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

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.

2.

Proof. 1.

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:

2. All valid outputs are verifiable in polynomial time:

(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:

2. GraphColoring:

3. IndependentSet-ThresholdSearch:

Potential non-example:

1. IndependentSet-OptimizationSearch:

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).

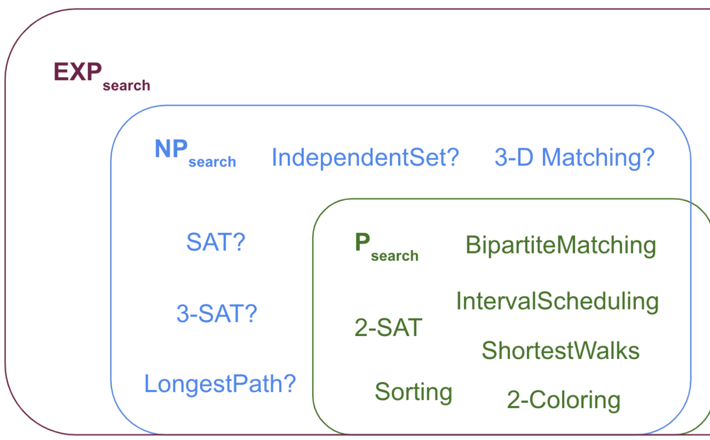
The following proposition shows that every problem in $\mathsf{NP}_{\mathsf{search}}$ can be solved in exponential time.

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

Proof.

□

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.

2.

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. □

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 $z_0, \ldots z_{n-1}$ 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 $\mathsf{NP}_{\mathsf{search}}$:

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 2). 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.

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

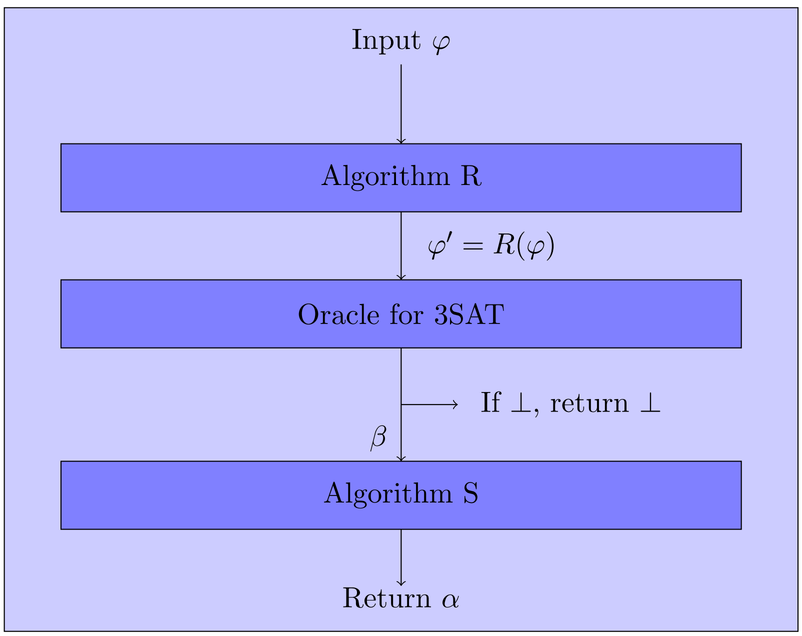


Figure 2: Reduction algorithm from SAT to 3SAT.

$\square$