment } \beta$$

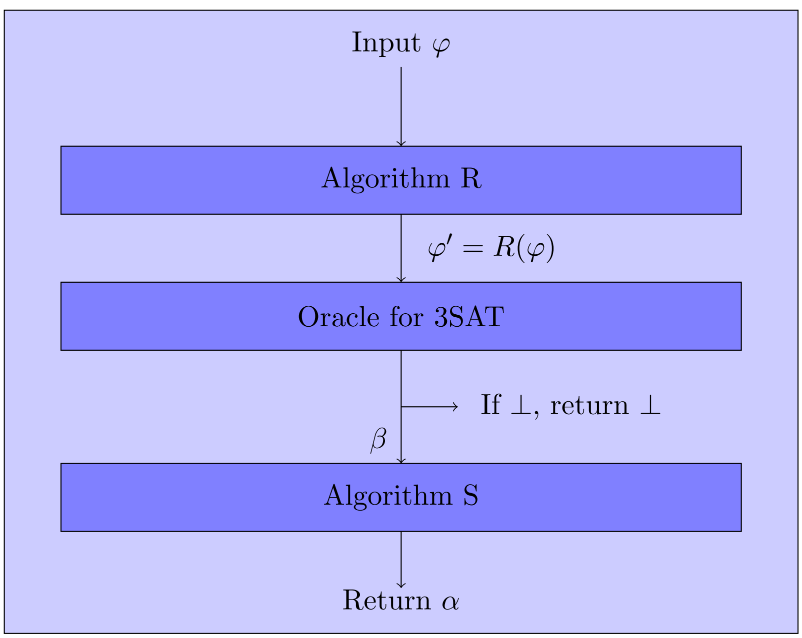


Figure 3: Reduction algorithm from SAT to 3SAT.

$\square$